

Theory of Computation

(Code : 314441)

Semester V - Information Technology
(Savitribai Phule Pune University)

**Strictly as per the New Credit System Syllabus (2015 Course)
Savitribai Phule Pune University w.e.f. academic year 2017-2018**

Dilip Kumar Sultania

B.Tech.(hons.) Computer Science and Engineering
I.I.T, Kharagpur.



PO274A



Theory of Computation

Dilip Kumar Sultania

(Semester V, Information Technology, SPPU)

Copyright © by Tech-Max Publications. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Printed in India : July 2010 (For Pune University)

First Edition : June 2017

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : Image Offset, Dugane Ind. Area Survey No. 28/25, Dhayari Near Pari Company, Pune – 41,
Maharashtra State, India. E-mail : rahulshahimage@gmail.com

ISBN 978-93-5224-581-9

Published by

Tech-Max Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India

Ph : 91-20-24225065, 91-20-24217965. Fax 020-24228978.

Email : info@techmaxbooks.com, Website : www.techmaxbooks.com

[314441] (FID : TP481) (Book Code : PO274A)

Preface

My dear students,

I am extremely happy to come out with this edition of "Theory of Computation" for you, the Information Technology students. I have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

A large number of examples have been included, so that the book will cater for all your needs.

I am thankful to Shri. Sachin Shah and Shri. Arunoday Kumar for the encouragement and support that they have extended. I am also thankful to all members of Tech-Max Publications and others for their efforts to make this book as good as it is. We have jointly made every possible effort to eliminate all the errors in this book. However if you find any, please let us know, because that will help me to improve further.

I am also thankful to my family members and friends for patience and encouragement.

- Dilip Kumar Sultania



Syllabus

314441 : Theory of Computation

| Teaching Scheme | Credit | Examination Scheme |
|--------------------------|--------|-------------------------|
| Lectures : 04 Hours/Week | 04 | In-Semester : 30 Marks |
| | | End-Semester : 70 Marks |

Prerequisite

1. Discrete Structures.
2. Data structures and problem solving.

Course Objectives

1. To understand problem classification and problem solving by machines.
2. To understand the basics of automata theory and its operations.
3. To study computing machines by describing, classifying and comparing different types of computational models.
4. Encourage students to study theory of computability and complexity.
5. To understand the P and NP class problems and its classification.
6. To understand the fundamentals of problem decidability and reducibility.

Course Outcomes

1. To construct finite state machines to solve problems in computing.
2. To write mathematical expressions for the formal languages
3. To apply well defined rules for syntax verification.
4. To construct and analyze Push Down, Post and Turing Machine for formal languages.
5. To express the understanding of the decidability and non-decidability problems.
6. To express the understanding of computational complexity.

Course Contents

Unit I : Finite State Machines

(08 Hours)

Basic Concepts : Symbols, Strings, Language, Formal Language, Natural Language, Basic Machine and Finite State Machine.

FSM without output : Definition and Construction-DFA, NFA, NFA with epsilon-Moves, Minimization Of FA, Equivalence of NFA and DFA, Conversion of NFA with epsilon moves to NFA, Conversion of NFA With epsilon moves to DFA.

FSM with output : Definition and Construction of Moore and Mealy Machines, Inter-conversion between Moore and Mealy Machines. (Refer Chapters 1 and 2)

Unit II : Regular Expressions

(08 Hours)

Definition and Identities of Regular Expressions, Construction of Regular Expression from the RE, Construction of FA from the given RE using direct and indirect methods, Arden's Theorem, Pumping Lemma for RL, Closure properties of RLs, Applications.

Given L, Construction of NFA for L, Conversion of FA to RE using NFA, Closure Properties of REs.

(Refer Chapter 3)

Unit III : Context Free Grammar and Languages

(08 Hours)

Introduction, Formal Definition of Grammar, Notations, Derivation Process, Derivation, derivation trees, Context Free Languages, Ambiguous CFG, Removal of Ambiguity, Normal Forms, Chomsky Hierarchy, Regular grammar, equivalence of RG(LRG) and FA.

Derivation, Rightmost Derivation, Uniqueness of derivation, Simplification of RLG and FA.

(Refer Chapters 4 and 5)

Unit IV : Pushdown Automata and Post Machines

(08 Hours)

Push Down Automata : Introduction and Definition of PDA, Construction (Pictorial/ Transition diagram) of PDA, Instantaneous Description and ACCEPTANCE of CFL by empty stack and final state, Deterministic PDA Vs Nondeterministic PDA, Closure properties of CFLs, pumping lemma for CFL.

Post Machine : Definition and construction.

(Refer Chapter 6)

Unit V : Turing Machines

(08 Hours)

Formal definition of a Turing machine, Recursive Languages and Recursively Enumerable Languages, Design of Turing machines, Variants of Turing Machines: Multi-tape Turing machines, Universal Turing Machine, Nondeterministic Turing machines. Comparisons of all automata.

(Refer Chapter 7)

Unit VI : Computational Complexity

(08 Hours)

Decidability : Decidable problems concerning regular languages, Decidable problems concerning context-free languages, Un-decidability, Halting Problem of TM, A Turing-unrecognizable language.

Reducibility : Un-decidable Problems from Language Theory, A Simple Un-decidable Problem PCP, Mapping Reducibility.

Time Complexity : Measuring Complexity, The Class P, Examples of problems in P, The Class NP, Examples of problems in NP, NP-completeness.

(Refer Chapter 8)



**UNIT - I**

Syllabus : Basic Concepts : Symbols, Strings, Language, Formal Language, Natural Language. Basic Machine and Finite State Machine.

FSM without output : Definition and Construction -

DFA, NFA, NFA with epsilon-Moves, Minimization Of FA, Equivalence of NFA and DFA, Conversion of NFA with epsilon moves to NFA, Conversion of NFA With epsilon moves to DFA.

FSM with output : Definition and Construction of Moore and Mealy Machines, Inter-conversion between Moore and Mealy Machines.

Chapter 1 : Basic Concepts 1-1 to 1-10

| | | |
|---------|---|-----|
| 1.1 | Basic Mathematical Objects | 1-1 |
| 1.2 | Sets..... | 1-1 |
| 1.2.1 | Operations on Sets..... | 1-1 |
| 1.2.2 | Important Postulates for Sets | 1-2 |
| 1.2.3 | Symmetric Difference | 1-2 |
| 1.2.4 | The Cartesian Product of Sets..... | 1-2 |
| 1.3 | Relations (SPPU - May 12). | 1-2 |
| 1.3.1 | Closure of Relations | 1-2 |
| 1.4 | Functions | 1-3 |
| 1.5 | Logic | 1-3 |
| ✓ | Syllabus Topic : Symbols, Strings, Language | 1-4 |
| 1.6 | Languages (SPPU - May 16)..... | 1-4 |
| 1.6.1 | Kleene Closure (SPPU - Dec. 13, May 16) | 1-5 |
| 1.6.2 | Recursive Definition of a Language..... | 1-5 |
| 1.7 | Proofs..... | 1-6 |
| 1.7.1 | Direct Proof..... | 1-6 |
| 1.7.2 | Proof by Contradiction | 1-6 |
| 1.7.3 | Mathematical Induction..... | 1-6 |
| ✓ | Syllabus Topic : Formal Language, Natural Language..... | 1-7 |
| 1.8 | Natural and Formal Language (SPPU - Dec. 12). | 1-7 |
| ✓ | Syllabus Topic : Basic Machine and Finite State Machine..... | 1-7 |
| 1.9 | Basic Machine | 1-7 |
| 1.9.1 | Introduction to Finite Automata..... | 1-7 |
| 1.9.1.1 | Working of a Finite Automata | 1-8 |
| 1.9.1.2 | Deterministic Finite Automata (DFA) | 1-9 |
| 1.9.1.3 | Definition of a DFA..... | 1-9 |
| 1.9.1.4 | Representation of a DFA..... | 1-9 |

| Chapter 2 : Finite Automata | | 2-1 to 2-111 |
|------------------------------------|---|---------------------|
| 2.1 | Introduction to Finite Automata (SPPU – May 12, Dec. 14, Dec. 16)..... | 2-1 |
| 2.1.1 | Working of a Finite Automata..... | 2-1 |
| 2.1.2 | Some Important Terms | 2-2 |
| 2.1.2.1 | Alphabet (SPPU - May 12)..... | 2-2 |
| 2.1.2.2 | Strings (words) (SPPU - Dec. 12)..... | 2-3 |
| 2.1.2.3 | Languages | 2-4 |
| 2.1.3 | Application of Finite Automata (SPPU - Dec. 16)..... | 2-4 |
| 2.1.4 | Limitations of Finite Automata (SPPU - Dec. 16)..... | 2-5 |
| 2.2 | Deterministic Finite Automata (DFA) | 2-5 |
| ✓ | Syllabus Topic : Definition of DFA | 2-5 |
| 2.2.1 | Definition of a DFA..... | 2-5 |
| 2.2.2 | Representation of a DFA..... | 2-5 |
| ✓ | Syllabus Topic : Construction of DFA | 2-7 |
| 2.2.3 | Designing a DFA..... | 2-7 |
| 2.2.4 | Solved Examples on DFA | 2-10 |
| 2.2.4.1 | Examples on Counting of Symbols | 2-10 |
| 2.2.4.2 | Examples on Substring | 2-16 |
| 2.2.4.3 | Examples of Divisibility | 2-26 |
| 2.2.5 | Language of DFA | 2-30 |
| ✓ | Syllabus Topic : Equivalence of DFA | 2-32 |
| 2.2.6 | Equivalence of DFAs (SPPU - May 15) | 2-32 |
| 2.2.7 | Closure Property of Language Accepted by a DFA | 2-35 |
| 2.2.7.1 | Union, Intersection, Difference | 2-35 |
| 2.2.7.2 | Complementation | 2-38 |
| ✓ | Syllabus Topic : Minimization of FA | 2-38 |
| 2.2.8 | Minimization of DFA | 2-38 |
| 2.2.8.1 | Algorithm for Minimization DFA's | 2-39 |
| ✓ | Syllabus Topic : NFA | 2-44 |
| 2.3 | Non-deterministic Finite Automata | 2-44 |
| 2.3.1 | Definition of NFA | 2-45 |
| 2.3.2 | Processing of a String by NFA | 2-45 |
| 2.3.3 | NFA to DFA Conversion | 2-51 |
| ✓ | Syllabus Topic : Equivalence of NFA and DFA | 2-54 |
| ✓ | Syllabus Topic : NFA with Epsilon Moves | 2-72 |
| 2.3.4 | NFA with e-Transitions | 2-72 |
| ✓ | Syllabus Topic : Conversion of NFA with Epsilon moves to NFA | 2-73 |
| 2.3.4.1 | Equivalence of e-NFA and NFA | 2-73 |
| 2.3.4.2 | The Formal Notation for an e-NFA (SPPU - Dec. 15) | 2-74 |



| | | |
|--|--|-------|
| 2.3.4.3 | ϵ -Closures (SPPU - Dec. 14, Dec. 15) | 2-74 |
| Syllabus Topic : Conversion of NFA With Epsilon | | |
| | Moves to DFA | 2-77 |
| 2.3.4.4 | ϵ -NFA to DFA..... | 2-77 |
| 2.3.5 | Difference between NFA and DFA (SPPU – Dec. 12)..... | 2-83 |
| 2.4 | Finite Automata as Output Devices | 2-84 |
| 2.4.1 | A Sample Mealy Machine | 2-84 |
| ✓ | Syllabus Topic : Definition and Construction of Mealy Machines..... | 2-85 |
| 2.4.2 | Formal Definition of a Mealy Machine | 2-85 |
| 2.4.3 | A Sample Moore Machine (SPPU – May 12)..... | 2-85 |
| ✓ | Syllabus Topic : Definition and Construction of Moore Machines | 2-86 |
| 2.4.4 | Formal Definition of a Moore Machine (SPPU – May 12)..... | 2-86 |
| ✓ | Syllabus Topic : Conversion of Mealy Machine into a Moore Machine | 2-96 |
| 2.4.5 | Conversion of a Mealy Machine into a Moore Machine | 2-96 |
| ✓ | Syllabus Topic : Conversion of Moore Machine into a Mealy Machine..... | 2-103 |
| 2.4.6 | Conversion of a Moore Machine into a Mealy Machine..... | 2-103 |
| 2.5 | Minimization of a Mealy Machine | 2-108 |

UNIT - II

Syllabus : Definition and Identities of Regular Expressions, Construction of Regular Expression of the given L, Construction of Language from the RE, Construction of FA from the given RE using direct method, Conversion of FA to RE using Arden's Theorem, Pumping Lemma for RL, Closure properties of RLs; Applications of Regular Expressions.

**Chapter 3 : Regular Expressions (RE) and Languages
3-1 to 3-38**

| | | |
|-------|---|------|
| ✓ | Syllabus Topic : Construction of Regular Expression of the given L | 3-1 |
| 3.1 | Introduction (SPPU - Dec. 13) | 3-1 |
| 3.2 | Finite Automata Representing a Regular Expression | 3-2 |
| 3.2.1 | Composite Finite State Automata..... | 3-3 |
| ✓ | Syllabus Topic : Definition and Identities of Regular Expressions..... | 3-5 |
| 3.3 | Determination of Regular Expression..... | 3-5 |
| ✓ | Syllabus Topic : Construction of Language from the RE..... | 3-5 |
| 3.3.1 | Language Generated by a Regular Expression..... | 3-5 |
| 3.3.2 | Basic Properties of Regular Expressions | 3-5 |
| 3.4 | DFA to Regular Expression | 3-14 |

| | | |
|---------|---|------|
| ✓ | Syllabus Topic : Construction of FA from the given RE using Direct Method..... | 3-14 |
| 3.4.1 | State/Loop Elimination Process | 3-14 |
| 3.4.1.1 | A Generic One State Machine | 3-15 |
| 3.4.1.2 | A Generic Two State Machine | 3-15 |
| ✓ | Syllabus Topic : Conversion of FA to RE using Arden's Theorem | 3-26 |
| 3.4.2 | Arden's Theorem..... | 3-26 |
| 3.4.2.1 | Application of Arden's Theorem | 3-26 |
| 3.5 | FA Limitations | 3-29 |
| ✓ | Syllabus Topic : Pumping Lemma for RL | 3-29 |
| 3.6 | Pumping Lemma for Regular Language | 3-29 |
| 3.6.1 | Definition of Pumping Lemma..... | 3-30 |
| 3.6.2 | Interpretation of Pumping Lemma..... | 3-30 |
| 3.6.3 | Proof of Pumping Lemma | 3-30 |
| 3.6.4 | Application of Pumping Lemma | 3-31 |
| ✓ | Syllabus Topic : Closure Properties of RLs | 3-34 |
| 3.7 | Closure Properties of Regular Language..... | 3-34 |
| 3.7.1 | Regular Language is Closed under Union | 3-35 |
| 3.7.2 | Regular Language is Closed under Concatenation | 3-35 |
| 3.7.3 | Regular Language is Closed under Kleene Star..... | 3-35 |
| 3.7.4 | Regular Language is Closed under Complementation..... | 3-36 |
| 3.7.5 | Regular Language is Closed under Intersection..... | 3-36 |
| 3.7.6 | Regular Languages are Closed under Difference..... | 3-36 |
| 3.7.7 | Regular Languages are Closed under Reversal | 3-36 |
| ✓ | Syllabus Topic : Applications of Regular Expressions..... | 3-38 |
| 3.8 | Application of RE..... | 3-38 |
| 3.8.1 | R.E. in Unix | 3-38 |
| 3.8.2 | Lexical Analysis..... | 3-38 |

UNIT - III

Syllabus : Introduction, Formal Definition of Grammar, Notations, Derivation Process: Leftmost Derivation, Rightmost Derivation, derivation trees, Context Free Languages, Ambiguous CFG, Removal of ambiguity, Simplification of CFG, Normal Forms, Chomsky Hierarchy.

Regular grammar, equivalence of RG(LRG and RLG) and FA.

**Chapter 4 : Context Free Grammar (CFG) and Languages
4-1 to 4-45**

| | | |
|-------|---|-----|
| ✓ | Syllabus Topic : Introduction..... | 4-1 |
| 4.1 | An Example to Explain Grammar (SPPU - May 12) | 4-1 |
| ✓ | Syllabus Topic : Formal Definition of Grammar | 4-3 |
| 4.2 | Context Free Grammar (SPPU - Dec. 14, Aug. 15, Dec. 15)..... | 4-3 |
| ✓ | Syllabus Topic : Notations | 4-3 |
| 4.2.1 | Notations | 4-3 |
| ✓ | Syllabus Topic : Context Free Languages | 4-3 |
| 4.2.2 | The Language of a Grammar..... | 4-3 |



| | | |
|---------|--|------|
| ✓ | Syllabus Topic : Derivation Process : Leftmost Derivation, Rightmost Derivation | 4-4 |
| 4.2.2.1 | Sentential Form | 4-4 |
| ✓ | Syllabus Topic : Derivation Trees | 4-4 |
| 4.2.2.2 | Parse Tree (SPPU - May 16)..... | 4-4 |
| 4.2.3 | Writing Grammar for a Language (SPPU - Dec. 13)..... | 4-7 |
| 4.2.3.1 | Union Rule for Grammar..... | 4-10 |
| 4.2.3.2 | Concatenation Rule for Grammar..... | 4-11 |
| ✓ | Syllabus Topic : Ambiguous CFG, Removal of Ambiguity | 4-19 |
| 4.3 | Ambiguous Grammar (SPPU - Dec. 14, Dec. 15)..... | 4-19 |
| ✓ | Syllabus Topic : Simplification of CFG | 4-26 |
| 4.4 | Simplification of CFG..... | 4-26 |
| 4.4.1 | Elimination of Useless Symbols (SPPU - May 12)..... | 4-26 |
| 4.4.1.1 | Non-generating Symbols | 4-26 |
| 4.4.1.2 | Non-reachable Symbols | 4-28 |
| 4.4.2 | Elimination of ϵ -productions (SPPU - May 12)..... | 4-29 |
| 4.4.3 | Elimination of Unit Productions (SPPU - May 12) | 4-31 |
| ✓ | Syllabus Topic : Normal Forms | 4-35 |
| 4.5 | Normal Forms for CFG | 4-35 |
| 4.5.1 | Chomsky Normal Form (CNF) (SPPU - Dec. 14)..... | 4-35 |
| 4.5.1.1 | Algorithm for CFG to CNF Conversion | 4-35 |
| 4.5.2 | Greibach Normal Form (GNF) (SPPU - Dec. 14)..... | 4-39 |
| 4.5.2.1 | Removing Left Recursion | 4-39 |
| 4.5.2.2 | Algorithm for Conversion from CFG to GNF..... | 4-40 |
| ✓ | Syllabus Topic : Chomsky Hierarchy..... | 4-45 |
| 4.6 | Chomsky Classification for Grammar (SPPU - Dec. 13, Aug. 15)..... | 4-45 |
| 4.6.1 | Type 3 or Regular Grammar..... | 4-45 |
| 4.6.2 | Type 2 or Context Free Grammar..... | 4-45 |
| 4.6.3 | Type 1 or Context Sensitive Grammar | 4-45 |
| 4.6.4 | Type 0 or Unrestricted Grammar | 4-45 |

Chapter 5 : Regular Grammar 5-1 to 5-12

| | | |
|-------|---|-----|
| ✓ | Syllabus Topic : Regular Grammar | 5-1 |
| 5.1 | Regular Grammar (SPPU - Dec. 15)..... | 5-1 |
| ✓ | Syllabus Topic : Equivalence of RLG and FA | 5-1 |
| 5.2 | DFA to Right Linear Regular Grammar | 5-1 |
| 5.2.1 | Right Linear Grammar to DFA | 5-3 |
| ✓ | Syllabus Topic : Equivalence of LRG and FA | 5-5 |
| 5.3 | DFA to Left-linear Grammar | 5-5 |
| 5.3.1 | Left Linear Grammar to DFA | 5-5 |
| 5.4 | Right Linear Grammar to Left Linear Grammar..... | 5-7 |
| 5.5 | Left Linear Grammar to Right Linear Grammar..... | 5-8 |

UNIT - IV

Syllabus : Push Down Automata : Introduction and Definition of PDA, Construction (Pictorial/ Transition diagram) of PDA, Instantaneous Description and ACCEPTANCE of CFL by empty stack and final state, Deterministic PDA Vs Nondeterministic PDA, Closure properties of CFLs, pumping lemma for CFL.

Post Machine- Definition and construction.

Chapter 6 : Pushdown Automata (PDA) 6-1 to 6-38

| | | |
|----------|---|------|
| ✓ | Syllabus Topic : Introduction to PDA, Construction (Pictorial/Transition Diagram) of PDA | 6-1 |
| 6.1 | Introduction to Pushdown Automata (PDA) (SPPU - Dec. 12) | 6-1 |
| ✓ | Syllabus Topic : Definition of PDA..... | 6-2 |
| 6.2 | The Formal Definition of PDA (SPPU - Dec. 12, Dec. 14, Dec. 15, May 16)..... | 6-2 |
| ✓ | Syllabus Topic : Instantaneous Description..... | 6-3 |
| 6.3 | Instantaneous Description of a PDA..... | 6-3 |
| 6.4 | The Language of a PDA..... | 6-4 |
| ✓ | Syllabus Topic : Acceptance of CFL by Final State | 6-5 |
| 6.4.1 | Acceptance by Final State (SPPU - Dec. 16) | 6-5 |
| ✓ | Syllabus Topic : Acceptance of CFL by Empty Stack | 6-5 |
| 6.4.2 | Acceptance by Empty Stack (SPPU - Dec. 16) | 6-5 |
| 6.5 | Non-deterministic PDA (NPDA) (SPPU - May 12)..... | 6-12 |
| 6.6 | Push Down Automata and Context Free Language | 8-15 |
| 6.6.1 | Construction of PDA from CFG..... | 6-15 |
| 6.6.2 | Construction of CFG from PDA..... | 6-18 |
| ✓ | Syllabus Topic : Deterministic PDA Vs Nondeterministic PDA | 6-27 |
| 6.7 | Deterministic Push Down Automata (DPDA) (SPPU - May 12) | 6-27 |
| 6.7.1 | Regular Language and DPDA | 6-27 |
| 6.8 | Application of PDA (SPPU - Dec. 12, Dec. 14, May 16) | 6-28 |
| ✓ | Syllabus Topic : Pumping Lemma for CFL | 6-28 |
| 6.9 | Pumping Lemma for CFG (SPPU - May 12, Dec. 15) | 6-28 |
| ✓ | Syllabus Topic : Closure Properties of CFLs | 6-30 |
| 6.10 | Properties of Context-free Languages | 6-30 |
| 6.10.1 | Closure Properties (SPPU - Dec. 13) | 6-30 |
| 6.10.1.1 | CFL is Closed under Union | 6-30 |
| 6.10.1.2 | CFL is Closed under Concatenation | 6-30 |
| 6.10.1.3 | CFL is Closed under Kleene Star | 6-31 |
| 6.10.1.4 | CFL is not Closed under Intersection | 6-31 |
| 6.10.1.5 | CFL is not Closed under Complementation | 6-31 |
| 6.10.1.6 | Intersection of CFL and RL | 6-31 |
| 6.10.1.7 | CFL is Closed under Reversal | 6-32 |



| | | |
|--------|--|------|
| 6.10.2 | Algorithmic Properties..... | 6-32 |
| ✓ | Syllabus Topic : Post Machine - Definition and Construction..... | 6-32 |
| 6.11 | Post Machine (SPPU - Dec. 12, May 16)..... | 6-32 |
| 6.11.1 | Definition of Post Machine (SPPU - Dec. 13, Dec. 14, May 15, Dec. 15). | 32 |
| 6.12 | Power of Various Machines (SPPU - Dec. 14)..... | 38 |

UNIT - V

Syllabus : Formal definition of a Turing machine, Recursive Languages and Recursively Enumerable Languages, Design of Turing machines, Variants of Turing Machines: Multi-tape Turing machines, Universal Turing Machine, Nondeterministic Turing machines. Comparisons of all automata.

Chapter 7 : Turing Machines 7-1 to 7-42

| | | |
|-------|---|------|
| 7.1 | Introduction to Turing Machine (SPPU - Dec. 13)..... | 7-1 |
| 7.1.1 | Limitation of Turing Machine..... | 7-3 |
| ✓ | Syllabus Topic : Formal Definition of a Turing Machine, Design of Turing Machines..... | 7-3 |
| 7.2 | The Formal Definition of Turing Machine | 7-3 |
| 7.2.1 | A String Accepted by TM | 7-4 |
| 7.2.2 | Instantaneous Descriptions for Turing Machines | 7-5 |
| 7.3 | Turing Machines as Computer of Functions..... | 7-17 |
| 7.4 | Languages of TM..... | 7-31 |
| ✓ | Syllabus Topic : Variants of Turing Machine | 7-31 |
| 7.5 | Extension of Turing Machine (SPPU - Dec. 13, May 15)..... | 7-31 |
| 7.5.1 | Two-way Infinite Turing Machine..... | 7-31 |
| 7.5.2 | A Turing Machine with Multiple Heads | 7-31 |
| ✓ | Syllabus Topic : Multi-Tape Turing Machine | 7-31 |
| 7.5.3 | Multi-Tape Turing Machine (SPPU - Dec. 14, Dec. 15, May 16, Dec. 16). | 7-31 |
| ✓ | Syllabus Topic : Non-Deterministic Turing Machine | 7-33 |
| 7.5.4 | Non-Deterministic Turing Machine (SPPU - May 12, Dec. 14)..... | 7-33 |
| ✓ | Syllabus Topic : Universal Turing Machine | 7-35 |
| 7.6 | Universal Turing Machine (SPPU - Dec. 13, Dec. 14, Dec. 15, May 16, Dec. 16). | 7-35 |
| 7.6.1 | Church-Turing Hypothesis (SPPU - Dec.12)..... | 7-36 |
| ✓ | Syllabus Topic : Comparisons of All Automata | 7-36 |
| 7.6.2 | Power of Various Machines | 7-36 |
| ✓ | Syllabus Topic : Recursive Language and Recursively Enumerable Languages..... | 7-36 |
| 7.7 | Recursively Enumerable and Recursive Language (SPPU - Dec. 14, May 15, Dec. 15, May 16, Dec. 16) | 7-36 |
| 7.7.1 | Turing Acceptable Language (SPPU - Dec. 15)..... | 7-37 |

| | | |
|-------|--|------|
| 7.8 | Enumerating a Language | |
| | (SPPU - Dec. 15, May 16, Dec. 16)..... | 7-39 |
| 7.8.1 | Finite and Infinite Sets..... | 7-40 |
| 7.9 | Compare Type 0, Type 1, Type 2 and Type 3 Grammars | 7-42 |

UNIT - VI

Syllabus : Decidability: Decidable problems concerning regular languages, Decidable problems concerning context-free languages, Un-decidability, Halting Problem of TM, A Turing-unrecognizable language.

Reducibility : Un-decidable Problems from Language Theory, A Simple Un-decidable Problem PCP, Mapping Reducibility

Time Complexity : Measuring Complexity, The Class P, Examples of problems in P, The Class NP, Examples of problems in NP, NP-completeness.

Chapter 8 : Computational Complexity 8-1 to 8-16

| | | |
|-------|---|-----|
| ✓ | Syllabus Topic : Un-decidability, Un-decidable Problems from Language Theory | 8-1 |
| 8.1 | Un-decidability (SPPU - May 12, May 13, May 14)..... | 8-1 |
| ✓ | Syllabus Topic : Halting Problem of TM..... | 8-1 |
| 8.1.1 | Halting Problem of a Turing Machine (SPPU - May 12, Dec. 12, May 13, Dec.13, Dec. 16)... | 8-1 |
| ✓ | Syllabus Topic : Decidable Problems Concerning Regular Languages, Decidable Problems Concerning Context-free Languages, A Simple Un-decidable Problem PCP | 8-2 |
| 8.1.2 | Un-decidability of Post Correspondence Problem (SPPU - May 12, Dec. 12, May 13, Dec. 13, May 14, Dec. 15, Dec. 16)..... | 8-2 |
| 8.1.3 | Modified PCP Problem..... | 8-5 |
| ✓ | Syllabus Topic : A Turing Unrecognizable Language | 8-5 |
| 8.1.4 | A Turing Unrecognizable Language | 8-5 |
| ✓ | Syllabus Topic : Measuring Complexity | 8-5 |
| 8.2 | Computational Complexity (SPPU - Dec. 15, Dec. 16)..... | 8-5 |
| 8.2.1 | P and NP-class Problem..... | 8-6 |
| 8.2.2 | Intractable Problems | 8-7 |
| ✓ | Syllabus Topic : The Class P, Examples of Problems in P, The Class NP, Examples of Problems in NP | 8-7 |
| 8.3 | The Classes P and NP (SPPU - Dec. 14, Dec. 15, May 16)..... | 8-7 |
| 8.3.1 | Problem Solvable in Polynomial Time | 8-7 |
| 8.3.2 | An Example : Kruskal's Algorithm..... | 8-7 |
| 8.3.3 | Minimal Spanning Tree | 8-7 |
| 8.3.4 | Kruskal's Algorithm (SPPU - May 16)..... | 8-8 |
| 8.3.5 | Kruskal's Algorithm using a Turing Machine (SPPU - May 16)..... | 8-8 |



| | | | | | |
|-------|---|------|-------|--|------|
| 8.3.6 | Polynomial-Time Reduction (SPPU - May 15, Dec. 15, May 16) | 8-9 | 8.6.1 | Normal Forms for Boolean Expressions | 8-11 |
| ✓ | Syllabus Topic : NP-completeness..... | 8-9 | 8.6.2 | Converting Expressions to CNF..... | 8-11 |
| 8.3.7 | NP-Complete Problems..... | 8-9 | 8.6.3 | Clique Problem is NP-Complete (SPPU - May 16) | 8-12 |
| 8.4 | An NP-Complete Problem | 8-9 | 8.6.4 | The Problem of Independent Sets | 8-13 |
| 8.4.1 | The Satisfiability Problem (SAT)..... | 8-9 | 8.6.5 | The Node-Cover Problem (SPPU - May 15) | 8-13 |
| 8.4.2 | Traveling Salesman Problem (TSP)..... | 8-9 | 8.6.6 | The Directed Hamilton-Circuit Problem | 8-14 |
| 8.4.3 | Tractable and Intractable (SPPU - May 15) | 8-9 | 8.6.7 | Undirected Hamiltonian Circuit | 8-14 |
| 8.5 | Representing SAT Instances..... | 8-10 | 8.7 | Partial Recursive Function (SPPU - Dec. 15) | 8-15 |
| 8.5.1 | NP-Completeness of the SAT Problem (SPPU - Dec. 14, May 15) | 8-10 | ✓ | Syllabus Topic : Mapping Reducibility | 8-16 |
| 8.5.2 | 3-SAT problem (SPPU - Dec. 15) | 8-11 | 8.8 | Turing Reducibility (SPPU - May 15, May 16, Dec. 16) | 8-16 |
| 8.6 | A Restricted Satisfiability Problem..... | 8-11 | | | |



CHAPTER

1

Basic Concepts**Syllabus**

Basic Concepts : Symbols, Strings, Language, Formal Language, Natural Language, Basic Machine and Finite State Machine.

1.1 Basic Mathematical Objects

Some of the fundamental mathematical ideas are :

1. Sets 2. Functions
3. Relations 4. Logic

A language is basically a set whose elements are strings.

1.2 Sets

Set is a collection of things called elements or members. A set can be described by listing its elements within braces.

- {5}, a set containing only one element 5.
- $N = \{1, 2, 3, \dots\}$ is a set of natural numbers.
- $Z = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ is the set of integers.

A set can also be described using the notation given below :

$$A = \{x \mid x \text{ is an odd integer, } x > 1\}$$

It should be read as "A is a set of all x such that x is an odd integer and x is greater than 1".

The set A can also be written as :

$$A = \{2k - 1 \mid k \in N\}$$

To indicate that x is an element of the set B, we write :

$$x \in B$$

(i) Equality of sets

Two sets A and B are equal, if and only if A and B contain the same elements or both are empty. If the two sets A and B are equal then we write

$$A = B$$

(ii) The empty set

A set which contains no elements is called an empty set.

(iii) Subsets

A set A is a subset of a set B, and we write $A \subseteq B$, if and only if every element of A is an element of B.

If $A \subseteq B$ and $A \neq B$, then A is called a proper subset of B and it is written as $A \subset B$.

(iv) The power set

The power set of a set A, denoted as $P(A)$, is the set of all subsets of A.

$$\text{If } A = \{a, b, c\} \text{ then}$$

$$P(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{ab\}, \{ac\}, \{bc\}, \{abc\}\}$$

1.2.1 Operations on Sets**(i) Complement of a set**

Complement of a set A is represented as A' . A' is a set of every element that is not in A.

$$A' = \{x \in U \mid x \notin A\},$$

where U is some universal set.

(ii) Union of two sets

Union of two sets A and B is represented as $A \cup B$. Union of two sets A and B is the set of elements in A or in B.

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



(iii) Intersection of two sets

Intersection of two sets A and B is represented as $A \cap B$. Intersection of two sets A and B is the set of elements which belong to both A and B.

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

(iv) Set difference

The set difference of two sets A and B is represented as $A - B$. It is the set of those elements of A which are not in B.

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

1.2.2 Important Postulates for Sets

1. $A \cup B = B \cup A$ [Commutative laws]
 $A \cap B = B \cap A$
2. $A \cup (B \cup C) = (A \cup B) \cup C$ [Associative laws]
 $A \cap (B \cap C) = (A \cap B) \cap C$
3. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ [Distributive laws]
 $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
4. $A \cup A = A$ [Idempotent laws]
 $A \cap A = A$
5. $A \cup (A \cap B) = A$ [Absorptive laws]
 $A \cap (A \cup B) = A$
6. $(A \cup B)' = A' \cap B'$ [De Morgan's laws]
 $(A \cap B)' = A' \cup B'$

Example 1.2.1

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

Solution : It can be proved through contradiction.

Let us assume that $A \cap C \neq \emptyset$ and hence we can assume that for some x, $x \in A \cap C$

$$x \in A \cap C \leftrightarrow x \in A \text{ and } x \in C \quad \dots(1)$$

$$\text{Since, } C \subseteq B \leftrightarrow x \in C \text{ then } x \in B \quad \dots(2)$$

From (1) and (2), we can say that

$$x \in A \text{ and } x \in C \text{ and } x \in B \quad \dots(3)$$

From (3) we can conclude that

$$x \in A \cap B$$

But $A \cap B = \emptyset$, Hence there is a contradiction.

1.2.3 Symmetric Difference

The symmetric difference of two sets A and B is written as $A \oplus B$. It consists of those elements which are either in A or in B but not in both.

$$A \oplus B = \{x \mid (x \in A \text{ and } x \notin B) \text{ or } (x \in B \text{ and } x \notin A)\}$$

The symmetric difference of two sets can be written as :

$$A \oplus B = (A \cup B) - (A \cap B)$$

1.2.4 The Cartesian Product of Sets

The Cartesian product of two sets A and B is the set :

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

Example : Let $A = \{a,b,c\}$ and

$$B = \{x,y\}$$

$$\text{then } A \times B = \{(a,x), (a,y), (b,x), (b,y), (c,x), (c,y)\}$$

1.3 Relations

SPPU - May 12

University Question

Q. Define and explain : Relation and its properties.

(May 2012, 2 Marks)

A binary relation from A to B is a subset of $A \times B$. A binary relation on A (on the same set) is a subset of $A \times A$.

Properties of relations

- (i) A relation R in set A is **reflexive** if xRx for every x in A.
- (ii) A relation R in set A is **symmetric** if $a,b \in A$ and aR_b then bR_a .
- (iii) A binary relation R on set A is **transitive** if and only if $a,b,c \in A$ and both aR_b and bR_c , then aR_c .

Equivalence relation

A relation R in set A is called an equivalence relation if it is reflexive, symmetric and transitive.

1.3.1 Closure of Relations

If a relation R in set A is not reflexive then it can be made reflexive by adding ordered pairs (x,y) to R which are not already there in R.

Example : Consider a relation $R = \{(1,2), (3,2), (1,1)\}$ R is not reflexive. But by adding $(2,2)$ and $(3,3)$ to R, it can be made reflexive.

$$\therefore \text{Reflexive closure of } R = \{(1,1), (1,2), (2,1), (2,2), (3,1), (3,2), (3,3)\}$$

If a relation R in set A is not symmetric then it can be made symmetric by adding ordered pairs (y,x) to R if $(x,y) \in R$ and $(y,x) \notin R$.

Example : Consider a relation $R = \{(1,2), (3,2), (1,1)\}$.

R is not symmetric. But by adding $(2,1)$ and $(2,3)$, it can be made symmetric.



- \therefore Symmetric closure of $R = \{(1,2),(3,2),(1,1),(2,1),(2,3)\}$
- If a relation R in set A is not transitive then it can be made transitive by adding ordered pairs (x,z) to R if $(x,y) \in R$ and $(y,z) \in R$ but $(x,z) \notin R$.

Example : Consider a relation $R = \{(1,2),(2,3),(1,1)\}$
 R is not transitive as $_1R_2$ and $_2R_3$ but 1 is not related to 3 . The relation R can be made transitive by adding the ordered pair $(1,3)$.

$$\therefore \text{Transitive closure of } R = \{(1,2),(2,3),(1,1),(1,3)\}$$

Example 1.3.1

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

Solution :

(i) Transitive closure

The relation can be made transitive by adding ordered pairs $(1,3)$ and $(2,4)$. $_1R_2$ and $_2R_3$, $_2R_3$ and $_3R_4$.

$$\therefore \text{Transitive closure of the relation} = \{(1,2)(1,3)(2,3)(2,4)(3,4)(5,4)\}$$

(ii) Symmetric closure

The relation can be made symmetric by adding ordered pairs $(2,1),(3,2),(4,3)$ and $(4,5)$

\therefore Symmetric closure of the relation

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

Example 1.3.2

Consider a relation $R = \{(1,1),(2,2),(3,4),(4,3)\}$, defined on set $A = \{1,2,3,4\}$.

Obtain R^* (Reflexive and transitive closure of R)

Solution :

Step1 : Ordered pairs $(3,3)$ and $(4,4)$ are added to make the relation reflexive.

$$\therefore \text{New relation } R = \{(1,1),(2,2),(3,3),(3,4),(4,3),(4,4)\}$$

Relation is already transitive.

$$\therefore R^* = \{(1,1),(2,2),(3,3),(3,4),(4,3),(4,4)\}$$

1.4 Functions

A function from a set A to a set B is a binary relation f from A to B such that for every $a \in A$, there is exactly one $b \in B$. The function is denoted by

$$f : A \rightarrow B$$

Example : A function $f : \{1,2,3\} \rightarrow \{a, b\}$ can be defined by $f(1) = b$, $f(2) = a$, $f(3) = a$.

- Let $f : A \rightarrow B$ be a function from A to B
 1. The domain of f is the set A
 2. $f(x)$ is called the image of x under f .
- $f : A \rightarrow B$ is said to be one-to-one (injective) if different elements in A have different images.

- $f : A \rightarrow B$ is onto (surjective) if every element in B is image of some element in A .
- $f : A \rightarrow B$ is said to be bijective if it is both one-to-one and onto.

1.5 Logic

(i) Proposition

It is a declarative statement having a truth value (true or false).

(ii) Implication

Implication is a statement of the form “ p implies q ”, where p and q are statements. It is written as

p is called $p \rightarrow q$ hypothesis and q is called conclusion.

(iii) Connectives

Logical connectives are used for compound statements. The most common logical connectives are :

| | |
|------------------------------|---------------------------|
| \wedge – And (conjunction) | \vee – Or (disjunction) |
| \neg – Not (negation) | |

(iv) Logical equivalences

$A \Leftrightarrow B$ denotes that A and B are logically equivalent. Some basic equivalences are :

1. Indempotence 2. Commutativity

| | |
|--------------------------------|---|
| $p \vee p \Leftrightarrow p$ | $p \vee q \Leftrightarrow q \vee p$ |
| $p \wedge p \Leftrightarrow p$ | $p \wedge q \Leftrightarrow q \wedge p$ |

3. Associativity

| | |
|---|--|
| $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ | $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \wedge r)$ |
| $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$ | $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$ |

5. Double negation

| | |
|----------------------------------|---|
| $\neg(\neg p) \Leftrightarrow p$ | $\neg(p \vee q) \Leftrightarrow (\neg p) \wedge (\neg q)$ |
| | $\neg(p \wedge q) \Leftrightarrow (\neg p) \vee (\neg q)$ |

6. De Morgan's laws

7. Implication

$$p \rightarrow q = \neg p \vee q$$

(v) Tautology

A compound statement that is always true, irrespective of the truth values assigned to its variables, is a tautology.

Example 1.5.1

Show that $(\neg p) \rightarrow (p \rightarrow q)$ is a tautology.

Solution :

$$\begin{aligned} [(\neg p) \rightarrow (p \rightarrow q)] &\Leftrightarrow [(\neg p) \rightarrow (\neg p \vee q)] \Leftrightarrow [\neg(\neg p) \vee (\neg p \vee q)] \\ &\Leftrightarrow [p \vee \neg p \vee q] \Leftrightarrow [1 \vee q] \Leftrightarrow 1 \end{aligned}$$



Syllabus Topic : Symbols, Strings, Language

1.6 Languages

SPPU - May 16

University Question

Q. Define the following terms :

- (i) Strings
- (ii) Language
- (iii) Alphabets

(May 2016, 4 Marks)

A subset of strings over an alphabet is a language. An **alphabet** is a finite, non empty set of symbols. Conventionally, the symbol Σ is used for an alphabet.

Common alphabet include :

1. $\Sigma\{0,1\}$, the binary alphabet.

2. $\Sigma\{A,B,...Z\}$, the set of uppercase letters.

3. $\Sigma\{0,1,2,...9\}$, the decimal alphabet.

- A **string** is a finite sequence of symbols from the alphabet. For example :

1. 10110 is a string from binary alphabet.

2. 5920 is a string from decimal alphabet.

3. 'STAMP' is a string from roman alphabet.

A string having no symbol is known as an empty string. An empty string is denoted by ϵ .

- The **exponential notation** is used to express the set of all strings of a certain length.

Σ^k = set of strings of length k,
each of whose symbol is in Σ .

For example,

if $\Sigma = \{0,1\}$ then $\Sigma^1 = \{0,1\}$

$\Sigma^2 = \{00,01,10,11\}$

And $\Sigma^0 = \{\epsilon\}$

- The reversal of string ω is denoted by ω^R .

For example,

If $\omega = xyz$

Then $\omega^R = zyx$.

- A string v is a **substring** of a string ω if and only if there are strings x and y such that,

$\omega = x v y$, where both x and y could be null

Every string is a substring of itself.

For example,

If $\omega = 'abcdef'$ is a string.

then 'cde' is a substring of ω .

- For each string ω , the string ω^i is defined as,

$\omega^0 = \epsilon$, the empty string

$\omega^{i+1} = \omega^i \cdot \omega$ for each $i \geq 0$ where, i is a natural number.

For example,

If $\omega = abb$

Then $\omega^0 = \epsilon$

$\omega^1 = abb$

$\omega^2 = \omega \cdot \omega = abbabb$

$\omega^3 = \omega^2 \cdot \omega = abbabbabb$

- The set of all strings over an alphabet Σ is denoted by Σ^* .

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots$$

/ | \

ϵ String of length 1 String of length 2

For example,

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

then $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

- A subset of strings over an alphabet Σ is a **language**. In other words, any subset of Σ^* is a language.

- If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ .

For example,

$L_1 = \{\omega \in \{0,1\}^* \mid \omega \text{ has an equal number of } 0's \text{ and } 1's\}$

i.e., L_1 is a language over alphabets {0, 1}, having equal number of 0's and 1's.

$\therefore L_1 = \{\epsilon, 01, 10, 0011, 0101, 1100, 1010, \dots\}$

Thus, a language can be specified by :

$L = \{\omega \in \Sigma^* \mid \omega \text{ has the given property}\}$

Some examples of language are :

1. A language of all strings, where the string represents a binary number.

$\{0, 1, 00, 01, 10, 11, \dots\}$

2. The set of strings of 0's and 1's with an equal number of each.

$\{\epsilon, 01, 10, 0011, 0101, 1010, 1100, \dots\}$

3. The set of strings of 0's and 1's, ending in 11.

$\{11, 011, 111, 0011, 0111, 1011, \dots\}$

4. Σ^* is a language for any alphabet Σ .

5. ϕ , the empty language, is a language over an alphabet.

6. $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.



1.6.1 Kleene Closure SPPU Dec 13, May 16

University Question

- Q. Define Kleene Closure with example.
(Dec. 2013, May 2016, 4 Marks)

Given an alphabet Σ , the Kleene closure of Σ is a language given by

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots$$

| | |
 ε Strings of Strings of
 length 1 length 2

For example :

1. If $\Sigma = \{x\}$, then

$$\Sigma^* = \{\epsilon, x, xx, \dots\}$$

2. If $\Sigma = \{0, 1\}$, then

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$$

If L is a language, then L^* is the set of all finite strings formed by concatenating words from L . Any word can be used any number of times including zero time. ϵ is a member of L^* .

Example : If $L = \{aa, b\}$ then

$$L^* = \{\epsilon, b, aa, bb, bbb, baa, aab, \dots\}$$

It may be noted that a always appears in pair.

- If $\Sigma = \emptyset$ (an empty language)

$$\text{Then, } \Sigma^* = \{\epsilon\}$$

- L^+ is known as the positive closure of L . L^+ is the set of all finite strings formed by concatenating words from L . Any word can be used one or more times.

Example : If $L = \{aa, b\}$ then

$$L^+ = \{b, aa, bb, bbb, baa, aab, \dots\}$$

Example 1.6.1

Let L be a language. It is clear from the definition that $L^+ \subseteq L^*$. Under what circumstances are they equal?

Solution :

L^* always contains an epsilon.

L^+ may not contain an epsilon.

Thus, the only difference between L^+ and L^* is that L^+ may or may not contain an epsilon but L^* will always contain an epsilon.

If L itself contains an epsilon then there will be no difference between L^+ and L^* .

1.6.2 Recursive Definition of a Language

A language over an alphabet Σ can be described recursively. A recursive definition has three steps :

1. Specify some basic objects in the set.
2. Specify the rules for constructing more objects from the objects already known.
3. Declaration that no objects except those constructed as given above are allowed in the set.

Example : Let us try to define a language of positive integers

$$L = \{1, 2, 3, \dots\}$$

The object 2 can be derived from 1 by adding 1 to it.

The object 3 can be derived from object 2 by adding 1 to it.

Thus, if 1 is taken as the base object then we can derive any positive integer from it.

Rules are given below :

Rule 1 : 1 is a positive integer

Rule 2 : If x is a positive integer, then so is $x + 1$.

Example 1.6.2

Define a language of polynomials recursively. Give derivation for $5x^2 + 7x$.

Solution : A polynomial is a finite sum of terms, where each term is of the form $c \cdot x^n$, c and n are integers.

We should be able to derive a term.

A polynomial can be derived by adding such terms.

Rule 1 : Any number is a term

Rule 2 : The variable x is a term

Rule 3 : If p and q are terms then pq is also a term

Rule 4 : A term is polynomial

Rule 5 : If the terms p and q are in polynomial, then so are $p + q$ and $p - q$.

Deriving $5x^2 + 7x$

| | |
|-----------------------------|---------------|
| 5 is a term | — (by Rule 1) |
| x is a term | — (by Rule 2) |
| $5x$ is a term | — (by Rule 3) |
| $(5x) \cdot x$ is a term | — (by Rule 3) |
| $5x^2$ is a polynomial | — (by Rule 4) |
| 7 is a term | — (by Rule 1) |
| $7x$ is a term | — (by Rule 3) |
| $5x^2 + 7x$ is a polynomial | — (by Rule 5) |



1.7 Proofs

A proof involves a statement of the form $p \rightarrow q$. There are several methods for establishing a proof of statements. Some of them are :

1. Direct proof
2. By contradiction
3. By mathematical induction.

1.7.1 Direct Proof

If we have to prove that $p \rightarrow q$, then a direct proof assumes p is true and uses this to show that q is true.

Example 1.7.1

Prove for any integer a and b if a and b are odd than ab is odd.

Solution :

Any odd integer x , can be written as $2y + 1$, where y is an integer.

\therefore There exists an integer x such that

$$a = 2x + 1 \quad \dots(1)$$

Similarly, there exists an integer y such that

$$b = 2y + 1$$

$$\begin{aligned} ab &= (2x+1)(2y+1) = 4xy + 2x + 2y + 1 \\ &= 2(2xy + x + y) + 1. \end{aligned}$$

The term $2(2xy + x + y)$ represents an even integer. When 1 is added to an even integer it becomes odd. This completes the proof.

1.7.2 Proof by Contradiction

If we have to prove that $p \rightarrow q$, then the method of contradiction assumes that p does not imply q and then try to derive some contradiction.

Example 1.7.2

Prove for any integer a and b if a and b are odd than ab is odd.

Solution : Contradiction step : ab is even. We can express a , b , and ab as given below :

$$a = 2x + 1 \quad [a \text{ is odd}]$$

$$b = 2y + 1 \quad [b \text{ is odd}]$$

$$ab = 2z \quad [ab \text{ is even}]$$

$$\therefore (2x+1)(2y+1) = 2z$$

$$4xy + 2x + 2y + 1 = 2z$$

$$2(2xy + x + y) + 1 = 2z$$

$$\therefore z = 2xy + x + y + \frac{1}{2}$$

z is not an integer and hence a contradiction.

1.7.3 Mathematical Induction

Proof by mathematical induction consists of three basic steps. If the statement p is to be proved then :

1. Show that p is true for some particular integer n_0 — this is called the **basis**.
2. Assume p is true for some particular integer $k \geq n_0$ — this is called **induction hypothesis**.
3. Prove is true for $n = k + 1$ — this is called **induction step**.

Example 1.7.3

Show for any $n \geq 0$

$$1 + 2 + 3 + \dots + n = (n^2 + n)/2$$

Solution : Basis step : for $n = 0$, L.H.S. = 0 and

R.H.S. = $(0^2 + 0)/2 = 0$. Hence the result is true for $n = 0$.

Induction hypothesis : we assume that we given statement is true for $n = k$.

$$\text{i.e. } 1 + 2 + 3 + \dots + k = (k^2 + k)/2$$

Induction step : we have to prove that the statement is true for $n = k + 1$

$$\begin{aligned} 1 + 2 + 3 + \dots + k + (k + 1) &= (k^2 + k)/2 + k + 1 \\ &= \frac{k^2 + k + 2k + 2}{2} = \frac{(k + 1)^2 + (k + 1)}{2} \end{aligned}$$

Hence, the statement is true for $n = k + 1$.

Therefore it is always true.

Example 1.7.4

Show by the principle of mathematical induction that $n^4 - 4n^2$ is divisible by 3 for all $n \geq 0$.

Solution :

Basis step : for $n = 0$, $n^4 - 4n^2 = 0 - 0 = 0$

0 is divisible by 3. Hence the statement is true for $n = 0$.

Induction hypothesis : $k^4 - 4k^2$ is divisible by 3 for $n = k$.

Induction step : we have to prove that $n^4 - 4n^2$ is divisible by 3 for $n = k + 1$

$$n^4 - 4n^2 \text{ for } n = k + 1 \text{ is } (k + 1)^4 - 4(k + 1)^2$$

Induction hypothesis can be subtracted from $(k + 1)^4 - 4(k + 1)^2$

If $(k + 1)^4 - 4(k + 1)^2$ is divisible by 3 then $4(k + 1) - 4(k + 1)^2 - (k^4 - 4k^2)$ is divisible by 3.



$$\begin{aligned}
 (k+1)^4 - 4(k+1)^2 - k^4 + 4k^2 &= k^4 + 4k^3 + 6k^2 + 4k + 1 \\
 &\quad - 4k^2 - 8k - 4 - k^4 + 4k^2 \\
 &= k^4 + 4k^3 + 2k^2 - 4k - 3 - k^4 + 4k^2 \\
 &= 4k^3 + 6k^2 - 4k - 3 \\
 &= 4k(k^2 - 1) + 6k^2 - 3 \\
 &= \underbrace{4k(k+1)(k-1)}_{\text{divisible by 3 as product of three consecutive numbers.}} + \underbrace{6k^2}_{\text{divisible by 3}} - 3
 \end{aligned}$$

Thus, $(k+1)^4 - 4(k+1)^2$ is divisible by 3. Hence proved.

Syllabus Topic : Formal Language, Natural Language

1.8 Natural and Formal Language

SPPU - Dec. 12

University Question

Q. Define formal language with example.
(Dec. 2012, 2 Marks)

In a natural language like English, it is very hard to state all the rules for the language. Understanding a natural language requires intelligence and ability. A natural language may have slang, idioms, dialect, metaphor and unintentional grammatical errors in the sentences. A spoken sentence in English may carry multiple meanings and the meaning may depend on the context in which it is spoken.

- A formal language is based on rules which are explicitly stated.
- 1. List of symbols is explicitly stated.
- 2. Rules for forming a string (word) are explicitly stated.
- 3. Rules for forming a sentence are explicitly stated.

The basic building block of a formal language is **alphabet**. Rules for forming a string from alphabet are defined without any ambiguity. The list of legal strings is called a formal language.

Syllabus Topic : Basic Machine and Finite State Machine

1.9 Basic Machine

A machine is defined as a system where energy, materials and information are transformed without direct intervention of human being. In computer science, a machine is used for transformation of information. Basic characteristics a machine are :

1. Input
2. Output
3. States
4. State relation
5. Output relation

1. **Input** : This is a sequence of symbols from the set of alphabet that is applied to the machine.
2. **Output** : It is a sequence of output symbols from the finite set of output symbols. This sequence is the output of the machine in response to any input.
3. **States** : A machine moves through various states like $q_0, q_1, q_2, \dots, q_n$.
4. **State relation** : The next state of a machine depends on the present state and the next input.
5. **Output relation** : Output is related to either state only or to both input and the current state.

1.9.1 Introduction to Finite Automata

Finite automaton or finite state machine can be thought of as a severely restricted model of a computer.

- Every computer has central processing unit, it executes a program stored in a memory. This program normally accepts some input and delivers processed result.
- The word Automata is for automation. A system where energy and information are transformed and used for performing some functions without direct involvement of men is called automaton.
- A finite automata is also called a finite state machine.
- A finite state machine is a mathematical model for actual physical process. By considering the possible inputs on which these machines can work, we can analyse their strengths and weaknesses.

- Finite automata are used for solving several common types of computer algorithms. Some of them are :
 1. Design of digital circuits.
 2. String matching.
 3. Communication protocols for information exchange.
 4. Lexical analyser of a typical compiler.

1.9.1.1 Working of a Finite Automata

Let us consider a T-flip flop.

A T-flip-flop has :

1. One input denoted by T.
2. Two outputs denoted by Q and \bar{Q} .
3. It has two distinct states, defined by the logical values of Q. The flip-flop is in q_0 state when $Q = 0$ and it is in q_1 state when $Q = 1$.

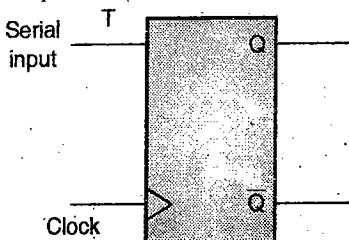


Fig. 1.9.1 : A T-flip-flop as a finite automata

4. A value of 1 applied to its input changes its state from q_0 to q_1 or from q_1 to q_0 .

A T-flip-flop can be used to check whether an input binary number contains an even number of 1's. Let us assume that a binary number 101110011 is applied to input T of the flip-flop. Initial state of the flip-flop is q_0 (i.e. $Q = 0$).

| | | | | | | | | | |
|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Input (T) | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| State (Q) | q_1 | q_1 | q_0 | q_1 | q_0 | q_0 | q_0 | q_1 | q_0 |

Fig. 1.9.2. : State transition of T-flip-flop under the input 101110011

From the Fig. 1.9.2, it should be clear that every even number of 1's in input will make the T-flip-flop toggle twice and thus it will be back to q_0 . Thus we can conclude the following :

1. If the initial state of flip flop is q_0 , it will come back to q_0 if the input number contains even number of 1's.

2. A single T-flip flop can be used as a machine for checking whether a binary number contains an even number of 1's.

The machine starts with initial state q_0 and after feeding the binary number, if the machine is found to be in q_0 , the input binary number contains an even number of 1's.

Working of finite automata can be understood with the help of an abstract model of finite automata. It is shown in Fig. 1.9.3.

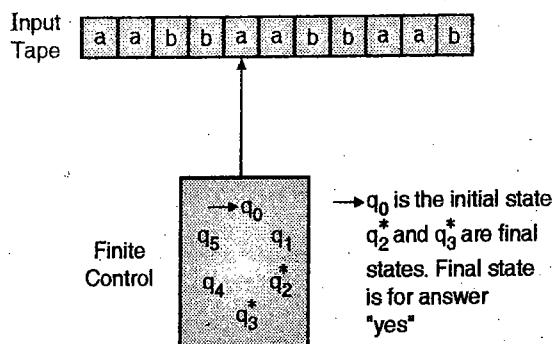


Fig. 1.9.3 : Model of a finite state machine

Operation of the finite automata is given below :

- Input string is fed to the machine through a tape. Tape is divided into squares and each square contains an input symbol.
- The main machine is shown as a box. Machine could be in any of the internal states ($q_0, q_1, q_2, q_3, q_4, q_5$).
- Initially, the machine is in the starting (initial) state q_0 . Reading head is placed at the leftmost square of the tape.
- At regular intervals, the machine reads one symbol from the tape and then enters a new state. Transition to a state depends only on the current state and the input symbol just read.

$$\delta(q_i, A_j) \Rightarrow q_j$$

Machine transits from q_i to q_j on input A_j .

- After reading an input symbol, the reading head moves right to the next square.
- This process is repeated again and again, i.e.
 1. A symbol is read.
 2. State of the machine (finite control) changes.
 3. Reading head moves to the right.
- Eventually, the reading head reaches the end of the input string.
- Now, the automaton has to say 'yes' or 'no'. If the machine ends up in one of a set of final states (q_2, q_3) then the answer is 'yes' otherwise the answer is 'no'.



1.9.1.2 Deterministic Finite Automata (DFA)

The word “deterministic” refers to the fact that the transition is deterministic. There is only one state to which an automaton can transit from the current state on each input. The word “finite” implies that number of states are finite. A finite automata consists of five parts :

1. A finite set of states, represented as Q .
 2. A finite set of alphabet, represented as Σ .
 3. An initial state, represented as q_0 .
 4. A set of accepting states. An accepting state or final state is for ‘yes’ answer. It is a subset of Q and is represented as F .
 5. A next state function or a transition function. Next state depends on the current state and the current input. It is a function from $Q \times \Sigma$ to Q . It is represented as δ .
- $F \subseteq Q$ [F is subset of Q]

1.9.1.3 Definition of a DFA

A deterministic finite automata is a quintuple.

$$M = (Q, \Sigma, \delta, q_0, F), \text{ where}$$

Q is a set of states.

Σ is a set of alphabet.

$q_0 \in Q$ is the initial state,

$F \subseteq Q$ is the set of final states, and δ , the transition function, is a function from $Q \times \Sigma$ to Q .

1.9.1.4 Representation of a DFA

Let the machine M be a deterministic finite automata.

$$M = \{Q, \Sigma, \delta, q_0, F\}, \text{ where}$$

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

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

q_0 is the starting state

$$F = \{q_1\}.$$

and δ is the transition function as given below :

$$\delta(q_0, 0) \Rightarrow q_0$$

$$\delta(q_0, 1) \Rightarrow q_1$$

$$\delta(q_1, 0) \Rightarrow q_1$$

$$\delta(q_1, 1) \Rightarrow q_0.$$

The above representation of transition function is not very readable. Conventionally, there are two representations for transition function :

1. State transition table.
2. State transition diagram.

1. State transition table

The Fig. 1.9.4 shows the state transition table of the transition function discussed in this section.

| | 0 | 1 | Input alphabets. |
|-------------------|-------|-------|---|
| $\rightarrow q_0$ | q_0 | q_1 | \rightarrow before q_0 indicates it is the starting state |
| q_1^* | q_1 | q_0 | * indicates a final state. |

↑
states

Fig. 1.9.4 : State transition table

- o In state transition table, there is a row for every state $q_i \in Q$.
- o In state transition table, there is a column for every alphabet $A_i \in \Sigma$.
- o Starting state is marked as ‘ \rightarrow ’.
- o Final state is marked as ‘*’.

2. State transition diagram

The Fig. 1.9.4(a) shows the state transition diagram of the transition function discussed in this section.

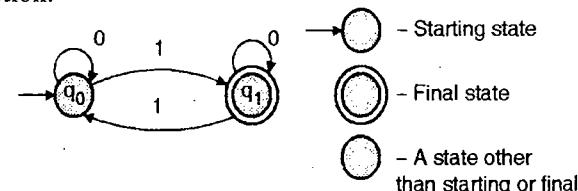


Fig. 1.9.4(a) : State transition diagram

Language of the above DFA

- It is easy to see that machine remain in same state on input 0.
- The machine transitions from q_0 to q_1 and q_1 to q_0 , when the input is 1.
- Machine will be in state q_1 (final state), after reading odd number of 1's.

Thus we can conclude that the DFA accepts a string if and only if the number of 1's is odd.

The language L accepted by M , the given DFA, is the set of all strings having odd number of 1's.

$$\text{Or, } L(M) = \{\omega \in \{0, 1\}^* \mid \omega \text{ contains odd number of 1's}\}$$

Simulating the functioning of the above DFA for a given input

If M is given an input 0010110, its initial configuration is $(q_0, 0010110)$.



| | |
|---|---------------------------|
| $(q_0, 0010110) \xrightarrow{\delta} (q_0, 010110)$ | [input 0 in state q_0] |
| $\xrightarrow{\delta} (q_0, 10110)$ | [input 0 in state q_0] |
| $\xrightarrow{\delta} (q_1, 0110)$ | [input 1 in state q_0] |
| $\xrightarrow{\delta} (q_1, 110)$ | [input 0 in state q_1] |
| $\xrightarrow{\delta} (q_0, 10)$ | [input 1 in state q_1] |

$\xrightarrow{\delta} (q_1, 0)$ [input 1 in state q_0]
 $\xrightarrow{\delta} (q_1, \epsilon)$ [input 0 in state q_1]

Therefore,
 $(q_0, 0010110) \xrightarrow{\delta^*} (q_1, \epsilon)$ and so the input 0010110 is accepted as q_1 is a final state.

□□□

Finite Automata

Syllabus

FSM without output: Definition and Construction-DFA, NFA, NFA with epsilon-Moves, Minimization Of FA, Equivalence of NFA and DFA, Conversion of NFA with epsilon moves to NFA, Conversion of NFA With epsilon moves to DFA.

FSM with output: Definition and Construction of Moore and Mealy Machines, Inter-conversion between Moore and Mealy Machines.

2.1 Introduction to Finite Automata

SPPU – May 12, Dec. 14, Dec. 16

University Questions

- | | |
|--|--------------------------------|
| Q. Define and explain Finite Automata | (May 2012, Dec. 2014, 4 Marks) |
| Q. Define FSM equivalence ? | (May 2012, 4 Marks) |
| Q. Write the formal definition of finite automata. | (Dec. 2014, 2 Marks) |
| Q. Explain the basic finite automata. | (Dec. 2016, 2 Marks) |

Finite automaton or finite state machine can be thought of as a severely restricted model of a computer.

- Every computer has central processing unit; it executes a program stored in a memory. This program normally accepts some input and delivers processed result.
- The word Automata is for automation. A system where energy and information are transformed and used for performing some functions without direct involvement of men is called automaton.
- A finite automata are also called a finite state machine.
- A finite state machine is a mathematical model for actual physical process. By considering the possible inputs on which these machines can work, we can analyse their strengths and weaknesses.
- Finite automata are used for solving several common types of computer algorithms. Some of them are :

 1. Design of digital circuits.
 2. String matching.
 3. Communication protocols for information exchange.
 4. Lexical analyser of a typical compiler.

2.1.1 Working of a Finite Automata

Let us consider a T-flip flop.

A T-flip-flop has :

1. One input denoted by T.
2. Two outputs denoted by Q and \bar{Q} .
3. It has two distinct states, defined by the logical values of Q. The flip-flop is in q_0 state when $Q = 0$ and it is in q_1 state when $Q = 1$.
4. A value of 1 applied to its input changes its state from q_0 to q_1 or from q_1 to q_0 .

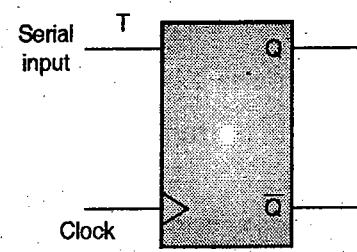


Fig. 2.1.1 : A T-flip flop as a finite automata

A T-flip-flop can be used to check whether an input binary number contains an even number of 1's. Let us assume that a binary number 101110011 is applied to input T of the flip-flop. Initial state of the flip-flop is q_0 (i.e. Q = 0)

| | | | | | | | | | |
|------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Input (T) | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| State (Q) | q_1 | q_1 | q_0 | q_1 | q_0 | q_0 | q_0 | q_1 | q_0 |

Fig. 2.1.2 : State transition of T-flip-flop under the input 101110011

From the Fig. 2.1.2, it should be clear that every even number of 1's in input will make the T-flip-flop toggle twice and thus it will be back to q_0 . Thus we can conclude the following :

1. If the initial state of flip flop is q_0 , it will come back to q_0 if the input number contains even number of 1's.
2. A single T-flip flop can be used as a machine for checking whether a binary number contains an even number of 1's.

The machine starts with initial state q_0 and after feeding the binary number, if the machine is found to be in q_0 , the input binary number contains an even number of 1's.

Working of finite automata can be understood with the help of an abstract model of finite automata. It is shown in Fig. 2.1.3.

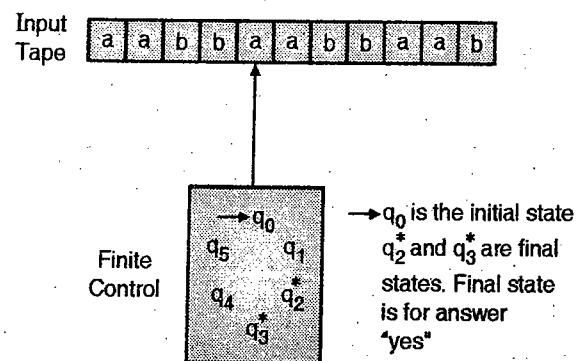


Fig. 2.1.3 : Model of a finite state machine

Operation of the finite automata is given below :

- Input string is fed to the machine through a tape. Tape is divided into squares and each square contains an input symbol.
- The main machine is shown as a box. Machine could be in any of the internal states ($q_0, q_1, q_2, q_3, q_4, q_5$).
- Initially, the machine is in the starting (initial) state q_0 . Reading head is placed at the leftmost square of the tape.
- At regular intervals, the machine reads one symbol from the tape and then enters a new state. Transition to a state depends only on the current state and the input symbol just read.
 $\delta(q_i, A_j) \Rightarrow q_j$
- Machine transits from q_i to q_j on input A_j .
- After reading an input symbol, the reading head moves right to the next square.
- This process is repeated again and again, i.e.
 1. A symbol is read.
 2. State of the machine (finite control) changes.
 3. Reading head moves to the right.
- Eventually, the reading head reaches the end of the input string.
- Now, the automaton has to say 'yes' or 'no'. If the machine ends up in one of a set of final states (q_2, q_3), then the answer is 'yes' otherwise the answer is 'no'.

2.1.2 Some Important Terms

2.1.2.1 Alphabet

University Question

Q. Define and explain alphabet.

SPPU - May 12

(May 2012, 4 Marks)

An alphabet is a finite, non empty set of symbols. Conventionally, the symbol Σ is used for an alphabet. Common alphabet include :



1. $\Sigma \{0, 1\}$, the binary alphabets.
2. $\Sigma \{A, B, \dots, Z\}$, the set of uppercase letters.
3. $\Sigma \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, the decimal alphabets.
4. $\Sigma \{0, 1, 2\}$, the ternary alphabet.

2.1.2.2 Strings (words)

SPPU - Dec. 12

University Question**Q. Explain term 'Word'.**

(Dec. 2012, 2 Marks)

It is a finite sequence of symbols from the alphabet.

For example :

110110 is a string from binary alphabet.

5920 is a string from decimal alphabet.

'STAMP' is a string from Roman alphabet.

- A string may have no symbol at all, in this case it is known as an **empty string** and is denoted by ϵ .
- The **length** of a string is number of positions for symbols in it. The standard notation for the length of a string ω is $|\omega|$.

For example,

If $\omega = 01101$

then $|\omega| = 5$

Length of an empty string is zero.

i.e., $|\epsilon| = 0$.

- The **exponential notation** is used to express the set of all strings of a certain length.

\sum^k = set of strings of length k, each of whose symbol is in Σ .

For example,

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

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

$\Sigma^2 = \{00, 01, 10, 11\}$

and $\Sigma^0 = \{\epsilon\}$

- The **reversal** of string ω is denoted by ω^R .

For example,

If $\omega = xyz$

Then $\omega^R = zyx$.

- A string v is a **substring** of a string ω if and only if there are strings x and y such that,

$\omega = x v y$, where both x and y could be null

Every string is a substring of itself.

For example,

if $\omega = 'abcdef'$ is a string.

then 'cde' is a substring of ω .

- For each string ω , the string ω^i is defined as,

$\omega^0 = \epsilon$, the empty string

$\omega^{i+1} = \omega^i \cdot \omega$ for each $i \geq 0$ where, i is a natural number.



For example,

$$\begin{aligned} \text{if } \omega &= abb \\ \text{then } \omega^0 &= \epsilon \\ \omega^1 &= abb \\ \omega^2 &= \omega \cdot \omega = abbabb \\ \omega^3 &= \omega^2 \cdot \omega = abbabbabb \end{aligned}$$

- The set of all strings over an alphabet Σ is denoted by Σ^* .

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots$$

ε String of length 1 String of q length 2.

For example,

$$\begin{aligned} \text{if } \Sigma &= \{0, 1\} \\ \text{then } \Sigma^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\} \end{aligned}$$

2.1.2.3 Languages

A subset of strings over an alphabet Σ is a **language**. In other words, any subset of Σ^* is a language. If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ .

For example,

$$\begin{aligned} L_1 &= [\omega \in \{0, 1\}^* \mid \omega \text{ has an equal number of 0's and 1's}] \\ \text{i.e., } L_1 &\text{ is a language over alphabets } \{0, 1\}, \text{ having equal number of 0's and 1's.} \\ \therefore L_1 &= \{\epsilon, 01, 10, 0011, 0101, 1100, 1010, \dots\} \end{aligned}$$

Thus, a language can be specified by :

$$L = \{\omega \in \Sigma^* \mid \omega \text{ has the given property}\}$$

Some examples of language are :

1. A language of all strings, where the string represents a binary number.
 $\{0, 1, 00, 01, 10, 11, \dots\}$
2. The set of strings of 0's and 1's with an equal number of each.
 $\{\epsilon, 01, 10, 0011, 0101, 1010, 1100, \dots\}$
3. The set of strings of 0's and 1's, ending in 11.
 $\{11, 011, 111, 0011, 0111, 1011, \dots\}$
4. Σ^* is a language for any alphabet Σ .
5. \emptyset , the empty language, is a language over an alphabet.
6. $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.

2.1.3 Application of Finite Automata

University Questions

SPPU - Dec. 16

Q. What are the various applications of finite automata ?

(Dec. 2016, 2 Marks)

- Finite automata are used for solving several common types of computer algorithms. Some of them are :
- (i) Design of digital circuit
 - (ii) String matching
 - (iii) Communication protocols for information exchange.
 - (iv) Lexical analysis phase of a compiler.

Finite automata can work as an algorithm for regular language. It can be used for checking whether a string $w \in L$, where L is a regular language.



2.1.4 Limitations of Finite Automata

SPPU-Dec.16

University Question

Q. What are the various limitation of finite automata?

(Dec. 2016, 2 Marks)

1. It does not have memory.
2. It cannot modify its own input.
3. It cannot handle context free or context sensitive language.
4. It cannot be used for computation.

2.2 Deterministic Finite Automata (DFA)

The word "deterministic" refers to the fact that the transition is deterministic. There is only one state to which an automaton can transit from the current state on each input. The word "finite" implies that number of states are finite. A finite automata consists of five parts :

1. A finite set of states, represented as Q .
2. A finite set of alphabet, represented as Σ .
3. An initial state, represented as q_0 .
4. A set of accepting states. An accepting state or final state is for 'yes' answer. It is a subset of Q and is represented as F .

$$F \subseteq Q \quad [F \text{ is subset of } Q]$$
5. A next state function or a transition function. Next state depends on the current state and the current input. It is a function from $Q \times \Sigma$ to Q . It is represented as δ .

Syllabus Topic : Definition of DFA

2.2.1 Definition of a DFA

A deterministic finite automata is a quintuple.

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

Q is a set of states.

Σ is a set of alphabet.

$q_0 \in Q$ is the initial state,

$F \subseteq Q$ is the set of final states, and δ , the transition function, is a function from $Q \times \Sigma$ to Q .

2.2.2 Representation of a DFA

Let the machine M be a deterministic finite automata.

$M = \{Q, \Sigma, \delta, q_0, F\}$, where

$Q = \{q_0, q_1\}$

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

q_0 is the starting state

$F = \{q_1\}$.

and δ is the transition function as given below :

$$\delta(q_0, 0) \Rightarrow q_0.$$

$$\delta(q_0, 1) \Rightarrow q_1$$

$$\delta(q_1, 0) \Rightarrow q_1$$

$$\delta(q_1, 1) \Rightarrow q_0.$$

The above representation of transition function is not very readable. Conventionally, there are two representations for transition function :

1. State transition table.
2. State transition diagram.

1. State transition table

The Fig. 2.2.1 shows the state transition table of the transition function discussed in this section.

| | 0 | 1 | Input alphabets. |
|-------------------|-------|-------|---|
| $\rightarrow q_0$ | q_0 | q_1 | \rightarrow before q_0 indicates it is the starting state |
| q_1^* | q_1 | q_0 | * indicates a final state. |
| states | | | |

Fig. 2.2.1 : State transition table

- o In state transition table, there is a row for every state $q_i \in Q$.
- o In state transition table, there is a column for every alphabet $A_i \in \Sigma$.
- o Starting state is marked as ' \rightarrow '.
- o Final state is marked as '*'.

2. State transition diagram

The Fig. 2.2.2 shows the state transition diagram of the transition function discussed in this section.

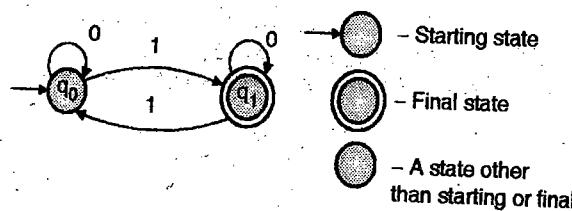


Fig. 2.2.2 : State transition diagram

Language of the above DFA

- It is easy to see that machine remain in same state on input 0.
- The machine transitions from q_0 to q_1 and q_1 to q_0 , when the input is 1.
- Machine will be in state q_1 (final state), after reading odd number of 1's.

Thus we can conclude that the DFA accepts a string if and only if the number of 1's is odd.

The language L accepted by M, the given DFA, is the set of all strings having odd number of 1's.

Or, $L(M) = \{\omega \in \{0, 1\}^* \mid \omega \text{ contains odd number of 1's}\}$

Simulating the functioning of the above DFA for a given input :

If M is given an input 0010110, its initial configuration is $(q_0, 0010110)$.

$$\begin{aligned}
 (q_0, 0010110) &\xrightarrow{\delta} (q_0, 010110) \quad [\text{input } 0 \text{ in state } q_0] \\
 &\xrightarrow{\delta} (q_0, 10110) \quad [\text{input } 0 \text{ in state } q_0] \\
 &\xrightarrow{\delta} (q_1, 0110) \quad [\text{input } 1 \text{ in state } q_0] \\
 &\xrightarrow{\delta} (q_1, 110) \quad [\text{input } 0 \text{ in state } q_1] \\
 &\xrightarrow{\delta} (q_0, 10) \quad [\text{input } 1 \text{ in state } q_1]
 \end{aligned}$$



- $$\begin{aligned}\delta &\Rightarrow (q_1, 0) \quad [\text{input 1 in state } q_0] \\ \delta &\Rightarrow (q_1, \epsilon) \quad [\text{input 0 in state } q_1]\end{aligned}$$

Therefore,

$(q_0, 0010110) \xrightarrow{\delta^*} (q_1, \epsilon)$ and so the input 0010110 is accepted as q_1 is a final state.

Syllabus Topic : Construction of DFA

2.2.3 Designing a DFA

Designing a DFA is like writing a program. In a program, the current state of a program is given by :

1. Values stored in variables.
2. Current line of execution.

In case of a DFA, variables are not used. Everything is remembered through explicit states. Transition function of a DFA is similar to algorithm. While designing a DFA, one has to concentrate on four issues.

These are :

- | | |
|---------------------|------------------------|
| 1. Number of states | 2. Transition function |
| 3. Start state | 4. Final states |

1. Number of states

Number of states depends on "what must be remembered as input symbols are read by a DFA."

For example,

1. DFA to check whether a binary numbers has even number of 1's :

DFA has to remember whether the number of 1's seen so far is even or odd. It does not have to count number of 1's. Let us look at a binary number given below :

| | | | | | | | | | | |
|-----------------|------|-----|------|------|-----|------|------|------|-----|------|
| Binary number → | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Number of 1's → | even | odd | even | even | odd | even | even | even | odd | even |

- Machine will have two states :
 - (i) One corresponding to even number of 1's, so far.
 - (ii) One corresponding to odd number of 1's, so far.
 - These two states are necessary and sufficient to conclude whether a binary number contains even number of 1's.
2. DFA to check whether a string over alphabets (a, b) contains a substring abb :
- DFA has to search for substring abb in input stream. Input string is not stored in a memory variable. Input is made one symbol at a time. As the substring is of length 3, it may be necessary to remember the preceding two symbols.
- Preceding symbol is a. (a is the first symbol in abb).
 - Preceding two symbols are ab. (ab constitutes first two symbols of abb).
 - DFA has already seen the substring abb in the input string. Once the substring abb is found in input string, this status will not change irrespective of what follows thereafter in the input string.
 - Preceding symbols are neither a nor ab, and abb has not come earlier.

The machine will have four states, each standing for one of the four situations.



2. Transition function

Transition function gives the next-state, depending on :

1. Current state
2. Current input

A transition function is problem specific and it depends on the problem.

For example

1. Give transition function for a “DFA to check whether a binary number has even number of 1's”.

We have already seen that the corresponding DFA will have two states.

(i) State q_0 , indicating even number of 1's seen so far.

(ii) State q_1 , indicating odd number of 1's seen so far.

o 0 as next input will have no effect on number of 1's.

o 1 as next input will make the transition from q_0 to q_1 and from q_1 to q_0 . Thus, the transition behaviour can be described using the Fig. 2.2.3.

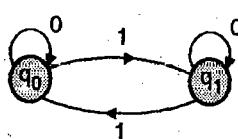
$$\delta(q_0, 0) \Rightarrow q_0$$

$$\delta(q_0, 1) \Rightarrow q_1$$

$$\delta(q_1, 0) \Rightarrow q_1$$

$$\delta(q_1, 1) \Rightarrow q_0$$

| | | |
|-------|-------|-------|
| | 0 | 1 |
| q_0 | q_0 | q_1 |
| q_1 | q_1 | q_0 |



(a) Tabular form

(b) Transition table

(c) Transition diagram

Fig. 2.2.3 : Transition behaviour

2. Give transition function for a “DFA to check whether a string over alphabets (a, b) contains a substring abb”.

We have already seen that the corresponding DFA will have four states.

1. State q_1 , preceding symbol is a.
2. State q_2 , preceding two symbols are ab.
3. State q_3 , substring abb has already been seen in input string.
4. State q_0 , situations q_1 , q_2 , or q_3 are not there.

State of the DFA after every input symbol for a sample input data is shown in Fig. 2.2.4.

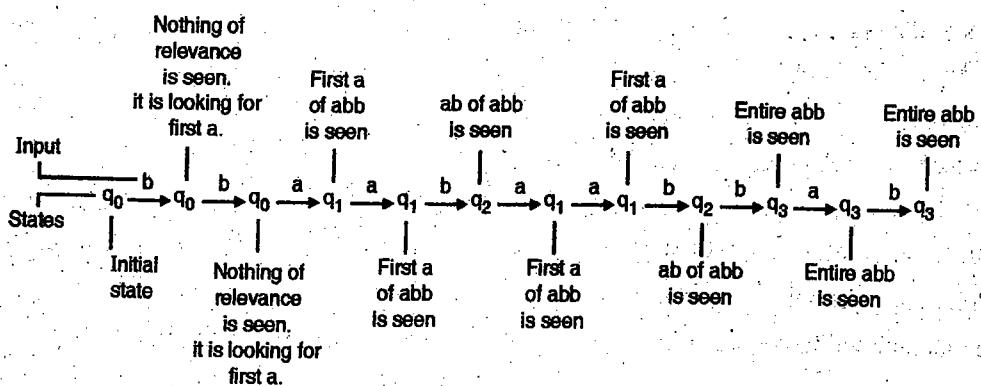


Fig. 2.2.4 : States of DFA

Transition behaviour is described using Fig. 2.2.5.

$$\delta(q_0, b) \Rightarrow q_0$$

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

$$\delta(q_1, a) \Rightarrow q_1$$

$$\delta(q_1, b) \Rightarrow q_2$$

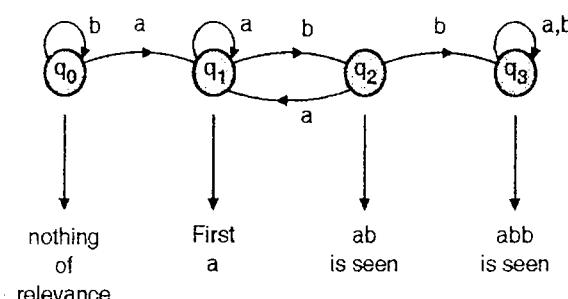
$$\delta(q_2, a) \Rightarrow q_1$$

$$\delta(q_2, b) \Rightarrow q_3$$

$$\delta(q_3, a) \Rightarrow q_3$$

$$\delta(q_3, b) \Rightarrow q_3$$

| | a | b |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_1 | q_0 |
| q_1 | q_1 | q_2 |
| q_2 | q_1 | q_2 |
| q_3^* | q_3 | q_3 |



(a) Tabular form

(b) Transition table

(c) Transition diagram

Fig. 2.2.5 : Transition behaviour

- Machine transits from q_0 to q_1 on input a, as the first symbol a of abb is seen.
- Machine remains in q_1 state on input a, as the previous two symbols aa will still make the first symbol a of abb.
- Machine transits from q_1 to q_2 on input b, as ab of abb is seen.
- Machine transits from q_2 to q_3 on input b, as entire abb is seen.
- Machine transits from q_2 to q_1 on input a, as the previous three symbols aba will still make the first symbol a of aba.

3. Starting state and final states

In Fig. 2.2.3(c), q_0 is both the initial state and final state as :

1. Zero number of 1's is even number of 1's.
2. Machine will have "yes" answer for even number of 1's. q_0 is final state. Machine in Fig. 2.2.3(c) is redrawn in Fig. 2.2.6 with initial and final states marked.

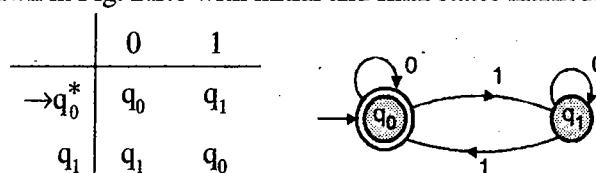


Fig. 2.2.6 : Final DFA for Fig. 2.2.3

In Fig. 2.2.5(c), q_0 is initial state and q_3 is the final state.

1. Initially, we have nothing that is relevant to substring abb. Therefore, q_0 is initial state.
2. Machine goes to state q_3 after seeing the substring abb. Therefore, q_3 is final state.

Machine in Fig. 2.2.5 is redrawn in Fig. 2.2.7 with initial and final states marked.

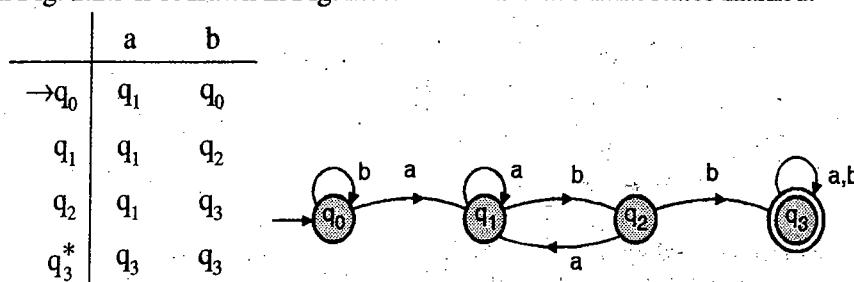


Fig. 2.2.7 : Final DFA for Fig. 2.2.5(c)

2.2.4 Solved Examples on DFA

2.2.4.1 Examples on Counting of Symbols

Example 2.2.1

Give deterministic finite automata accepting the following language over the alphabet $\{0, 1\}$.

- Number of 1's is multiple of 3.
- Number of 1's not multiple of 3.

Solution :

- Number of 1's is multiple of 3.

Number of 1's seen, so far by the machine can be written as :

- $3n$
- $3n + 1$
- $3n + 2$

Corresponding to three cases mentioned above, there will be three states.

State q_0 – no. of 1's, so far is $3n$

State q_1 – no. of 1's, so far is $3n + 1$

State q_2 – no. of 1's, so far is $3n + 2$

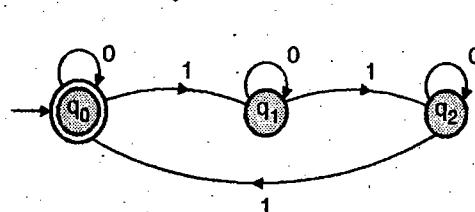
An input of 1 will cause a transition from :

q_0 to q_1 , if the machine is in q_0 .

q_1 to q_2 , if the machine is in q_1 .

q_2 to q_0 , if the machine is in q_2 .

An input of 0 will not cause any transition.



(a) Transition diagram

| | 0 | 1 |
|---------------------|-------|-------|
| $\rightarrow q_0^*$ | q_0 | q_1 |
| q_1 | q_1 | q_2 |
| q_2 | q_2 | q_0 |

(b) Transition table

Fig. Ex. 2.2.1 : Final DFA for Example 2.2.1(a)

- o q_0 is the starting state. Zero number of 1's implies that number of 1's is of the form $3n$.
- o q_0 is a final state. Machine will be in q_0 if number of 1's seen so far is multiple of 3.

- Number of 1's is not multiple of three. Let L be a language over alphabets $\{0, 1\}$ such that number of 1's is multiple of 3.

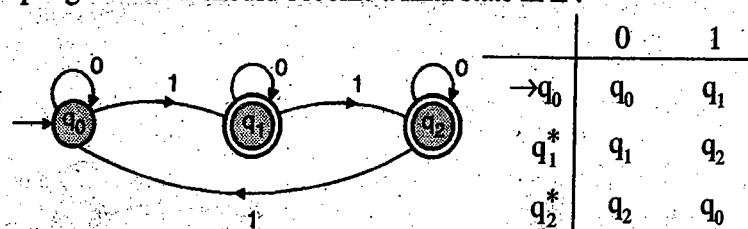
Thus, $L = [\omega \in \{0, 1\}^* \mid \text{no. of 1's in } \omega \text{ is multiple of 3}]$

Complement of L is given by :

$$L' = [\omega \in \{0, 1\}^* \mid \text{no. of 1's in } \omega \text{ is not multiple of 3}]$$

DFA for L' can be obtained through following modifications on DFA for L .

- Every final state of L should become a non-accepting state in L' .
- Every non-accepting state of L should become a final state in L' .



(c) Transition diagram

| | 0 | 1 |
|-------------------|-------|---------|
| $\rightarrow q_0$ | q_0 | q_1^* |
| q_1^* | q_1 | q_2^* |
| q_2^* | q_2 | q_0 |

(d) Transition table

Fig. Ex. 2.2.1 : Final DFA for Example 2.2.1(b)

Example 2.2.2

Give deterministic finite automata accepting the following languages over the alphabet {0, 1}

- (a) Number of 1's is even and number of 0's is even.
- (b) Number of 1's is odd and number of 0's is odd.

Solution :

- (a) Number of 1's is even and number of 0's is even.

At any instance of time, we will have following cases for number of 0's and number of 1's seen by the machine.

| Situations | | State |
|---------------|---------------|-------|
| Number of 0's | Number of 1's | |
| Even | Even | q_0 |
| Even | Odd | q_1 |
| Odd | Even | q_2 |
| Odd | Odd | q_3 |

- An input 0 in state q_0 , will make number of 0's odd.

$$\delta(q_0, 0) \Rightarrow q_1$$

- An input 1 in state q_0 , will make number of 1's odd.

$$\delta(q_0, 1) \Rightarrow q_2$$

- An input 0 in state q_1 , will make number of 0's odd.

$$\delta(q_1, 0) \Rightarrow q_3$$

- An input 1 in state q_1 , will make number of 1's even.

$$\delta(q_1, 1) \Rightarrow q_0$$

- An input 0 in state q_2 , will make number of 0's even.

$$\delta(q_2, 0) \Rightarrow q_0$$

- An input 1 in state q_2 , will make number of 1's odd.

$$\delta(q_2, 1) \Rightarrow q_3$$

- An input 0 in state q_3 , will make number of 0's even.

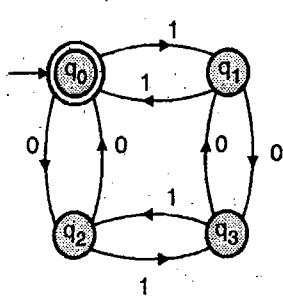
$$\delta(q_3, 0) \Rightarrow q_1$$

- An input 1 in state q_3 , will make number of 1's even.

$$\delta(q_3, 1) \Rightarrow q_2$$

- q_0 is the starting state. An empty string contains even number of 0's and even number of 1's.

- q_0 is a final state. q_0 stands for even number of 0's and even number of 1's.



(a) Transition diagram

| | | |
|-------------------|-------|-------|
| | 0 | 1 |
| $\rightarrow q_0$ | q_2 | q_1 |
| q_1 | q_3 | q_0 |
| q_2 | q_0 | q_3 |
| q_3 | q_1 | q_2 |

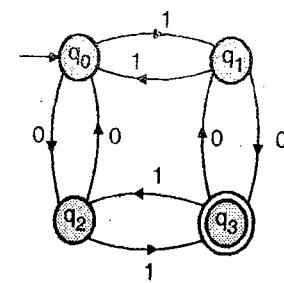
(b) Transition table

Fig. Ex. 2.2.2 : Final DFA for Example 2.2.2(a)



(b) Number of 1's is odd and number of 0's is odd.

In solution of Example 2.2.2(a), the state q_3 stands for odd number of 0's should be declared as final state.



(c) Transition diagram

| | 0 | 1 |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_2 | q_1 |
| q_1 | q_3 | q_0 |
| q_2 | q_0 | q_3 |
| q_3^* | q_1 | q_2 |

(d) Transition table

Fig. Ex. 2.2.2 : Final DFA for Example 2.2.2(b)

Example 2.2.3

Design a DFA which accepts the odd number 1's and any number of 0's over $\Sigma = \{0, 1\}$.

Solution :

The DFA must keep track of number of 1's in the string already seen by it.

- The number of 1's seen could be even, state q_0 .
- The number of 1's seen could be odd, state q_1 .

The required DFA is given in Fig. Ex. 2.2.3.

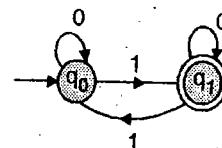


Fig. Ex. 2.2.3

Example 2.2.4

Design a DFA for a set of strings over alphabet $\{0, 1\}$ such that the number of 0's is divisible by five, and number of 1's divisible by 3.

Solution :

At any instance of time, we will have following cases for number of 0's.

Case 1 – $5n$

Case 2 – $5n + 1$

Case 3 – $5n + 2$

Case 4 – $5n + 3$

Case 5 – $5n + 4$

Number of 0's should be divisible by 5.

Similarly, there will be three cases for number of 1's.

Case 1 – $3m$

Case 2 – $3m + 1$

Case 3 – $3m + 2$

Number of 1's is divisible by 3.

Thus, depending on number of 0's and 1's there will be $5 \times 3 = 15$ cases.

Let us represent a state of the DFA under consideration as q_{ij} , where i can take a value from 0 to 4 depending on number of 0's seen so far :

q_{0j} – number of 0's is $5n$

q_{1j} – number of 0's is $5n + 1$

q_{2j} – number of 0's is $5n + 2$

q_{3j} – number of 0's is $5n + 3$

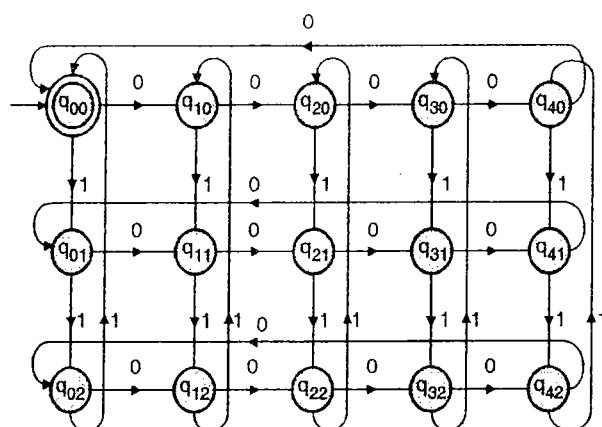
q_{4j} – number of 0's is $5n + 4$

Similarly, in state q_{ij} can take value from 0 to 2 depending on number of 1's seen so far.

q_{i0} – number of 1's is $3m$

q_{i1} – number of 1's is $3m + 1$

q_{i2} – number of 1's is $3m + 2$



(a) State transition diagram

(b) State transition table

Fig. Ex. 2.2.4 : Final DFA for Example 2.2.4

Example 2.2.5Draw DFA for the following language over $\{0, 1\}$:

- All strings of length at most five
- All strings with exactly two 1's
- All string containing at least two 0's.
- All strings containing at most two 0's.
- All strings starting with 1 and length of the string is divisible by 3.

Solution :

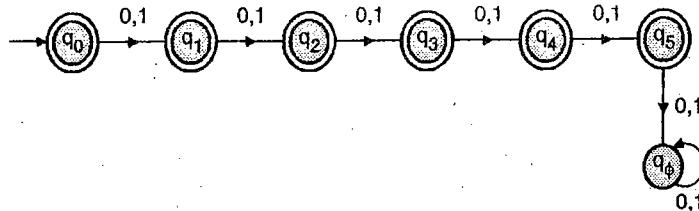
- All strings of length at most five**

These are the valid strings :

- String of length 0 – accept through q_0 .
- String of length 1 – accept through q_1 .
- String of length 2 – accept through q_2 .
- String of length 3 – accept through q_3 .
- String of length 4 – accept through q_4 .
- String of length 5 – accept through q_5 .

String of length > 5 should be rejected through a dead state or a failure state. A failure state is designated as q_ϕ .

| | 0 | 1 |
|-------------------------|----------|----------|
| $\rightarrow q_{(0)}^*$ | q_{10} | q_{01} |
| q_{01} | q_{11} | q_{02} |
| q_{02} | q_{12} | q_{00} |
| q_{10} | q_{20} | q_{11} |
| q_{11} | q_{21} | q_{12} |
| q_{12} | q_{22} | q_{10} |
| q_{20} | q_{30} | q_{21} |
| q_{21} | q_{31} | q_{22} |
| q_{22} | q_{32} | q_{20} |
| q_{30} | q_{40} | q_{31} |
| q_{31} | q_{11} | q_{32} |
| q_{32} | q_{42} | q_{30} |
| q_{40} | q_{00} | q_{41} |
| q_{41} | q_{01} | q_{42} |
| q_{42} | q_{02} | q_{40} |



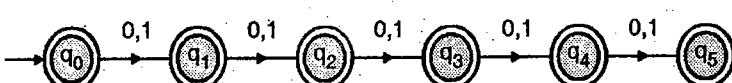
(a) State transition diagram

| | 0 | 1 |
|---------------------|----------|----------|
| $\rightarrow q_0^*$ | q_1 | q_1 |
| 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_ϕ | q_ϕ |
| q_ϕ | q_ϕ | q_ϕ |

(b) State transition table

Fig. Ex. 2.2.5(a) : Final DFA for Example 2.2.5(i) with explicit failure state

It is not necessary to mention the failure/dead state explicitly. If a transition is not mentioned, it is taken as a failure transition. Thus, the solution can be given without explicit failure state.



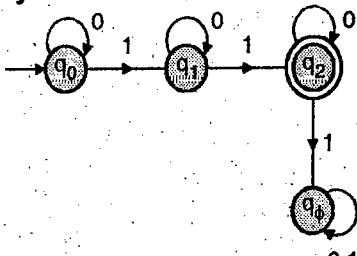
(c) State transition diagram

| | 0 | 1 |
|---------------------|-------------|-------------|
| $\rightarrow q_0^*$ | q_1 | q_1 |
| 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^* | \emptyset | \emptyset |

(d) State transition table

Fig. Ex. 2.2.5 : Final DFA for Example 2.2.5(i) without a failure state or with failure transition omitted.

(ii) All strings with exactly two 1's



(e) State transition diagram

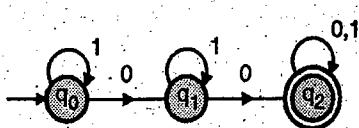
| | 0 | 1 |
|---------------------|----------|----------|
| $\rightarrow q_0^*$ | q_0 | q_1 |
| q_1 | q_1 | q_2 |
| q_2^* | q_2 | q_ϕ |
| q_ϕ | q_ϕ | q_ϕ |

(f) State transition table

Fig. Ex. 2.2.5 : Final DFA for Example 2.2.5(ii)

- First 1 takes the machine from q_0 to q_1 .
- Second 1 takes the machine from q_1 to q_2 .
- q_2 is a final state.
- A 1 in q_2 takes the machine to a failure state q_ϕ .

(iii) All string containing at least two 0's



(g) State transition diagram

| | 0 | 1 |
|---------------------|-------|-------|
| $\rightarrow q_0^*$ | q_1 | q_0 |
| q_1 | q_2 | q_1 |
| q_2^* | q_2 | q_2 |

(h) State transition table

Fig. Ex. 2.2.5 : Final DFA for Example 2.2.5(iii)

- First two 0's will take the machine from q_0 to q_2 . Thereafter, the machine remains in q_2 as the string should contain at least two 0's.

(iv) All strings containing at most two 0's

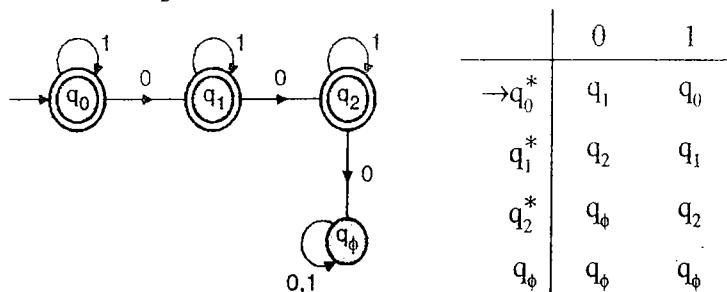
These are the valid strings :

A string not containing 0 – accept through q_0 .

A string containing one 0 – accept through q_1 .

A string containing two 0's – accept through q_2 .

Another occurrence of 0 in state q_2 will cause a transition to dead state.



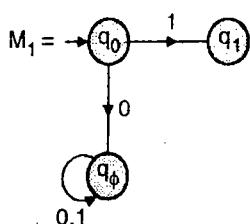
(i) State transition diagram

(j) State transition table

Fig. Ex. 2.2.5 : Final DFA for Example 2.2.5(iv)

(v) All strings starting with 1 and length of the string is divisible by 3

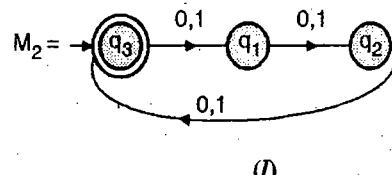
Step 1 : A machine M_1 , representing a DFA accepting a string starting with 1 is given in Fig. Ex. 2.2.5(k) :



A 0 as the first input takes the machine to a dead state

Fig. Ex. 2.2.5(k)

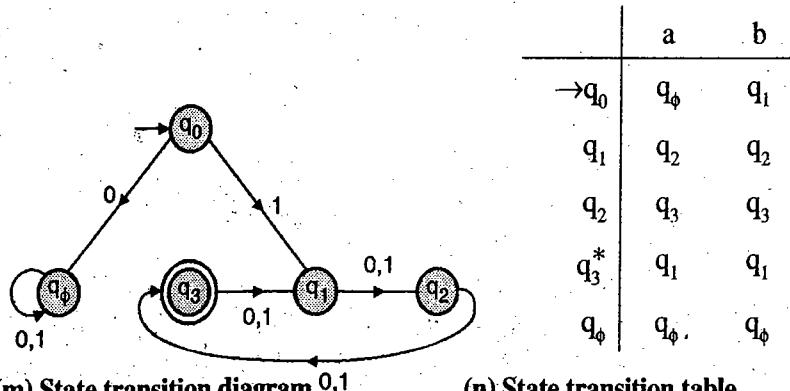
Step 2 : A machine M_2 , representing a DFA accepting a strings starting having length divisible by 3 is given in Fig. Ex. 2.2.5(l) :



(l)

To design the required machine, we can combine M_1 and M_2 .

- Merge q_1 of M_1 with q_1 of M_2 .
- Initial input is handled by M_1 , subsequent inputs are handled by M_2 .



(m) State transition diagram

(n) State transition table

Fig. Ex. 2.2.5 : Final DFA for Example 2.2.5(v)

2.2.4.2 Examples on Substring

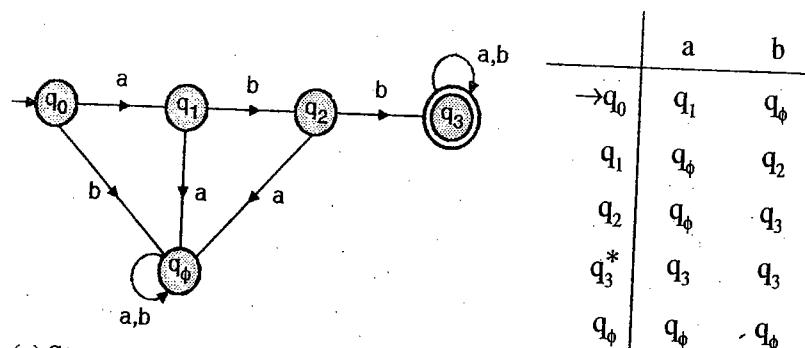
Example 2.2.6

Draw DFA for the following language over $\{a, b\}$:

- All strings starting with abb.
- All strings with abb as a substring i.e., abb anywhere in the string.
- All strings ending in abb.

Solution : (a) All strings starting with abb

- First input as 'b' will take the machine to a failure state.
- First two inputs as 'aa' will take the machine to a failure state.
- First three inputs as 'aba' will take the machine to a failure state.
- First three inputs as 'abb' will take the machine to a final state.

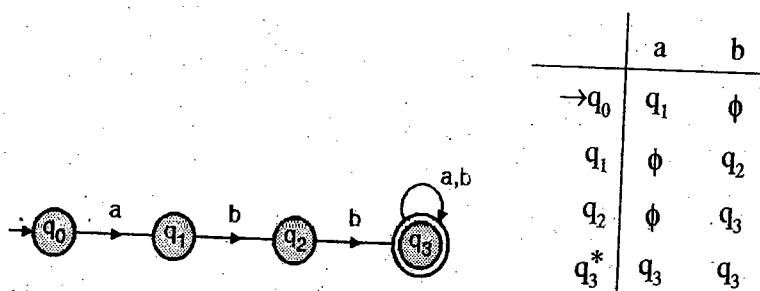


(a) State transition diagram

(b) State transition table

Fig. Ex. 2.2.6 : Final DFA for Example 2.2.6(a)

A DFA without explicit failure state is given in Fig. Ex. 2.2.6(a).



(c) State transition diagram

(d) State transition table

Fig. Ex. 2.2.6 : Final DFA for Example 2.2.6(a), without a failure / dead state

(b) All strings with abb as a substring :

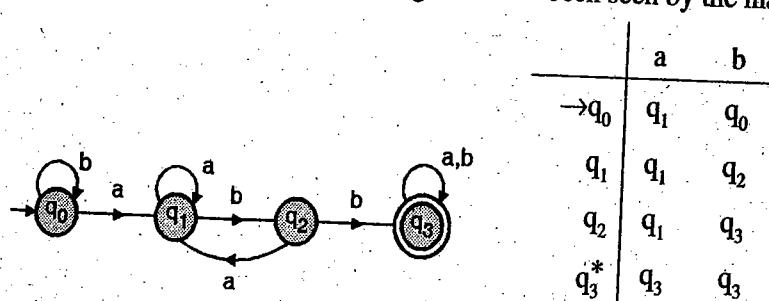
The machine will have fours states :

State q_0 – It is the starting state and indicates that nothing of relevance to complete 'abb' has been seen.

State q_1 – preceding character is 'a' and 'bb' is required to complete 'abb'.

State q_2 – Preceding characters are 'ab' and 'b' is required to complete 'abb'.

State q_3 – Preceding characters are 'abb' and the substring 'abb' has been seen by the machine.



(e) State transition diagram

(f) State transition table

Fig. Ex. 2.2.6 : Final DFA for example 2.2.6(b)



q_0 to q_0 on input 'b' :

First character in 'abb' is a.

q_0 to q_1 on input 'a' :

q_1 is for preceding characters as 'a', first character of abb.

q_1 to q_1 on input 'a' :

An input of 'a' in state q_1 will make the preceding two characters as 'aa'. Last 'a' will still constitute the first 'a' of abb.

q_1 to q_2 on input 'b' :

q_2 is for preceding two characters as 'ab' of 'abb'.

q_2 to q_1 on input 'a' :

An input 'a' in q_2 will make the preceding three characters as 'aba'. Out of the three characters 'aba', only the last character 'a' is relevant to 'abb'.

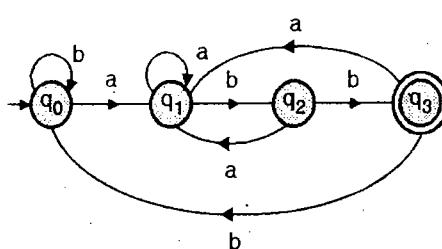
q_2 to q_3 on input b :

q_3 is for preceding three characters as 'abb'.

q_3 to q_3 on input a or b :

The substring 'abb' has been seen by the machine and a new input will not change this status.

(c) All strings ending in abb : This can be seen as an extension of solution which is given in Example 2.2.6(b) as the substring 'abb' should be at the end of the string. Transitions from q_3 should be modified to handle the condition that the string has to end in 'abb'.



(g) State transition diagram

| | a | b |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_1 | q_0 |
| q_1 | q_1 | q_2 |
| q_2 | q_1 | q_3 |
| q_3^* | q_1 | q_0 |

(h) State transition table

Fig. Ex. 2.2.6 : Final DFA for Example 2.2.6(c)

q_3 to q_1 on input a :

An input of a in q_3 will make the previous four characters as 'abba'. Out of the four characters as 'abba' only the last character 'a' is relevant to 'abb'.

q_3 to q_0 on input b :

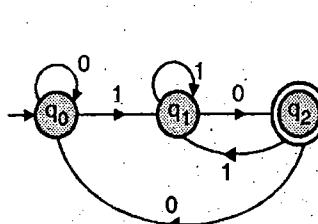
An input of b in q_3 will make the previous four characters 'abbb'. Out of the four characters 'abbb', nothing is relevant to 'abb'.

Example 2.2.7

Design DFA for a language of string 0 and 1 that :

- (I) Ending with 10
- (II) Ending with 11
- (III) Ending with 1

Solution : (I) Ending with 10



| | 0 | 1 |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_0 | q_1 |
| q_1 | q_2 | q_1 |
| q_2^* | q_0 | q_1 |

(a) State transition diagram

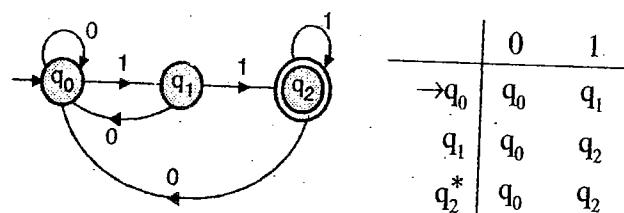
Fig. Ex. 2.2.7 : DFA for string ending with 10

Meaning of various states

q_0 : Starting state, nothing of sequence 10 is seen.

q_1 : '1' of sequence 10 is seen.

q_2 : Final state for strings ending with 10.

(II) Ending with 11

(b) State transition diagram

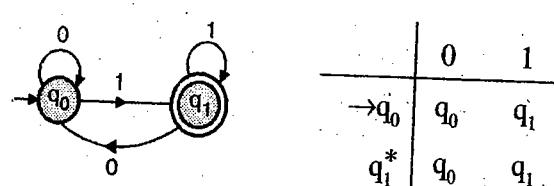
Fig. Ex. 2.2.7 : DFA for string ending with 11

Meaning of various states

q_0 : Starting state, nothing of sequence 11 is seen.

q_1 : First '1' of sequence '11' is seen

q_2 : Final state of strings ending with '11'

(III) Ending with 1

(c) State transition diagram

Fig. Ex. 2.2.7 : DFA for string ending with 1

- The DFA will be in state q_1 , whenever the preceding symbol is 1.

Example 2.2.8

Design the DFA which accepts set of strings such that every string containing 00 as a substring but not 000 as a substring.

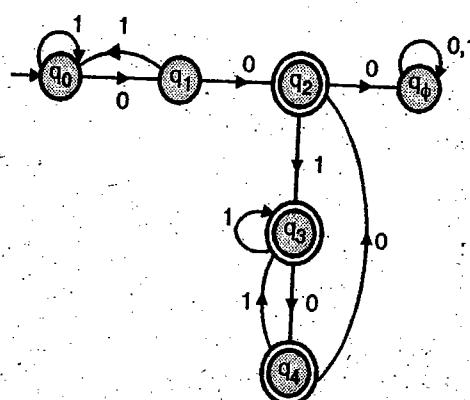
Solution :

Fig. Ex. 2.2.8 : Final DFA for Example 2.2.8

- A sequence of 00 takes the machine from q_0 to q_2 .
- q_2 is a final state.
- A '0' input in q_2 will cause a substring 000.
- An input of 1 in state q_2 causes a transition from q_2 to q_3 . If starting from q_3 , a substring '000' is detected, machine goes to the failure state q_4 .

Example 2.2.9

Design the DFA for the language, containing strings in which leftmost symbol differ from rightmost symbol. Σ is given by $\{0, 1\}$.

Solution :

- Machine will end in the final state q_2 if the leftmost symbol is 0 and the rightmost symbol is 1.
- Machine will end in the final state q_4 if the leftmost symbol is 1 and the rightmost symbol is 0.
- A transition from q_0 to q_1 is for '0' as the first symbol.
- A transition from q_0 to q_3 for '1' as the first symbol.

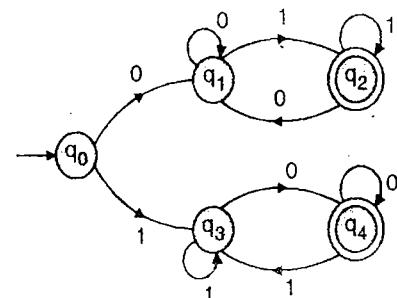
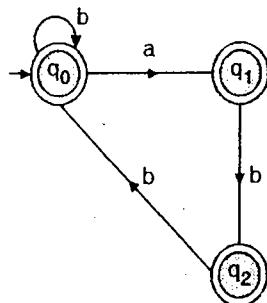


Fig. Ex. 2.2.9 : Final DFA for Example 2.2.9

Example 2.2.10

Design a DFA for set of strings over $\{a, b\}$ in which there are at least two occurrences of b between any two occurrences of a.

Solution :



(a) State transition diagram

| | a | b |
|---------------------|-------------|-------|
| $\rightarrow q_0^*$ | q_1 | q_0 |
| q_1^* | \emptyset | q_2 |
| q_2^* | \emptyset | q_0 |

(b) State transition table

Fig. Ex. 2.2.10 : Final DFA (without explicit failure state) for Example 2.2.10

- An input 'a' in q_0 takes the machine from q_0 to q_1 . Before the next 'a' can come, there should be at least two b's taking the machine from q_1 to q_2 and from q_2 to q_0 .
- An input 'a' in either q_1 or q_2 causes a failure.
- All the three states are 'accepting states'.

Example 2.2.11

Design a DFA for set of all strings over $\{a, b\}$ ending in either ab or ba.

Solution :

Meaning of different states :

$q_0 \rightarrow$ starting state

$q_1 \rightarrow a$ of sequence ab

$q_2 \rightarrow ab$ of sequence ab

$q_3 \rightarrow b$ of sequence ba

$q_4 \rightarrow ba$ of sequence ba

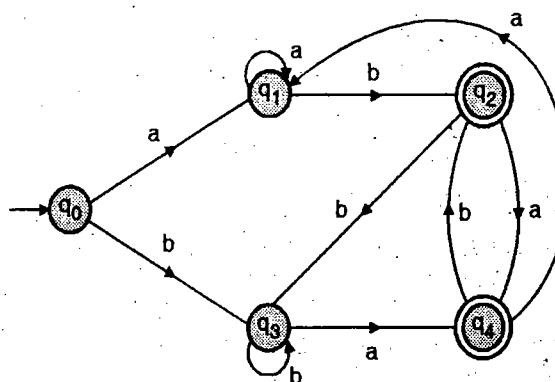


Fig. Ex. 2.2.11 : DFA for Example 2.2.11

Transitions

- Input a in q_0 takes the machine to q_1 as the first character 'a' of 'ab' is the preceding character.
- Input b in q_0 takes the machine to q_3 as the first character 'b' of 'ba' is the preceding character.



- Input 'a' in q_1 makes the preceding two characters as 'aa'. Out of 'aa', only the last character 'a' is relevant to 'ab' and hence the machine requires in q_1 .
- Input 'b' in q_1 makes the preceding two characters as 'ab'. Machine enters the state q_2 which stands for previous two characters as ab.
- Input 'a' in q_2 makes the preceding two characters as 'ba'. Machine enters the state q_4 which stands for previous two characters as ba.
- Input b in q_2 makes the preceding two characters as 'bb'. Out of 'bb', only the last character 'b' is relevant to 'ba' and hence the machine enters the state q_3 .
- Similar explanation can be given for q_3 and q_4 .

Example 2.2.12

Design an DFA for set of all strings over {a, b} containing both ab and ba as substrings.

Solution :

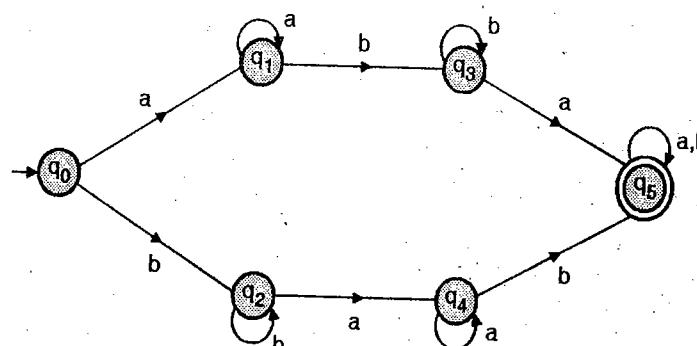


Fig. Ex. 2.2.12 : DFA for Example 2.2.12

- Machine takes the path $q_0 - q_1 - q_3 - q_5$, if the first substring is ab and the next substring is ba.
- Machine takes the path $q_0 - q_2 - q_4 - q_5$, if the first substring is ba and the next substring is ab.

Example 2.2.13

Design a FA that reads string defined over $\Sigma = \{a,b\}$ and accepts only those strings which end up in either 'aa' or 'bb'.

Solution :

Meaning of different states :

- $q_0 \rightarrow$ starting state
- $q_1 \rightarrow$ a of sequence aa is seen.
- $q_2 \rightarrow$ aa of sequence aa is seen.
- $q_3 \rightarrow$ b of sequence bb is seen.
- $q_4 \rightarrow$ bb of sequence bb is seen.

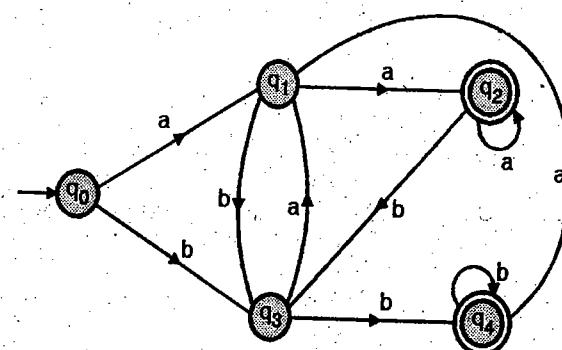


Fig. Ex. 2.2.13

Example 2.2.14

Design a DFA for set of all strings over {a, b} containing neither aa nor bb as a substring.

Solution :

- Machine enters the failure state q_4 on seeing either aa or bb.
- An input b in state q_1 will make preceding two characters as bb.
- An input a in state q_2 will make the preceding two characters as aa.
- Machine moves from q_1 to q_2 and q_2 to q_1 as long as the input sequence contains a and b alternatively.

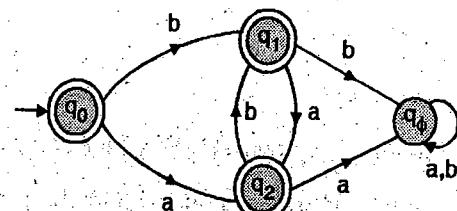
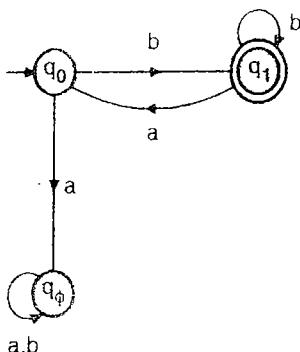


Fig. Ex. 2.2.14 : DFA for Example 2.2.14

Example 2.2.15

Design a DFA for set of all strings over $\{a, b\}$ such that each 'a' in ω is immediately preceded and immediately followed by a 'b'.

Solution :



(a) State transition diagram

| | a | b |
|-------------------|----------|----------|
| $\rightarrow q_0$ | q_ϕ | q_1 |
| q_1^* | q_0 | q_1 |
| q_ϕ | q_ϕ | q_ϕ |

(b) State transition table

Fig. Ex. 2.2.15 : DFA for Example 2.2.15

Example 2.2.16 SPPU - May 15, Dec. 16, 4 Marks

Design a DFA for accepting L over $\{0, 1\}$ such that every substring of length 4 contains at least three 1's.

Solution :

To solve this problem, we must ensure that there are at least three 1's between every pair of 0's.

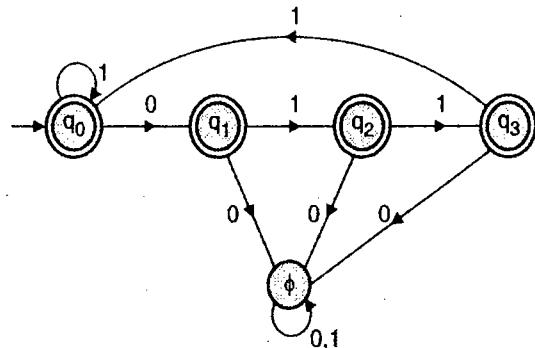


Fig. Ex. 2.2.16

Example 2.2.17 SPPU - Dec. 13, 10 Marks

- Design a DFA for set of all strings over $\{0, 1\}$ such that every block of five consecutive symbols contains at least two 0's.
- Design a DFA for following language, $L = \{ W \mid W \text{ is Binary word of length } 4i \text{ (where } i \geq 1\text{)} \text{ such that each consecutive block of 4 bits contains atleast 2 0's} \} = \{0000, 0110, 01101100, \dots\}$

Solution :

- To solve this problem, we must maintain a count of :

- Number of 1's before 0.
- Number of 1's after 0.

If the sum total of number of 1's before 0 and number of 1's after 0 is 4, the machine enters a failure state.

Meaning of various states :

$q_0 \rightarrow$ number of 1's before 0 is zero.

$q_4 \rightarrow$ number of 1's before 0 is one.

$q_7 \rightarrow$ number of 1's before 0 is two.

$q_9 \rightarrow$ number of 1's before 0 is three.

$q_5 \rightarrow$ number of 1's before 0 is one and after zero is 1.

$q_6 \rightarrow$ number of 1's before 0 is one and after zero is 2.

$q_8 \rightarrow$ number of 1's before 0 is two and after zero is 1.

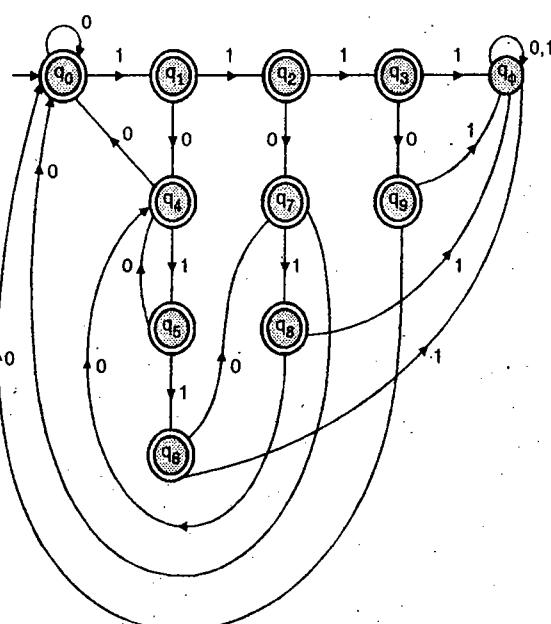


Fig. Ex. 2.2.17 : DFA for Example 2.2.17

An input 0 in $q_4, q_5, q_6, q_7, q_8, q_9$ will make number of 1's after 0 as number of 1's before 0.

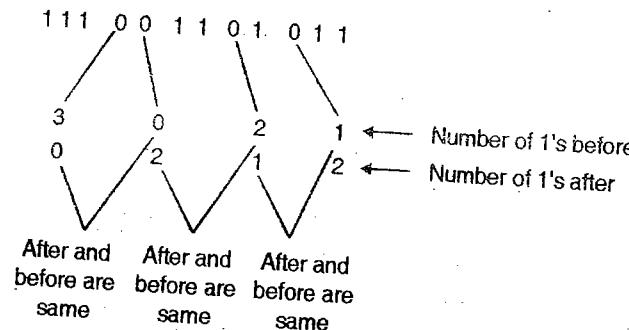


Fig. Ex. 2.2.17(a)

- (ii) To solve this problem, we must maintain a count of :
1. Number of 1's before 0.
 2. Number of 1's after 0.

If the sum total of number of 1's before 0 and number of 1's after 0 is 3, the machine enters a failure state.

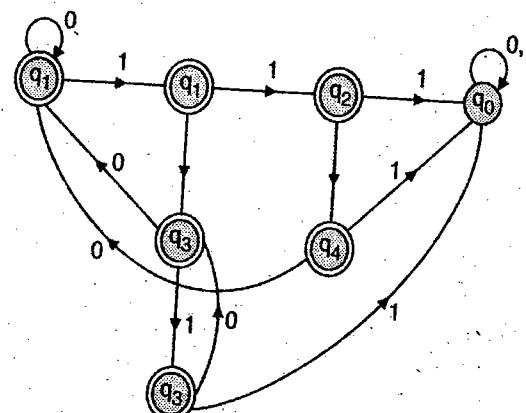


Fig. Ex. 2.2.17(b)

Example 2.2.18 SPPU – Dec. 12, 10 Marks

Obtain a DFA to accept strings of a's and b's such that
 $L = \{W/W \in (a+b)^* \text{ such that } N_a(W) \bmod 3 = 0 \text{ and } N_b(W) \bmod 2 = 0\}$

Solution :

In the required strings, no. of a's are divisible by 3 and no. of b's are divisible by 2.

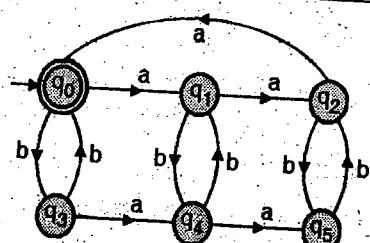


Fig. Ex. 2.2.18

Example 2.2.19

Design a DFA for set of all strings over $\{0, 1\}$ such that strings either begin or end (or both) with 01.

Solution :

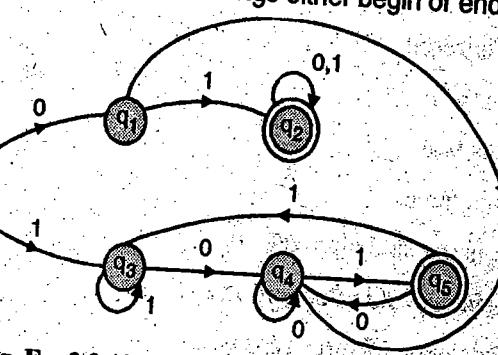


Fig. Ex. 2.2.19 : DFA for Example 2.2.19



- Path $q_0 - q_1 - q_2$ is for strings beginning with 01.
- Final state q_5 is for strings ending in 01.
- State q_3 – Nothing relevant to '01' is seen.
- State q_4 – First character 0 of 01 is the preceding character.
- State q_5 – preceding two characters are 01.

Example 2.2.20

Design a DFA for set of all strings over {0, 1} such that the third symbol from the right end is 1.

Solution :

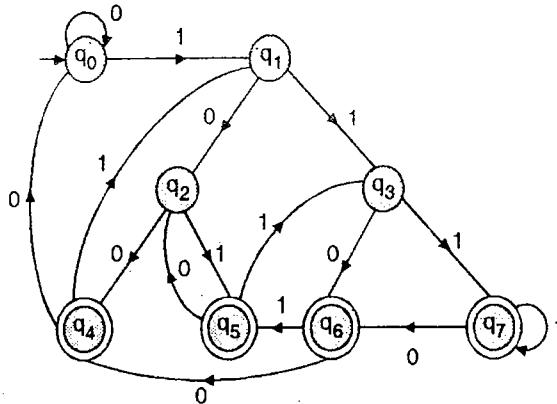


Fig. Ex. 2.2.20 : DFA for Example 2.2.20

Following 4 combinations for the preceding three characters are accepting.

100, 101, 110, 111.

- Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_4$ is for 100.
- Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_5$ is for 101.
- Path $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_6$ is for 110.
- Path $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_7$ is for 111.

State number is same as the value of preceding three inputs.

Transitions

- An input 0 in q_4 will make the preceding three characters as 000, machine moves to state q_0 .
- An input 1 in q_4 will make the preceding three characters as 001, machine moves to state q_1 .
- An input 0 in q_5 will make the preceding three characters as 010, machine moves to state q_2 .
- An input 1 in q_5 will make the preceding three characters as 011, machine moves to state q_3 .
- An input 0 in q_6 will make the preceding three characters as 100, machine moves to state q_4 .
- An input 1 in q_6 will make the preceding three characters as 101, machine moves to state q_5 .
- An input 0 in q_7 will make the preceding three characters as 110, machine moves to state q_6 .
- An input 1 in q_7 will make the preceding three characters as 111, machine moves in state q_7 .

Example 2.2.21 | SPPU - Aug. 15, 4 Marks

Construct a deterministic finite automata (DFA) that recognizes the language $L = \{x \in (0,1)^* \mid (x \text{ contains atleast two consecutive } 0's) \text{ and } (x \text{ does not contain two consecutive } 1's)\}$.

Solution : The required DFA is given below.

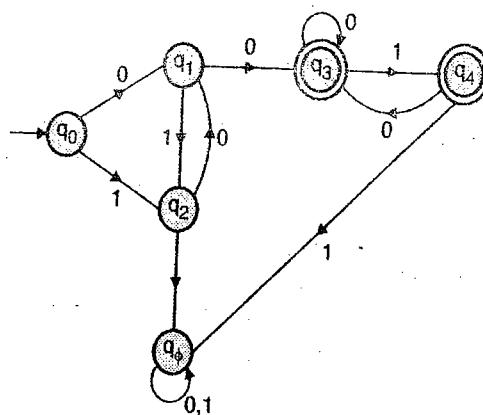


Fig. Ex. 2.2.21

- In case of two consecutive 1's the machine will enter the failure state q_6 .
- If the string contains two consecutive 1's, the machine will enter the state q_3 .

Example 2.2.22

Construct a DFA for set of strings containing either the substring 'aaa' or 'bbb'.

Solution :

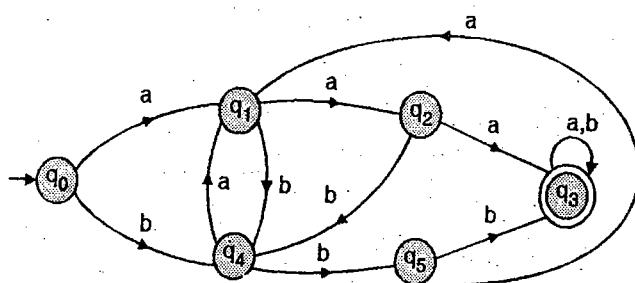


Fig. Ex. 2.2.22 : DFA for Example 2.2.22

Meaning of various states :

$q_0 \rightarrow$ Starting state.

$q_1 \rightarrow$ previous character is a of 'aaa'.

$q_2 \rightarrow$ previous two character are 'aa' of 'aaa'.

$q_4 \rightarrow$ previous character is b of 'bbb'.

$q_5 \rightarrow$ previous two character are 'bb' of 'bbb'.

$q_3 \rightarrow$ substring 'aaa' or 'bbb' is seen.

- An input 'b' in q_0 , q_1 or q_2 moves the machine to q_4 as b is the first character of the sequence bbb.
- An input 'a' in q_0 , q_4 or q_5 moves the machine to q_1 as 'a' is the first character of the sequence aaa.

Example 2.2.23 SPPU- Dec. 12, 2 Marks

Construct a DFA for accepting a set of strings over alphabet {0,1}

- Strings not ending with 010
- Strings ending with 101

Solution :

- Strings not ending with 010 :** Above DFA can be constructed in two steps :

1. DFA for strings ending in 010.
2. By taking complement of DFA derived in step 1; make every final state as non-final state and non-finish state as final state.

Step 1 : DFA for accepting strings ending in 010.

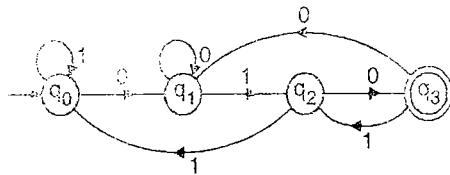


Fig. Ex. 2.2.23(a) : DFA for strings ending in 010

Step 2 : Complementing the DFA by reversing a non-final state to final state and a final state to non-final state.

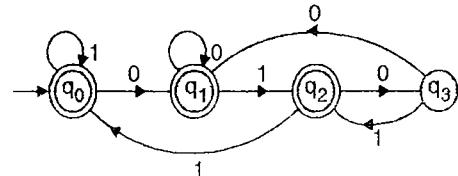


Fig. Ex. 2.2.23(b) : DFA for strings not ending in 010

(ii) Strings ending with 101

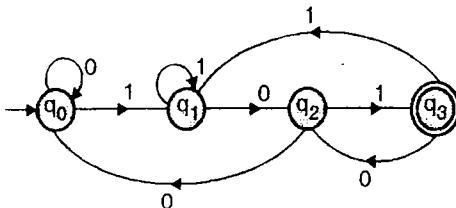
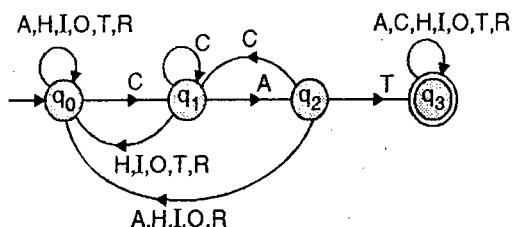


Fig. Ex. 2.2.23(c)

Example 2.2.24 SPPU - May 12, 8 Marks

Design a DFA that reads strings made up of letters in the word 'CHARIOT' and recognizes these strings that contain the word 'CAT' as a substring.

Solution :



(a) State transition diagram

| | 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 |

(b) State transition table

Fig. Ex. 2.2.24 : DFA for Example 2.2.24

Meaning of various states :

q_0 : Starting state.

q_1 : First character C of 'CAT' is the preceding character.

q_2 : First two characters CA of 'CAT' are the preceding two characters.

q_3 : entire 'CAT' has been seen.

Example 2.2.25

Design a FA that reads strings made of letter in the word 'UNIVERSITY' and recognize these strings that contains the word UNITY as substring.

Solution :

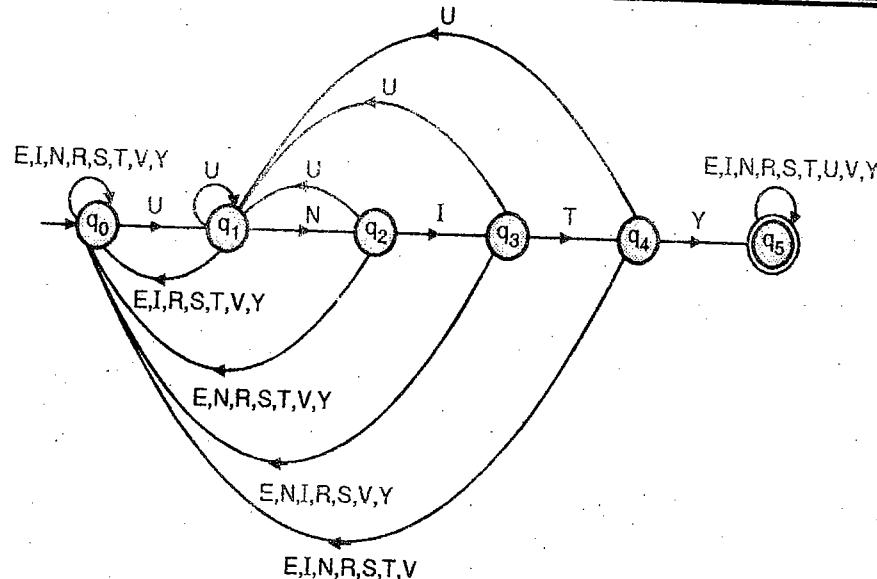


Fig. Ex. 2.2.25

Meaning of various states

q_0 : Starting state

q_1 : First character U of 'UNITY' is the preceding character.

q_2 : First two characters UN of 'UNITY' are the preceding two characters.

q_3 : First three characters UNI of 'UNITY' are the preceding three characters.

q_4 : First four characters UNIT of 'UNITY' are the preceding four characters.

q_5 : Entire 'UNITY' has been seen.

2.2.4.3 Examples of Divisibility

Example 2.2.26 SPPU – May 12, 8 Marks

Design a DFA which can accept a binary number divisible by 3.

OR

Design of a divisibility – by – 3 – tester for a binary number.

Solution :

A binary number is divisible by 3, if the remainder when divided by 3 will work out to be zero. We must devise a mechanism for finding the final remainder.

- We can calculate the running remainder based on previous remainder and the next input.

- The running remainder could be :

0 → associated state, q_0

1 → associated state, q_1

2 → associated state, q_2

- Starting with the most significant bit, input is taken one bit at a time. Running remainder is calculated after every input.

The process of finding the running remainder is being explained with the help of an example.

Number to be divided : 101101.

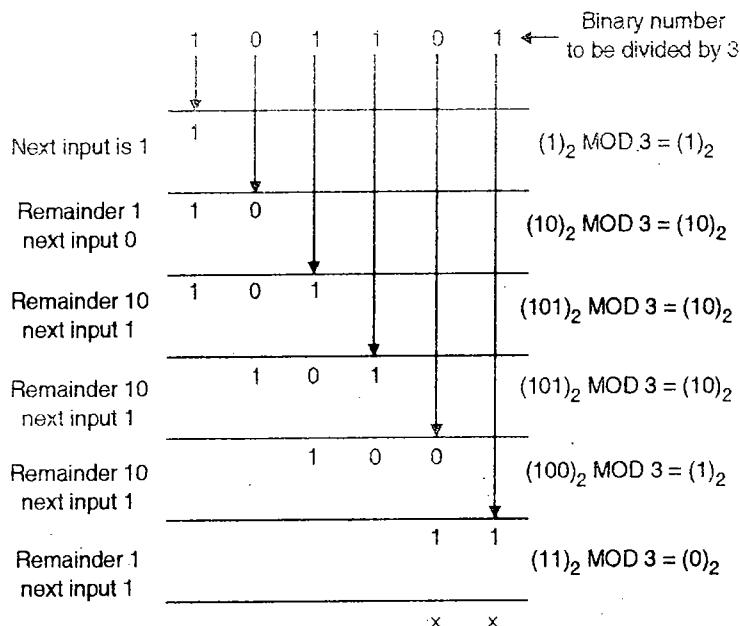


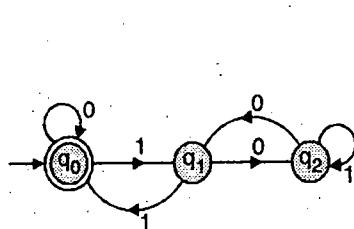
Fig. Ex. 2.2.26(a)

The calculation of next remainder is shown below,

| Previous remainder | Next input | Calculation of remainder | Next remainder |
|--------------------|------------|--------------------------|------------------------|
| $0 (q_0)$ | 0 | $00 \% 3$ | $\Rightarrow 0 (q_0)$ |
| $0 (q_0)$ | 1 | $01 \% 3$ | $\Rightarrow 1 (q_1)$ |
| $1 (q_1)$ | 0 | $10 \% 3$ | $\Rightarrow 10 (q_2)$ |
| $1 (q_1)$ | 1 | $11 \% 3$ | $\Rightarrow 0 (q_0)$ |
| $10 (q_2)$ | 0 | $100 \% 3$ | $\Rightarrow 1 (q_1)$ |
| $10 (q_2)$ | 1 | $101 \% 3$ | $\Rightarrow 10 (q_2)$ |

↑ ↑ ↑

Binary Binary decimal Binary



| | | |
|---------------------|-------|-------|
| | 0 | 1 |
| $\rightarrow q_0^*$ | q_0 | q_1 |
| q_1 | q_2 | q_0 |
| q_2 | q_1 | q_2 |

(b) State transition diagram

(c) State transition table

Fig. Ex. 2.2.26 : DFA for Example 2.2.26

Example 2.2.27

Design a DFA which can accept a ternary number divisible by 4.

Solution :

A ternary system has three alphabets.

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

Base of a ternary number is 3.

The running remainder could be :

- $(0)_3 = 0 \rightarrow$ associated state, q_0
 - $(1)_3 = 1 \rightarrow$ associated state, q_1
 - $(2)_3 = 2 \rightarrow$ associated state, q_2
 - $(10)_3 = 3 \rightarrow$ associated state, q_3
- ↑ ↑

Ternary Decimal

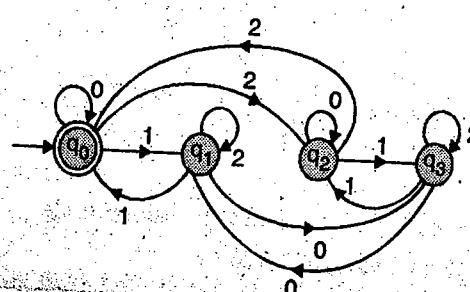
Transition behaviour will depend on the current remainder (current state) and the next input. These two will be used to get the next remainder (next state).

Transition behaviour is explained below :

| Current remainder | Next input | Calculation of remainder | Remainder |
|-------------------|------------|---|-----------|
| (q_0) | 0 | $00 \Rightarrow 0 \text{ MOD } 4 \Rightarrow 0 (q_0)$ | |
| | 1 | $01 \Rightarrow 1 \text{ MOD } 4 \Rightarrow 1 (q_1)$ | |
| | 2 | $02 \Rightarrow 2 \text{ MOD } 4 \Rightarrow 2 (q_2)$ | |
| (q_1) | 0 | $10 \Rightarrow 3 \text{ MOD } 4 \Rightarrow 3 (q_3)$ | |
| | 1 | $11 \Rightarrow 4 \text{ MOD } 4 \Rightarrow 0 (q_0)$ | |
| | 2 | $12 \Rightarrow 5 \text{ MOD } 4 \Rightarrow 1 (q_1)$ | |
| (q_2) | 0 | $20 \Rightarrow 6 \text{ MOD } 4 \Rightarrow 2 (q_2)$ | |
| | 1 | $21 \Rightarrow 7 \text{ MOD } 4 \Rightarrow 3 (q_3)$ | |
| | 2 | $22 \Rightarrow 8 \text{ MOD } 4 \Rightarrow 0 (q_0)$ | |
| (q_3) | 0 | $100 \Rightarrow 9 \text{ MOD } 4 \Rightarrow 1 (q_1)$ | |
| | 1 | $101 \Rightarrow 10 \text{ MOD } 4 \Rightarrow 2 (q_2)$ | |
| | 2 | $102 \Rightarrow 11 \text{ MOD } 4 \Rightarrow 3 (q_3)$ | |

↑ ↑ ↑

Ternary Decimal Decimal



(a) State transition diagram

| | 0 | 1 | 2 |
|---------------------|-------|-------|-------|
| $\rightarrow q_0^*$ | q_0 | q_1 | q_2 |
| q_1 | q_3 | q_0 | q_1 |
| q_2 | q_2 | q_3 | q_0 |
| q_3 | q_1 | q_2 | q_3 |

(b) State transition table

Fig. Ex. 2.2.27 : DFA for Example 2.2.27

Example 2.2.28

Design a DFA for a mod 5 tester for ternary input.

Solution : A ternary system has three alphabets

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



Base of a ternary number is 3.

The running remainder could be :

$$(0)_3 = 0 \rightarrow \text{associated state, } q_0$$

$$(1)_3 = 1 \rightarrow \text{associated state, } q_1$$

$$(2)_3 = 2 \rightarrow \text{associated state, } q_2$$

$$(10)_3 = 3 \rightarrow \text{associated state, } q_3$$

$$(11)_3 = 4 \rightarrow \text{associated state, } q_4$$

↑ ↑

Ternary Decimal

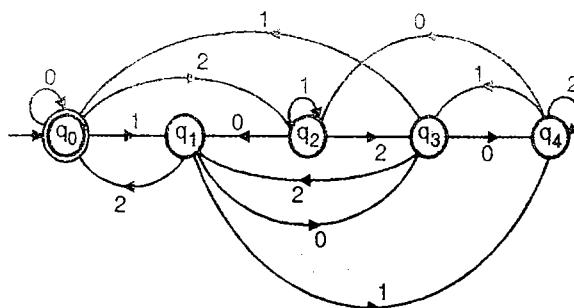


Fig. Ex. 2.2.28

Example 2.2.29 SPPU - Dec 13, 6 Marks

Design a DFA which can accept a decimal number divisible by 3.

Solution : A decimal system has 10 alphabets.

$$\Sigma = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$$

The running remainder could be

$$0 \rightarrow \text{associated state, } q_0$$

$$1 \rightarrow \text{associated state, } q_1$$

$$2 \rightarrow \text{associated state, } q_2$$

Transition behaviour will depend on the current remainder (current state) and the next input. These two will be used to get the next remainder (next state).

- Next digit as 0, 3, 6, 9 will be mapped to the same next state.
- Next digit as 1, 4, 7 will be mapped to the same next state
- Next digit as 2, 5, 8 will be mapped to the same next state.

Thus it is sufficient to consider transition for just three inputs :

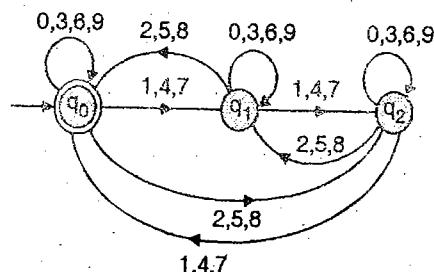
$$0 \text{ for } (0, 3, 6, 9)$$

$$1 \text{ for } (1, 4, 7)$$

$$2 \text{ for } (2, 5, 8)$$

Transition behaviour is explained below :

| Current remainder | Next input | Calculation of remainder | Remainder |
|-------------------|------------|------------------------------------|-----------------------|
| 0 | 0 | $00 \Rightarrow 0 \text{ MOD } 3$ | $\Rightarrow 0 (q_0)$ |
| | 1 | $01 \Rightarrow 1 \text{ MOD } 3$ | $\Rightarrow 1 (q_1)$ |
| | 2 | $02 \Rightarrow 2 \text{ MOD } 3$ | $\Rightarrow 2 (q_2)$ |
| 1 | 0 | $10 \Rightarrow 10 \text{ MOD } 3$ | $\Rightarrow 1 (q_1)$ |
| | 1 | $11 \Rightarrow 11 \text{ MOD } 3$ | $\Rightarrow 2 (q_2)$ |
| | 2 | $12 \Rightarrow 12 \text{ MOD } 3$ | $\Rightarrow 0 (q_0)$ |
| 2 | 0 | $20 \Rightarrow 20 \text{ MOD } 3$ | $\Rightarrow 2 (q_2)$ |
| | 1 | $21 \Rightarrow 21 \text{ MOD } 3$ | $\Rightarrow 0 (q_0)$ |
| | 2 | $22 \Rightarrow 22 \text{ MOD } 3$ | $\Rightarrow 1 (q_1)$ |



(a) State transition diagram

| | $(0, 3, 6, 9)$ | $(1, 4, 7)$ | $(2, 5, 8)$ |
|---------------------|----------------|-------------|-------------|
| $\rightarrow q_0^*$ | q_0 | q_1 | q_2 |
| q_1 | q_1 | q_2 | q_0 |
| q_2 | q_2 | q_0 | q_1 |

(b) State transition table

Fig. Ex. 2.2.29 : DFA for Example 2.2.29

Example 2.2.30

Design an FSM for divisibility by 3 tester for a unary number.

Solution :

Let us assume that $\Sigma = \{0\}$

For unary number to be divisible by 3, the number of 0's should be multiple of 3. The FSM is given in Fig. Ex. 2.2.30.

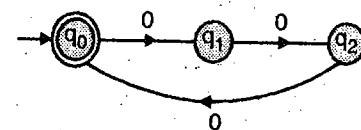


Fig. Ex. 2.2.30

Example 2.2.31

Design a Finite State Machine for divisibility by 5 tester of a given decimal number.

Solution :

A decimal number will be divisible by 5 if the rightmost digit is either '0' or '5'.

The required DFA is given in Fig. Ex. 2.2.31.

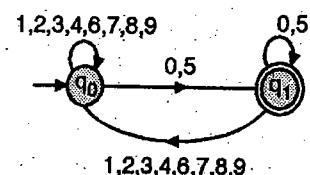


Fig. Ex. 2.2.31

2.2.5 Language of DFA

The language of a DFA $M = \{Q, \Sigma, \delta, q_0, F\}$ is denoted by $L(M)$ and is defined by :

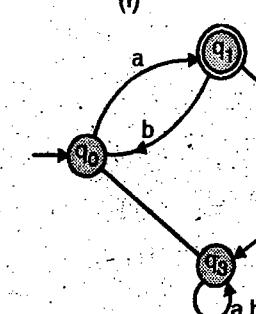
$$L(M) = \{\omega \mid \delta^*(q_0, \omega) \text{ is in } F\}$$

That is, the language of DFA M is the set of strings accepted by M . The language of a DFA is also known as regular language. $\delta^*(q_0, \omega)$ stands for a series of transitions starting from q_0 .

Example 2.2.32

Describe the language accepted by the deterministic finite automata shown in Fig. Ex. 2.2.32.

(i)



(ii)

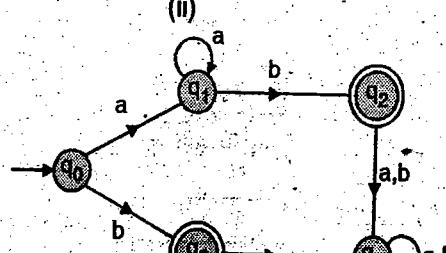


Fig. Ex. 2.2.32

Solution :

(i)

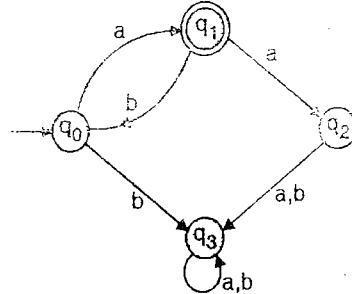


Fig. Ex. 2.2.32(a)

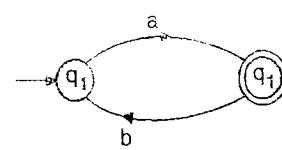


Fig. Ex. 2.2.32(b)

- o There is no path from q_2 to q_1 (final state) or from q_3 to q_1 . Thus q_2 to q_3 are dead states.
- o The DFA can be redrawn after elimination of q_2 and q_3 .
- o The language of the DFA can be defined as given below :

$$L = \{ \omega \in \{a, b\}^* \mid \omega \text{ starts and ends with } a \text{ and } a, b \text{ alternate} \}$$

(ii)

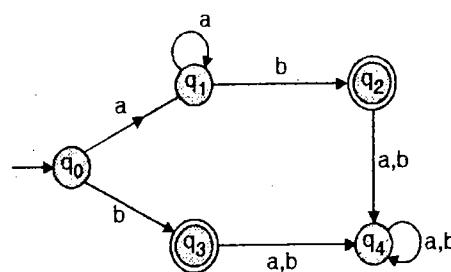


Fig. Ex. 2.2.32(c)

- o q_4 is dead state. The DFA can be redrawn after elimination of q_4 .

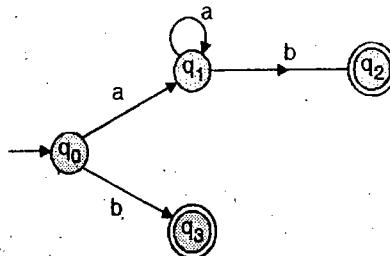


Fig. Ex. 2.2.32(d)

- o A string 'b' of length 1 is accepted through the path $q_0 \rightarrow q_3$.
- o A string having 1 or more a's followed by a 'b' is accepted through the path.
- o $q_0 \rightarrow q_1 \rightarrow q_1 \dots q_1 \rightarrow q_2$

The language of the DFA can be described as given below :

$$L = \{ \omega \in \{a, b\}^* \mid \omega \text{ consists of 0 or more } a's \text{ followed by a } 'b' \}$$

Example 2.2.33

Consider the following transition diagram in Fig. Ex. 2.2.33.

Test whether the string 110101 is accepted by the finite automata represented by above transition diagram. Show the entire sequence of states traversed.

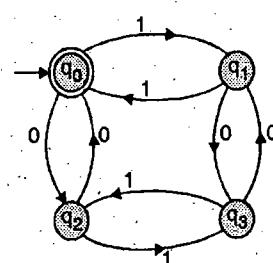


Fig. Ex. 2.2.33



Solution :

$$\begin{aligned}
 (q_0, 110101) &\xrightarrow{\delta} (q_1, 10101) \\
 &\xrightarrow{\delta} (q_0, 0101) \\
 &\xrightarrow{\delta} (q_2, 101) \\
 &\xrightarrow{\delta} (q_3, 01) \\
 &\xrightarrow{\delta} (q_1, 1) \\
 &\xrightarrow{\delta} (q_0, \epsilon) \Rightarrow q_0
 \end{aligned}$$

String 110101 will be accepted as q_0 is a final state.

Syllabus Topic : Equivalence of DFA

2.2.6 Equivalence of DFAs

SPPU - May 15

University Questions

- Q.** Prove that the following decision problems are recursive:
- Two DFA's are equivalent or not.
 - DFA accepts a word or not.

(May 2015, 5 Marks)

Two finite automata M_1 and M_2 are said to be equivalent if they accept the same language i.e., $L(M_1) = L(M_2)$

Two finite automata M_1 and M_2 are not equivalent over Σ , if there exists a ω such that :

$$\omega \in L(M_1) \text{ and } \omega \notin L(M_2)$$

or

$$\omega \notin L(M_1) \text{ and } \omega \in L(M_2)$$

That is, one DFA reaches a final state on application of ω and other reaches a non final state.

Thus, if two DFAs are equivalent, then for every $\omega \in \Sigma^*$:

On application of ω , either both M_1 and M_2 will be in a final state or both M_1 and M_2 will be in a non final state, simultaneously.

Testing equivalence of two DFAs

Equivalence of two DFA's can be established by constructing a combined DFA for two DFAs M_1 and M_2 . Let the combined DFA for M_1 and M_2 be M_3 .

where, $M_1 = (S, \Sigma, \delta_1, s_0, F_1)$

and $M_2 = (Q, \Sigma, \delta_2, q_0, F_2)$ The combined machine M_3 can be described in terms of M_1 and M_2 .

$$M_3 = (S \times Q, \Sigma, \delta_3, < s_0, q_0 >, F_3)$$

A state of M_3 is of the form $< s_i, q_i >$

Where,

$$s_i \in S \text{ of } M_1 \text{ and } q_i \in Q \text{ of } M_2.$$

The transition function δ_3 is defined as :

$$\delta_3(< s_i, q_i >, a) = < \delta_1(s_i, a), \delta_2(q_i, a) >$$

δ_3 is a function from $S \times Q$ to $S \times Q$.



The algorithm for generation of transition behaviour δ_3 of M_3 is given below :

1. Add the state $\langle s_i, q_0 \rangle$ to M_3 .
2. for every state $\langle s_i, q_i \rangle$ in M_3 .
 - { if ($\langle s_i, q_i \rangle$ has not been expanded)
 - { for each alphabet $a_i \in \Sigma$
 - { add a transition $\langle \delta_1(s_i, a_i), \delta_2(q_i, a_i) \rangle$ to M_3

M_1 and M_2 are equivalent if :

- for every state $\langle s_i, q_i \rangle$ in M_3
 - { both s_i and q_i are final states
 - or
 - both s_i and q_i are non-final states

M_1 and M_2 are not equivalent if :

- for any state $\langle s_i, q_i \rangle$ in M_3
 - { s_i is a final state and q_i is a non-final state
 - or
 - s_i is a non-final state and q_i is a final state

Example 2.2.34

Show whether the automata M_1 and M_2 in given Fig. Ex. 2.2.31 (a) and (b) are equivalent or not.

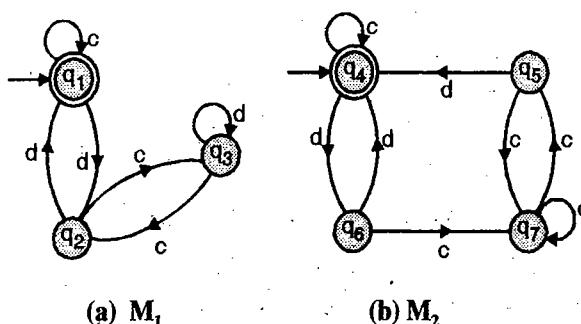


Fig. Ex. 2.2.34

Solution : Construction of combined DFA M_3

Step 1 : We start with the combined state $\langle q_1, q_4 \rangle$, where q_1 is start state of M_1 and q_4 is the start state of M_2 . $\langle q_1, q_4 \rangle$ is expanded and necessary transitions added.

$$\delta_3(\langle q_1, q_4 \rangle, c) \Rightarrow \langle \delta_1(q_1, c), \delta_2(q_4, c) \rangle$$

$$\Rightarrow \langle q_1, q_4 \rangle$$

$$\delta_3(\langle q_1, q_4 \rangle, d) \Rightarrow \langle \delta_1(q_1, d), \delta_2(q_4, d) \rangle$$

$$\Rightarrow \langle q_2, q_6 \rangle$$

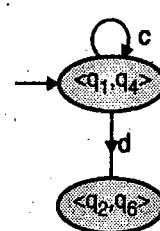


Fig. Ex. 2.2.34(c)

Step 2 : State $\langle q_2, q_6 \rangle$ is selected for expansion.

$$\begin{aligned}\delta_3 (\langle q_2, q_6 \rangle, c) &\Rightarrow \langle \delta_1 (q_2, c), \delta_2 (q_6, c) \rangle \\ &\Rightarrow \langle q_3, q_7 \rangle \\ \delta_3 (\langle q_2, q_6 \rangle, d) &\Rightarrow \langle \delta_1 (q_2, d), \delta_2 (q_6, d) \rangle \\ &\Rightarrow \langle q_1, q_4 \rangle\end{aligned}$$

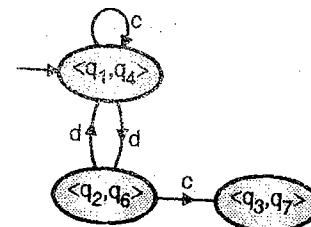
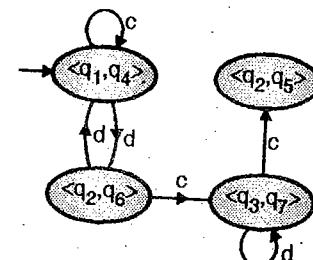


Fig. Ex. 2.2.34(d)

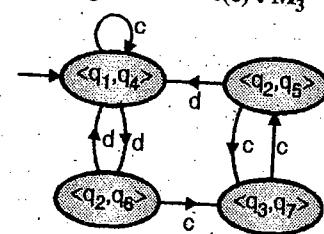
Step 3 : State $\langle q_3, q_7 \rangle$ is selected for expansion.

$$\begin{aligned}\delta_3 (\langle q_3, q_7 \rangle, c) &\Rightarrow \langle \delta_1 (q_3, c), \delta_2 (q_7, c) \rangle \\ &\Rightarrow \langle q_2, q_5 \rangle \\ \delta_3 (\langle q_3, q_7 \rangle, d) &\Rightarrow \langle \delta_1 (q_3, d), \delta_2 (q_7, d) \rangle \\ &\Rightarrow \langle q_3, q_7 \rangle\end{aligned}$$

Fig. Ex. 2.2.34(e) : M_3

Step 4 : State $\langle q_2, q_5 \rangle$ is selected for expansion.

$$\begin{aligned}\delta_3 (\langle q_2, q_5 \rangle, c) &\Rightarrow \langle \delta_1 (q_2, c), \delta_2 (q_5, c) \rangle \\ &\Rightarrow \langle q_3, q_7 \rangle \\ \delta_3 (\langle q_2, q_5 \rangle, d) &\Rightarrow \langle \delta_1 (q_2, d), \delta_2 (q_5, d) \rangle \\ &\Rightarrow \langle q_1, q_4 \rangle\end{aligned}$$

Fig. Ex. 2.2.34(f) : M_3

Every state in M_3 has been expanded. The construction of combine DFA is over. During the construction of M_3 , four states have been generated.

$\langle q_1, q_4 \rangle$: both q_1 and q_4 are final states.

$\langle q_2, q_5 \rangle$: q_2 and q_5 are non final states.

$\langle q_2, q_6 \rangle$: both are non final states.

$\langle q_3, q_7 \rangle$: q_3 and q_7 are non final states.

Thus, we can infer that M_1 and M_2 are equivalent.

Example 2.2.35

Find out whether M_1 and M_2 are equivalent.

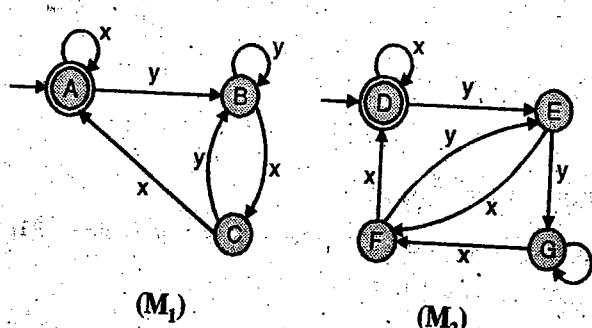


Fig. Ex. 2.2.35

Solution :

Two DFAs are equivalent if for any string $\omega \in \Sigma^*$, either the machines will be in a final state or both the machines will be in a non-final state.

This can be established by constructing a combined DFA for M_1 and M_2 .



Construction of combined DFA

In the Fig. Ex. 2.2.35(a), following subsets are generated.

1. {A, D} – Both A and D are accepting states.
2. {D, E} – Both D and E are non-accepting states.
3. {C, F} – Both C and F are non-accepting states.
4. {B, G} – Both B and G are non-accepting states.

Hence, machines M_1 and M_2 are equivalent.

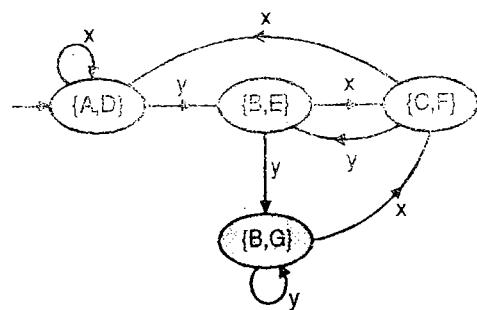


Fig. Ex. 2.2.35(a)

2.2.7 Closure Property of Language Accepted by a DFA

The class of languages accepted by a deterministic finite automata is closed under :

- (a) Union (b) Intersection
 (c) Difference (d) Complementation.

The closure property can be established by constructing an appropriate finite state automata in each case.

2.2.7.1 Union, Intersection, Difference

Let $M_1 = (S, \Sigma, \delta_1, s_0, F_1)$ and $M_2 = (Q, \Sigma, \delta_2, q_0, F_2)$

be two arbitrary DFAs. To prove the various closure properties, we must show that there is another machine :

M_3 such that $L(M_3) = L(M_1) \cup L(M_2)$

M_4 such that $L(M_4) = L(M_1) \cap L(M_2)$

M_5 such that $L(M_5) = L(M_1) - L(M_2)$

We will first construct a combined machine M_c , simulating the transition behaviour of both. M_1 and M_2 . Then, we will derive M_3 , M_4 or M_5 from M_c .

The combine machine M_c can be described in terms of M_1 and M_2 .

$$M_c = (S \times Q, \Sigma, \delta_c, <s_0, q_0>, F_c)$$

A state of M_c is of the form $<s_i, q_j>$ where,

$s_i \in S$ of M_1 .

and $q_j \in Q$ of M_2 .

The transition function δ_c is defined as :

$$\delta_c(<s_i, q_j>, a) = <\delta_1(s_i, a), \delta_2(q_j, a)>$$

δ_c is a function from $S \times Q$ to $S \times Q$.

The **algorithm** for generation of transition behaviour δ_c of M_c is given below :

1. Add the state $<s_0, q_0>$ to M_c .

2. for every state $<s_i, q_j>$ in M_c ,

{

if($<s_i, q_j>$ has not been expanded)

{

for each alphabet $a_i \in \Sigma$

{

add a transition $<\delta_1(s_i, a_i), \delta_2(q_j, a_i)>$ to M_c .

}

}

}

Deriving M_3 from M_c ($L(M_3) = L(M_1) \cup L(M_2)$)

M_3 can be derived from M_c by selecting a set of states from M_c as a set of final states for M_3 .

Set of final states F_3 for $M_3 = \{s_i, q_i \in \text{states of } M_c \mid s_i \in F_1 \vee q_i \in F_2\}$

where F_1 is set of final states for M_1 and F_2 is set of final states for M_2 .

Deriving M_4 from M_c ($L(M_4) = L(M_1) \cap L(M_2)$)

M_4 can be derived from M_c by selecting a set of states from M_c as a set of final states for M_4 .

Set of final states F_4 for $M_4 = \{s_i, q_i \in \text{states of } M_c \mid s_i \in F_1 \wedge q_i \in F_2\}$

where, F_1 is set of final states for M_1 and F_2 , is set of final states for M_2 .

Deriving M_5 from M_c ($L(M_5) = L(M_1) - L(M_2)$)

M_5 can be derived from M_c by selecting a set of states from M_c as a set of final states for M_5 .

Set of final states F_5 from $M_5 = \{s_i, q_i \in \text{states of } M_c \mid s_i \in F_1 \cap q_i \notin F_2\}$

where, F_1 is a set of final states for M_1 and F_2 is a set of final states for M_2 .

Example 2.2.36

Suppose, we have two machines M_1 and M_2 .

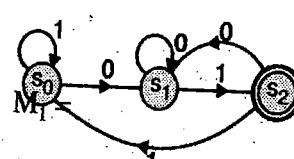
M_1 : For accepting a set of strings over alphabet {0, 1} ending in 01.

M_2 : For accepting a set of strings over alphabet {0, 1} containing even number of 1's.

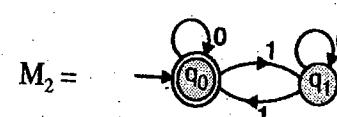
Design a DFA for the following :

- (i) $L(M_1) \cup L(M_2)$ (ii) $L(M_2) \cap L(M_1)$ (iii) $L(M_1) - L(M_2)$

Solution :



(a) DFA for strings ending in 01



(b) DFA for strings having even number of 1's.

Fig. Ex. 2.2.36

Construction of combined DFA M_c

Step 1 : We start with the state $\langle s_0, q_0 \rangle$, where s_0 is the initial state of M_1 and q_0 is the initial state of M_2 . $\langle s_0, q_0 \rangle$ is expanded and necessary transitions added.

$$\delta(\langle s_0, q_0 \rangle, 0) \Rightarrow \langle \delta(s_0, 0), \delta(q_0, 0) \rangle$$

$$\Rightarrow \langle s_1, q_0 \rangle$$

$$\delta(\langle s_0, q_0 \rangle, 1) \Rightarrow \langle \delta(s_0, 1), \delta(q_0, 1) \rangle$$

$$\Rightarrow \langle s_0, q_1 \rangle$$

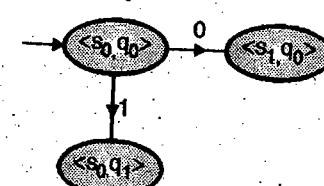


Fig. Ex. 2.2.36(c)

Step 2 : State $\langle s_1, q_0 \rangle$ and $\langle s_0, q_1 \rangle$ are selected for expansion.

$$\delta(\langle s_0, q_1 \rangle, 0) \Rightarrow \langle \delta(s_0, 0), \delta(q_1, 0) \rangle$$

$$\Rightarrow \langle s_1, q_1 \rangle$$

$$\delta(\langle s_0, q_1 \rangle, 1) \Rightarrow \langle \delta(s_0, 1), \delta(q_1, 1) \rangle$$

$$\Rightarrow \langle s_0, q_0 \rangle$$

$$\delta(\langle s_1, q_0 \rangle, 0) \Rightarrow \langle \delta(s_1, 0), \delta(q_0, 0) \rangle$$

$$\Rightarrow \langle s_1, q_0 \rangle$$

$$\delta(\langle s_1, q_0 \rangle, 1) \Rightarrow \langle \delta(s_1, 1), \delta(q_0, 1) \rangle$$

$$\Rightarrow \langle s_2, q_1 \rangle$$

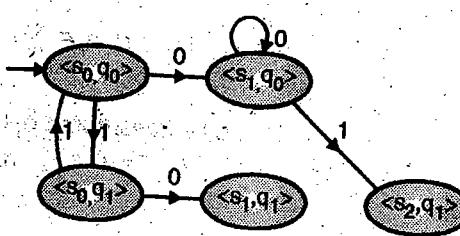


Fig. Ex. 2.2.36(d)

Step 3 : State $\langle s_1, q_1 \rangle$ and $\langle s_2, q_1 \rangle$ are selected for expansion.

$$\begin{aligned} \delta(\langle s_1, q_1 \rangle, 0) &\Rightarrow \langle \delta(s_1, 0), \delta(q_1, 0) \rangle \\ &\Rightarrow \langle s_1, q_1 \rangle \end{aligned}$$

$$\begin{aligned} \delta(\langle s_1, q_1 \rangle, 1) &\Rightarrow \langle \delta(s_1, 1), \delta(q_1, 1) \rangle \\ &\Rightarrow \langle s_2, q_0 \rangle \end{aligned}$$

$$\begin{aligned} \delta(\langle s_2, q_1 \rangle, 0) &\Rightarrow \langle \delta(s_2, 0), \delta(q_1, 1) \rangle \\ &\Rightarrow \langle s_1, q_1 \rangle \end{aligned}$$

$$\begin{aligned} \delta(\langle s_2, q_1 \rangle, 1) &\Rightarrow \langle \delta(s_2, 1), \delta(q_1, 1) \rangle \\ &\Rightarrow \langle s_0, q_0 \rangle \end{aligned}$$

Step 4 : State $\langle s_2, q_0 \rangle$ is selected for expansion.

$$\begin{aligned} \delta(\langle s_2, q_0 \rangle, 0) &\Rightarrow \langle \delta(s_2, 0), \delta(q_0, 1) \rangle \\ &\Rightarrow \langle s_1, q_0 \rangle \end{aligned}$$

$$\begin{aligned} \delta(\langle s_2, q_0 \rangle, 1) &\Rightarrow \langle \delta(s_2, 1), \delta(q_0, 1) \rangle \\ &\Rightarrow \langle s_0, q_1 \rangle \end{aligned}$$

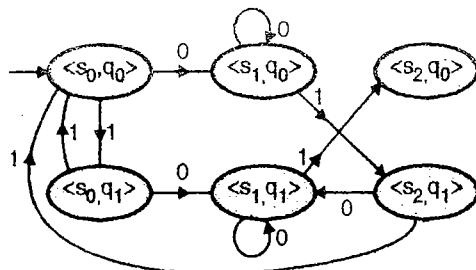


Fig. Ex. 2.2.36(e)

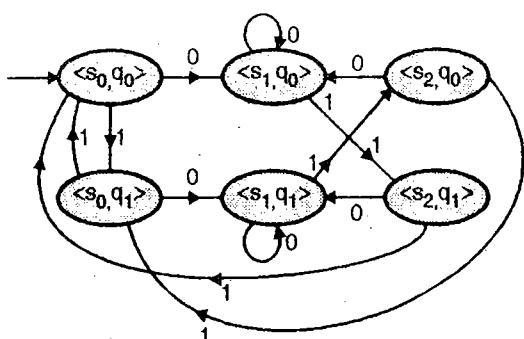


Fig. Ex. 2.2.36(f)

Finding a DFA for $L(M_1) \cup L(M_2)$

The set of final states of the required DFA is given by :

Every $\langle s_i, q_i \rangle$ such that

$s_i \in$ final states of M_1 or $q_i \in$ final states of M_2 .

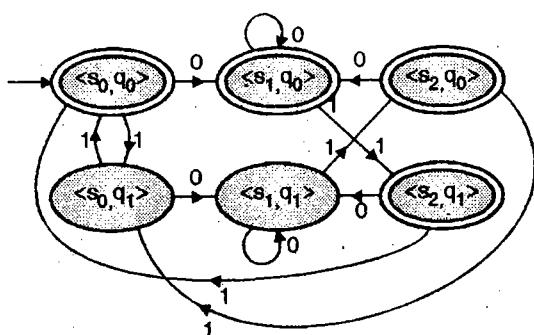
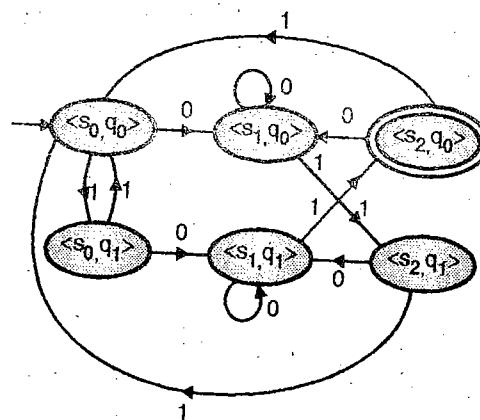


Fig. Ex. 2.2.36(g) : DFA for $L(M_1) \cup L(M_2)$

Finding DFA for $L(M_1) \cap L(M_2)$

The set of final states of the required DFA is given by :

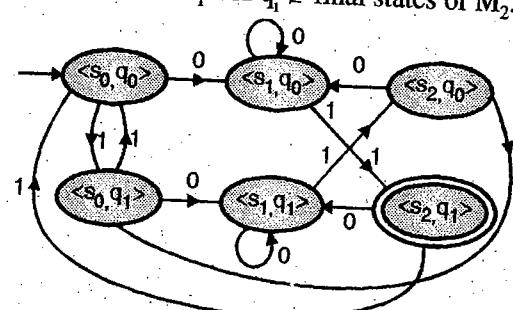
Every $\langle s_i, q_i \rangle$ such that $s_i \in$ final states of M_1 and $q_i \in$ final states of M_2 .

Fig. Ex. 2.2.36(h) : DFA for $L(M_1) \cap L(M_2)$

Finding DFA for $L(M_1) - L(M_2)$

The set of final states of the required DFA is given by :

every $\langle s_i, q_i \rangle$ such that $s_i \in$ final states of M_1 and $q_i \notin$ final states of M_2 .

Fig. Ex. 2.2.36(i) : DFA for $L(M_1) - L(M_2)$

2.2.7.2 Complementation

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be an arbitrary DFA. To prove the closure property complementation, we must show that there is another machine M_2 such that,

$$M_2(L) = (M_1(L))'$$

M_2 can be constructed from of M_1 by making the following changes to M_1 .

1. Make every final state M_1 of as a non-final state.
2. Make every non-final state of M_1 as a final state.

Machine M_2 can be defined as :

$$(Q, \Sigma, \delta, q_0, Q - F)$$

Syllabus Topic : Minimization of FA

2.2.8 Minimization of DFA

It may happen that a DFA contains redundant states. i.e. states whose functions can be handled by other states. Minimization of a DFA is a process of constructing an equivalent DFA with minimum number of states.

- It may be possible that for a given DFA some of the states might be equivalent or they are not distinguishable.
- Equivalent states from an equivalent class or group. Every state from an equivalent class exhibit the same transition behaviour.
- States of a DFA can be divided into a group of equivalent classes.

Equivalent states

Two states q_i and $q_j \in Q$ are said to be equivalent if they are not distinguishable.

- (1) If q_i is an accepting state and q_j is non-accepting, then q_i and q_j are distinguishable.
- (2) If q_i is non-accepting and q_j is accepting, then q_i and q_j are distinguishable.
- (3) If there is string such that.

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

or

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

then q_i and q_j are distinguishable.

K-equivalence : q_i and q_j are K-equivalent states if and only if no string of length $\leq k$ can distinguish them. If there exists a string of length k , which can distinguish q_i and q_j , the states q_i and q_j are said to be K-distinguishable.

2.2.8.1 Algorithm for Minimization DFA's

Algorithm for minimization divides states of a DFA into blocks such that :

1. States in a block are equivalent.
2. No two states taken from two different blocks are equivalent.

The first step is to partition the states of M into blocks such that all states in a block are 0-equivalent. This is accomplished by placing :

1. Final states into one block.
2. Non-final states into another block.

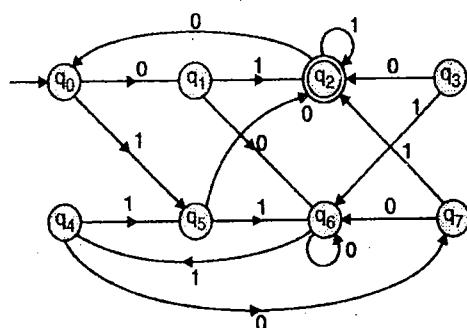


Fig. 2.2.8 : DFA considered for minimization

States of the DFA in Fig. 2.2.8 can be divided into two blocks.

$$\begin{aligned} \text{block 1} &= \text{set of non-final states} \\ &= (q_0, q_1, q_3, q_4, q_5, q_6, q_7) \end{aligned}$$

$$\begin{aligned} \text{block 2} &= \text{set of final states.} \\ &= (q_2) \end{aligned}$$

\therefore 0-equivalence partitioning of Q ,

$$P_0 = (q_0, q_1, q_3, q_4, q_5, q_6, q_7) (q_2)$$

- The next step is to obtain the partition P_1 whose blocks consists of the set of states which are 1-equivalent, i.e. equivalent under any input sequence of length 1.

Checking $(q_0, q_1, q_3, q_4, q_5, q_6, q_7)$ for 1-equivalence

Input '0'

$$\delta(q_0, 0) \Rightarrow q_1 \text{ (block 1, } q_1 \text{ is in block 1)}$$

$$\delta(q_1, 0) \Rightarrow q_6 \text{ (block 1, } q_6 \text{ is in block 1)}$$

$$\delta(q_3, 0) \Rightarrow q_2 \text{ (block 2, } q_2 \text{ is in block 2)}$$

$$\delta(q_4, 0) \Rightarrow q_7 \text{ (block 1, } q_7 \text{ is in block 1)}$$

$$\delta(q_5, 0) \Rightarrow q_2 \text{ (block 2, } q_2 \text{ is in block 2)}$$

$$\delta(q_6, 0) \Rightarrow q_6 \{ \text{block 1, } q_6 \text{ is in block 1} \}$$

$$\delta(q_7, 0) \Rightarrow q_6 \{ \text{block 1, } q_6 \text{ is in block 1} \}$$

On input 0, block 1, is successor of q_0, q_1, q_4, q_6, q_7 .

On input 0, block 2, is successor of q_3, q_5 .

Input '1'

$$\delta(q_0, 1) \Rightarrow q_5 \{ \text{block 1} \}$$

$$\delta(q_1, 1) \Rightarrow q_2 \{ \text{block 2} \}$$

$$\delta(q_3, 1) \Rightarrow q_6 \{ \text{block 1} \}$$

$$\delta(q_4, 1) \Rightarrow q_5 \{ \text{block 1} \}$$

$$\delta(q_5, 1) \Rightarrow q_6 \{ \text{block 1} \}$$

$$\delta(q_6, 1) \Rightarrow q_4 \{ \text{block 1} \}$$

$$\delta(q_7, 1) \Rightarrow q_2 \{ \text{block 2} \}$$

On input 1, block 1 is successor of q_0, q_3, q_4, q_5, q_6 .

On input 1, block 2 is successor of q_1, q_7 .

q_3, q_5 are distinguishable from, q_0, q_1, q_4, q_6, q_7 on input 0.

q_1, q_7 are distinguishable from, q_0, q_3, q_4, q_5, q_6 on input 1.

\therefore 1-equivalence partitioning of Q,

$$\begin{array}{cccc} P_1 = (q_3, q_5) & (q_1, q_7) & (q_0, q_4, q_6) & (q_2) \\ \uparrow & \uparrow & \uparrow & \uparrow \\ \text{block 11} & \text{block 12} & \text{block 13} & \text{block 2} \end{array}$$

The next step is to obtain the partition P_2 whose blocks consists of the sets of states which are 2-equivalent, i.e. equivalent under any input sequence of length 2.

Checking (q_3, q_5) for equivalence – block 11

$$\left. \begin{array}{l} \delta(q_3, 0) = q_2 \\ \delta(q_5, 0) = q_2 \end{array} \right\} \text{mapped to same block (q}_2\text{)}$$

$$\left. \begin{array}{l} \delta(q_3, 1) = q_6 \\ \delta(q_5, 1) = q_6 \end{array} \right\} \text{mapped to same block (q}_0, q_4, q_6\text{)}$$

q_3 and q_5 are 2-equivalent.

Checking (q_1, q_7) for equivalence – block 12

$$\left. \begin{array}{l} \delta(q_1, 0) = q_6 \\ \delta(q_7, 0) = q_6 \end{array} \right\} \text{mapped to same block (q}_0, q_4, q_6\text{)}$$

$$\left. \begin{array}{l} \delta(q_1, 1) = q_2 \\ \delta(q_7, 1) = q_2 \end{array} \right\} \text{mapped to same block (q}_2\text{)}$$

q_1 and q_7 are 2-equivalent.

Checking (q_0, q_4, q_6) for equivalence - block 13

$$\delta(q_0, 0) = q_1 \text{ [block 12]}$$

$$\delta(q_4, 0) = q_7 \text{ [block 12]}$$

$$\delta(q_6, 0) = q_6 \text{ [block 13]}$$

$$\delta(q_0, 1) = q_5 \text{ [block 11]}$$

$$\delta(q_4, 1) = q_5 \text{ [block 11]}$$

$$\delta(q_6, 1) = q_4 \text{ [block 13]}$$

q_6 is distinguishable from q_0, q_4 on input 0/1.

\therefore 2-equivalence partitioning of Q_1 .

$$P_2 = (q_3, q_5) (q_1, q_7) (q_0, q_4) (q_6) (q_2)$$

- The next step is to obtain the partition P_3 whose blocks consists of the sets of states which are 3-equivalent, i.e. equivalent under any input sequence of length 3.

It can be seen easily that P_2 can not be portioned further.

$$\therefore P_3 = P_2 (q_3, q_5), (q_1, q_7) (q_0, q_4) (q_6) (q_2)$$

- If for some k , $P_k = P_{k+1}$, the process terminates and P_k defines the blocks of equivalent states of the machine.
- The transition diagram for minimum state DFA for DFA of Fig. 2.2.8 is given in Fig. 2.2.9. The process of drawing of minimum-state DFA is as below :

1. Start state is $\langle q_0, q_4 \rangle$, since q_0 was start state in Fig. 2.2.8.
2. Only accepting state is $\langle q_2 \rangle$, since q_2 is the only accepting state of Fig. 2.2.8.
3. Since, there is a transition from q_0 to q_1 on input 0 in Fig. 2.2.8, we add a transition from $\langle q_0, q_4 \rangle$ to $\langle q_1, q_7 \rangle$ on input 0 in Fig. 2.2.9.
4. Since, there is a transition from q_0 to q_5 on input 1 in Fig. 2.2.8, we add a transition from $\langle q_0, q_4 \rangle$ to $\langle q_3, q_5 \rangle$ on input 1 in Fig. 2.2.9.
5. Other transitions can be added in a similar fashion.

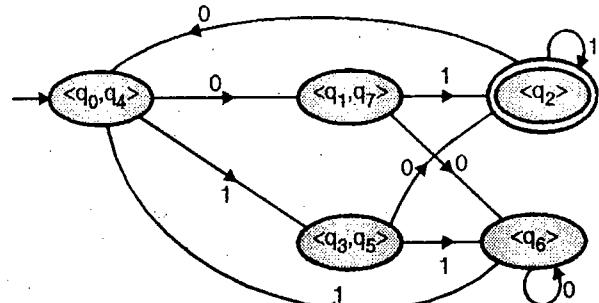


Fig. 2.2.9 : Minimum state DFA equivalent to Fig. 2.2.8

Example 2.2.37 SPPU - Dec.14. 4 Marks

Construct the minimum state automata equivalent to given DFA.

| | 0 | 1 |
|-------------------|-------|-------|
| $\rightarrow q_0$ | 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 |

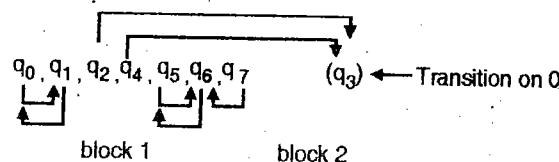
Solution :

Step 1 : Finding 0-equivalence partitioning of states by putting final and non-final states into independent block.

$$P_0 = (q_0, q_1, q_2, q_4, q_5, q_6, q_7) \quad (q_3)$$

block 1 block 2

Step 2 : Finding 1-equivalence partitioning of states by considering transition on '0' and transition on '1'.



On input 0, block 1 is successor of q_0, q_1, q_5, q_6, q_7 .

On input 0, block 2 is successor of q_2, q_4 .

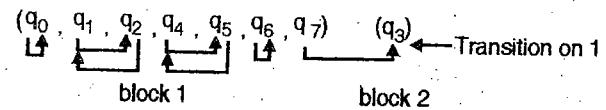
$\therefore q_2, q_4$ are distinguishable from q_0, q_1, q_5, q_6, q_7

On input 1, block 2 is successor of q_7 .

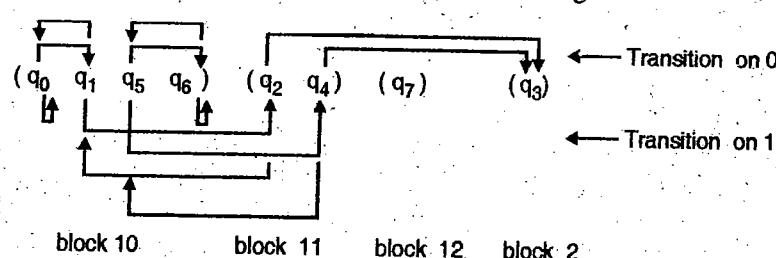
On input 1, block 1 is successor of $q_0, q_1, q_2, q_4, q_5, q_6$.

q_7 is distinguishable from $q_0, q_1, q_2, q_4, q_5, q_6$.

$$P_1 = (q_0, q_1, q_5, q_6) (q_2, q_4) (q_7) (q_3)$$



Step 3 : Finding 2-equivalence partitioning of states by considering transition on '0' and transition on '1'.



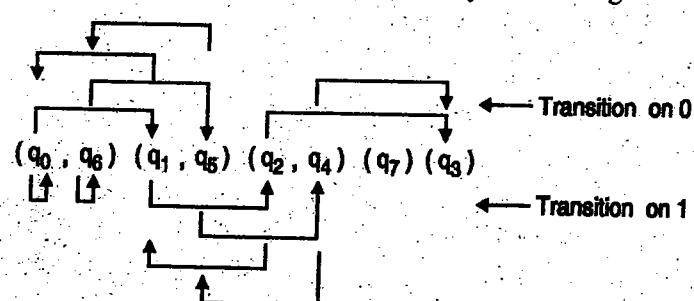
On input 1, block 11 is successor of q_1, q_5 .

On input 1, block 10 is successor of q_0, q_6 .

q_1, q_5 is distinguishable from q_0, q_6 .

$$P_2 = (q_0, q_6) (q_1, q_5) (q_2, q_4) (q_7) (q_3)$$

Step 4 : Finding 3-equivalence partitioning of states by considering transition on 0 and 1.



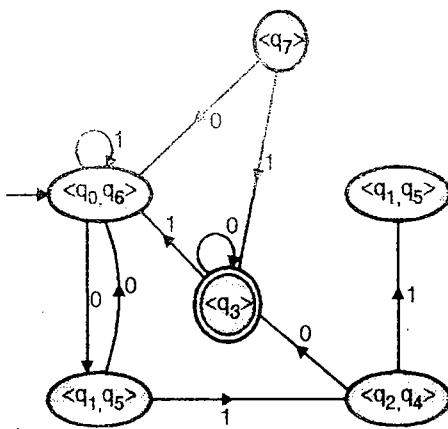
Blocks can not be divided further.

$\therefore P_3 = P_2 = (q_0, q_6) (q_1, q_5) (q_2, q_4) (q_7) (q_3)$ which is final set of blocks of equivalent classes.

Step 5 : Construction of minimum state DFA.

| | 0 | 1 |
|-------------------------|--------------|--------------|
| $\rightarrow(q_0, q_6)$ | (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) |

(a) State transition diagram for minimum state DFA



(b) State transition diagram for minimum state DFA

Fig. Ex. 2.2.37

Example 2.2.38

Construct the minimum-state DFA equivalent to given DFA.

| | 0 | 1 |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_1 | q_4 |
| q_1 | q_2 | q_5 |
| q_2^* | q_3 | q_7 |
| q_3 | q_4 | q_7 |
| q_4 | q_5 | q_8 |
| q_5^* | q_6 | q_1 |
| q_6 | q_7 | q_1 |
| q_7 | q_8 | q_2 |
| q_8^* | q_0 | q_4 |

Solution :

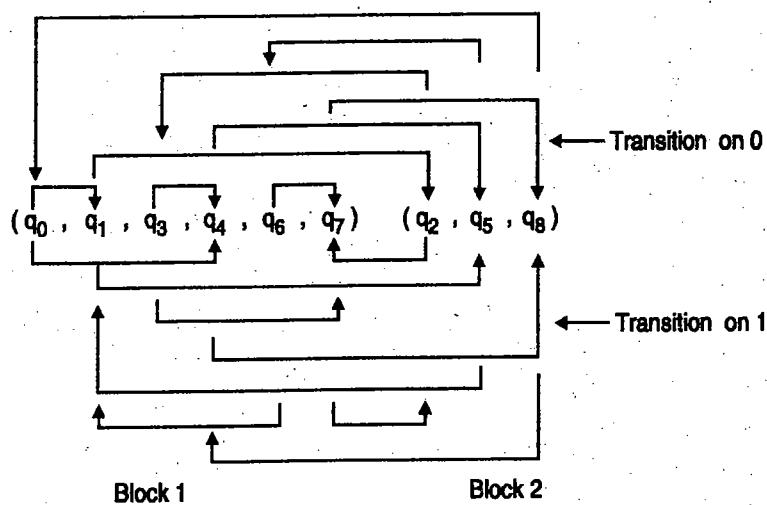
Step 1 : Finding 0-equivalence partitioning of states by putting final and non-final states into independent blocks.

$$P_0 = (q_0, q_1, q_3, q_4, q_6, q_7) \quad - (q_2, q_5, q_8)$$

block 1

block 2

Step 2 : Finding 1-equivalence partitioning of states by considering transitions on '0' on '1'.



On input 0, block 1 is successor of q_0, q_3, q_6

On input 0, block 2 is successor of q_1, q_4, q_7 , block 1

On input 1, block 1 is successor of q_0, q_3, q_6

On input 1, block 2 is successor of q_1, q_4, q_7

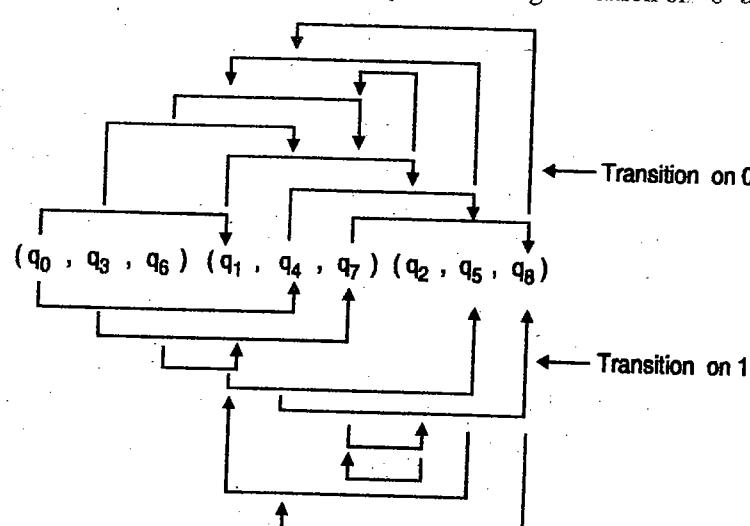
States q_0, q_3, q_6 are distinguishable from q_1, q_4, q_7 .

Block 2 is 1-equivalent

$$P_1 = (q_0, q_3, q_6) \quad (q_1, q_4, q_7) \quad (q_2, q_5, q_8)$$

block 11 block 12 block 2

Step 3 : Finding 2-equivalence partitioning of states by considering transition on '0' and transition on '1'.



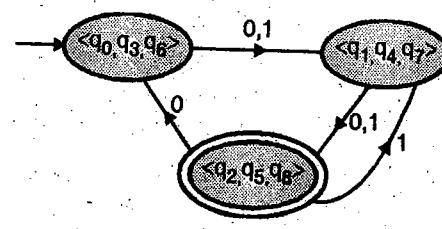
Blocks can not be divided further.

$\therefore P_2 = P_1 = (q_0, q_3, q_6) (q_1, q_4, q_7) (q_2, q_5, q_8)$ which is final set of blocks of equivalent classes.

Step 4 : Construction of minimum state DFA.

| | 0 | 1 |
|------------------------------|-------------------|-------------------|
| $\rightarrow(q_0, q_3, q_6)$ | (q_1, q_4, q_7) | (q_1, q_4, q_7) |
| (q_1, q_4, q_7) | (q_2, q_5, q_8) | (q_2, q_5, q_8) |
| $(q_2, q_5, q_8)^*$ | (q_0, q_3, q_6) | (q_1, q_4, q_7) |

(a) State transition table
for minimum state DFA



(b) State transition diagram
for minimum state DFA

Fig. Ex. 2.2.38 : Final DFA for Example 2.2.38

Syllabus Topic : NFA

2.3 Non-deterministic Finite Automata

Non-deterministic finite automata can reside in multiple states at the same time. The concept of non-deterministic finite automata is being explained with the help of transition diagram (Fig. 2.3.1).

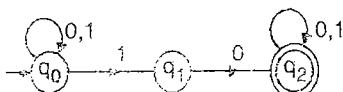


Fig. 2.3.1 : Transition diagram of a nondeterministic finite automata

If the automata is in the state q_0 and the next input symbol is 1, then the next state will be either :

- (1) q_0 (2) q_1

Thus, the move from q_0 on input 1 cannot be determined uniquely. Such machines are called Non-deterministic automata.

- Non-deterministic finite automata can be in several states at any time. Thus, NFA can guess about an input sequence.
- Every NFA can be converted into an equivalent DFA.
- An NFA can be designed with fewer states compare to its deterministic counterpart.
- Due to non-determinism, NFA takes more time to recognize a string.

2.3.1 Definition of NFA

A Non-deterministic finite automata is a 5-tuple.

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

Where,

Q = A finite set of states

Σ = A finite set of inputs

δ = A transition function from $Q \times \Sigma$ to the power set of Q i.e.to 2^Q .

$q_0 = q_0 \in Q$ is the start / initial state

$F = F \subseteq Q$ is a set of final / accepting states.

The NFA, for Fig. 2.3.1, can be formally represented as,

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

where, the transition function δ is given in Fig. 2.3.2.

| | 0 | 1 |
|-------------------|-----------|----------------|
| $\rightarrow q_0$ | $\{q_0\}$ | $\{q_0, q_1\}$ |
| q_1 | $\{q_2\}$ | \emptyset |
| q_2^* | $\{q_2\}$ | $\{q_2\}$ |

Fig. 2.3.2 : A transition table for NFA in Fig. 2.3.1

2.3.2 Processing of a String by NFA

A string $\omega \in \Sigma^*$ is accepted by NFA M if $\delta^*(q_0, \omega)$ contains a final state..

Let us see how the string 011010 is processed by the NFA given in Fig. 2.3.3.

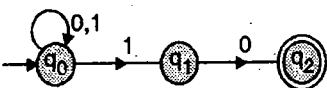


Fig. 2.3.3 : A sample NFA

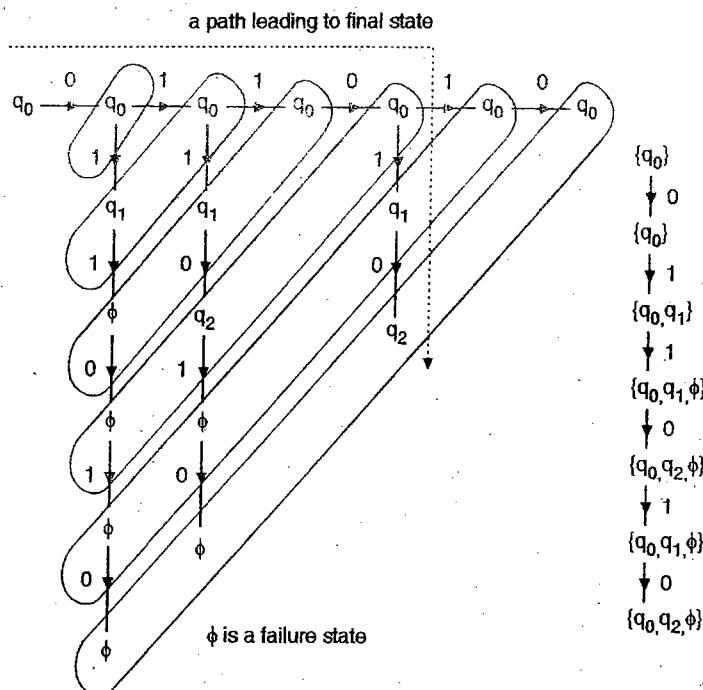


Fig. 2.3.4 : States reached while processing of 011010

- From the state q_0 , after input 0, resulting state is q_0 .
 - From the state q_0 , after input 1, resulting states are q_0, q_1 .
- An input 0 in q_0 , generates two paths :

1. $q_0 \xrightarrow{1} q_0$
2. $q_0 \xrightarrow{1} q_1$

On input 011010, the automata can take any of the four paths :

1. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} \phi \xrightarrow{0} \phi \xrightarrow{1} \phi \xrightarrow{0} \phi$

[Once in ϕ state (dead state) machine will remain in ϕ state]

2. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} \phi \xrightarrow{0} \phi$

3. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$

4. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0$

Out of the four paths, the path number 3 takes the machine from start state to the final state on input 011010. q_2 is a final state.

A string is accepted by non-deterministic finite automata, if there exists a path that takes the machine to a final state.

Thus, 011010 will be accepted by the NFA of Fig. 2.3.3.

- The language accepted by the above NFA consists of string ending in 10. A string of length n ($|w| = n$), ending 10 can be accepted by the above NFA if :

1. The machine remains in q_0 for first $n-2$ inputs.

2. On last two inputs 10, the machine makes the transition as shown below :

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

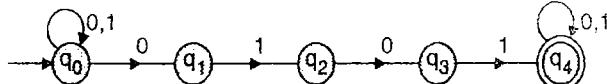
- To reach the final state, and to remain there, it is necessary that the last two characters of the string should be 10.

Thus the NFA in Fig. 2.3.3 accepts a string, if and only if it ends in 10.

**Example 2.3.1**

Draw a non-deterministic automata to accept strings containing the substring 0101.

Solution :



(a) State transition diagram

| | 0 | 1 |
|-------------------|----------------|-------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| q_1 | \emptyset | $\{q_2\}$ |
| q_2 | $\{q_3\}$ | \emptyset |
| q_3 | \emptyset | $\{q_4\}$ |
| q_4^* | $\{q_4\}$ | $\{q_4\}$ |

(b) State transition table

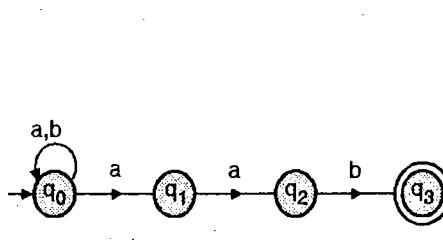
Fig. Ex. 2.3.1

- For reaching the final state q_4 , from the start state q_0 , a substring 0101 is required.
- Any string, containing a substring 0101 will be accepted by the above NFA. The machine can remain in q_0 , for the portion of string before 0101. The machine can remain in q_4 for the portion of string after 0101.

Example 2.3.2

Construct a NFA that accepts any positive number of occurrences of various strings from the following language L given by : $L = \{x \in \{a,b\}^* \mid x \text{ ends with } aab\}$

Solution :



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

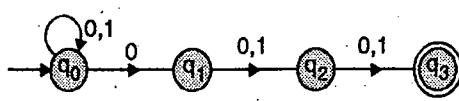
Fig. Ex. 2.3.2

- The language accepted by the NFA shown in Fig. Ex. 2.3.2 consists of string ending in aab. A string of length n , ending in aab can be accepted by the NFA if :
 1. The machine remains in q_0 for first $n-3$ inputs.
 2. On last three inputs aab, the machine makes the transition as : $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$
- To reach the final state, and to remain there, it is necessary that the last two characters of the string should be 10.
- Thus the NFA accepts a string, if and only if it ends in 10.

Example 2.3.3

Draw a non-deterministic finite automata to accept strings over alphabet $\{0, 1\}$, such that the third symbol from the right end is 0.

Solution :



(a) State transition diagram

| | 0 | 1 |
|-------------------|----------------|-------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| q_1 | $\{q_2\}$ | $\{q_2\}$ |
| q_2 | $\{q_3\}$ | $\{q_3\}$ |
| q_3^* | \emptyset | \emptyset |

(b) State transition table

Fig. Ex. 2.3.3 : NFA for Example 2.3.3

- For the first $n-3$ symbols, the machine can remain in the start state q_0 .
- On seeing the third symbol from the right end as 0 (NFA can guess), it makes a move towards the final state q_3 .

Example 2.3.4

Design a NFA to recognize the following sets of strings : abc, abd, aacd. Assume that alphabet is {a, b, c, d}. Give the transition table and transition diagram data.

Solution :

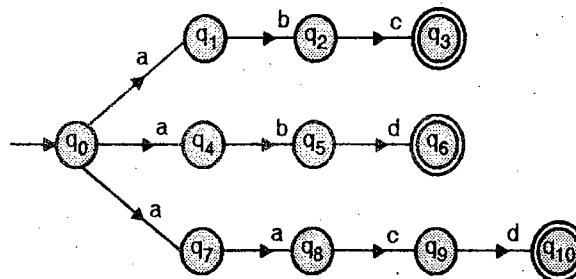


Fig. Ex. 2.3.4 : State diagram of NFA

- Machine can guess and take one of the three paths as shown :
 - Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$ for string abc.
 - Path $q_0 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6$ for string abd.
 - Path $q_0 \rightarrow q_7 \rightarrow q_8 \rightarrow q_9 \rightarrow q_{10}$ for string aacd.

Table Ex. 2.3.4 : State table for NFA

| | a | b | c | d |
|-------------------|---------------------|-------------|-------------|-------------|
| $\rightarrow q_0$ | $\{q_1, q_4, q_7\}$ | \emptyset | \emptyset | \emptyset |
| q_1 | \emptyset | q_2 | \emptyset | \emptyset |
| q_2 | \emptyset | \emptyset | q_3 | \emptyset |
| q_3^* | \emptyset | \emptyset | \emptyset | \emptyset |
| q_4 | \emptyset | q_5 | \emptyset | \emptyset |
| q_5 | \emptyset | \emptyset | \emptyset | q_6 |
| q_6^* | \emptyset | \emptyset | \emptyset | \emptyset |
| q_7 | q_8 | \emptyset | \emptyset | \emptyset |
| q_8 | \emptyset | \emptyset | q_9 | \emptyset |
| q_9 | \emptyset | \emptyset | \emptyset | q_{10} |
| q_{10}^* | \emptyset | \emptyset | \emptyset | \emptyset |

Example 2.3.5

Construct the NFA and DFA for the following languages.

- $L = \{x \in \{a, b, c\}^*: x \text{ contains exactly one } b \text{ immediately following } c\}$
- $L = \{x \in \{0, 1\}^*: x \text{ is starting with } 1 \text{ and } |x| \text{ is divisible by } 3\}$
- $L = \{x \in \{a, b\}^*: x \text{ contains any number of } a's \text{ followed by at least one } b\}$

Solution :

- $L = \{x \in \{a, b, c\}^*: x \text{ contains exactly one } b \text{ immediately following } c\}$

Since, the problem is full deterministic in nature; it will serve no purpose to design an NFA. We can design a DFA and since every DFA is also an NFA, a DFA can stand for both DFA and NFA.

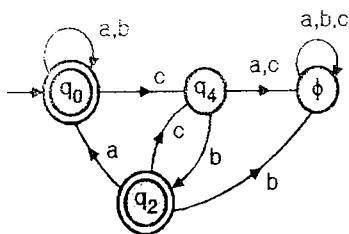


Fig. Ex. 2.3.5(a) : A DFA standing for both DFA and NFA

- (ii) $L \{x \in \{0, 1\}^*: x \text{ is starting with } 1 \text{ and } |x| \text{ is divisible by 3}\}$

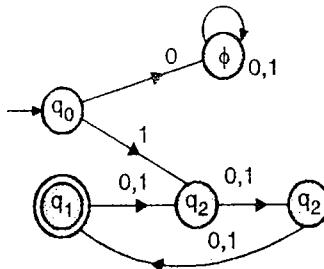


Fig. Ex. 2.3.5(b) : A DFA standing for both DFA and NFA

- (iii) $L \{x \in \{a, b\}^*: x \text{ contains any number of } a's \text{ followed by at least one } b\}$

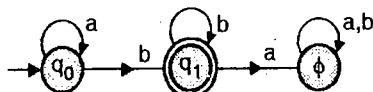


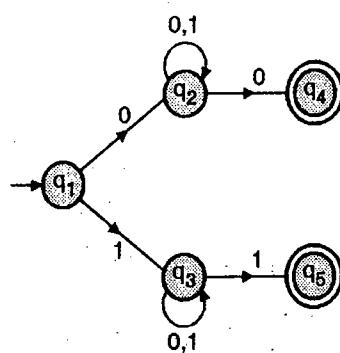
Fig. Ex. 2.3.5(c) : A DFA standing for both DFA and NFA

Note : Technically, a DFA with ϕ transitions omitted may be called an NFA.

Example 2.3.6

Design a NFA for a binary number where the first and the last digits are same.

Solution :



(a) State transition diagram

| | 0 | 1 |
|-------------------|----------------|----------------|
| $\rightarrow q_1$ | q_2 | q_3 |
| q_2 | $\{q_2, q_4\}$ | q_2 |
| q_3 | q_3 | $\{q_3, q_4\}$ |
| q_4^* | — | — |
| q_5^* | — | — |

(b) State transition table

Fig. Ex. 2.3.6

- If the starting and ending digits are 0 then NFA will end in q_4 by making a move to q_2 from q_1 on first 0 thereafter it will wait non-deterministically in q_2 and on last 0 it will enter the final state q_4 .
- If the starting and ending digits are 1 then NFA will end in q_5 by making a move to q_3 from q_1 on first 1 thereafter it will wait non-deterministically in q_3 and on last 1 it will enter the final state q_5 .

Example 2.3.7

An NFA with states 1-5 and input alphabet {a, b} has following transition table.

- (a) Draw a transition diagram (b) Calculate $\delta^*(1, ab)$ (c) Calculate $\delta^*(1, abaab)$

| q | $\delta(q, a)$ | $\delta(q, b)$ |
|---|----------------|----------------|
| 1 | {1, 2} | {1} |
| 2 | {3} | {3} |
| 3 | {4} | {4} |
| 4 | {5} | \emptyset |
| 5 | \emptyset | {5} |

Solution :

- (a) Transition diagram is given in Fig. Ex. 2.3.7.

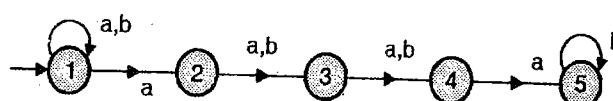


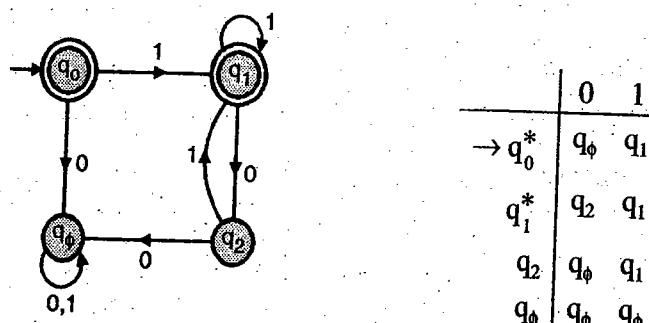
Fig. Ex. 2.3.7 : State transition diagram

$$\begin{aligned}
 \text{(b)} \quad \delta^*(1, ab) &= \delta((\delta(1, a)), b) = \delta(\{1, 2\}, b) = \delta(1, b) \cup \delta(2, b) \\
 &= \{1\} \cup \{3\} = \{1, 3\} \\
 \text{(c)} \quad \delta^*(1, abaab) &= \delta^*(\{1, 2\}, baab) = \delta^*(\{1, 3\}, aab) \\
 &= \delta^*(\{1, 2, 4\}, ab) = \delta(\{1, 2, 3, 5\}, b) = \{1, 3, 4, 5\}
 \end{aligned}$$

Example 2.3.8

Construct a FA for accepting L over {0,1} such that each 0 is immediately preceded and immediately followed by 1.

Solution :



| | 0 | 1 |
|---------------------|----------|----------|
| $\rightarrow q_0^*$ | q_ϕ | q_1 |
| q_1^* | q_2 | q_1 |
| q_2 | q_ϕ | q_1 |
| q_ϕ | q_ϕ | q_ϕ |

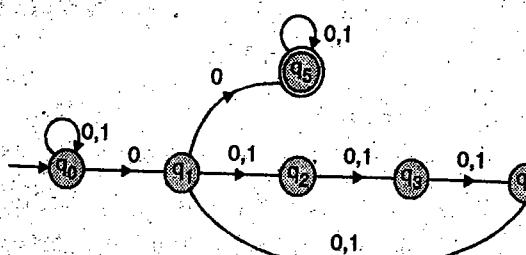
Fig. Ex. 2.3.8

- If the starting symbol is 0, string is rejected by entering the failure state q_ϕ .
- A loop through $q_1 \rightarrow q_2 \rightarrow q_1$ ensures that every 0 is preceded and succeeded by 1.
- An input 0 in state q_2 will mean two consecutive 0's. The string is rejected by entering the failure state q_ϕ .

Example 2.3.9

Construct a NFA that accept the set of strings in $(0 + 1)^*$ such that some two 0's are separated by string whose length is $4i$, for some $i \geq 0$.

Solution :



(a) State transition diagram

Fig. Ex. 2.3.9



| | 0 | 1 |
|-------------------|----------------|-----------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| q_1 | $\{q_2, q_5\}$ | $\{q_2\}$ |
| q_2 | $\{q_3\}$ | $\{q_3\}$ |
| q_3 | $\{q_4\}$ | $\{q_4\}$ |
| q_4 | $\{q_1\}$ | $\{q_1\}$ |
| q_5^* | $\{q_5\}$ | $\{q_5\}$ |

(b) State transition table

Fig. Ex. 2.3.9 : NFA for Example 2.3.9

- The loop $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_1$ can absorb a string of length 4i.
- On receiving the 0 of the portion of the string containing 'two 0's separated by string of length 4i, the machine makes a move to q_1 from q_0 . Then, it loops i times in $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_1$. Then it gets a 0 and makes a move from q_1 to q_5 .

Example 2.3.10 SPPU - Dec. 15, 5 Marks

Design an FA for the languages that contain strings with next-to-last symbol 0.

Solution : Required FA (NFA) is given below

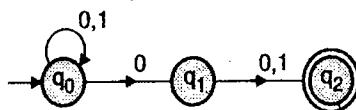


Fig. Ex. 2.3.10

Example 2.3.11

Design NFA for the set of strings on the alphabet {0, 1} that start with 01 and end with 10.

Solution :

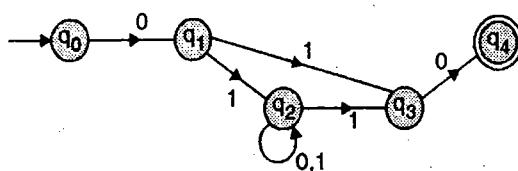


Fig. Ex. 2.3.11 : NFA for Example 2.3.11

- A string 010 (starting with 01 and ending with 10) can be accepted through the path $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_4$.
- For any other string, the portion between starting 01 and ending 10 can be absorbed by the state q_2 .

2.3.3 NFA to DFA Conversion

The NFA to DFA conversion is based on subset construction. If a non-deterministic finite state machine consists of three states.

$$\text{i.e. } Q = (q_0, q_1, q_2)$$

The machine could be in any of the following states :

- | | |
|---|--------------------------|
| 1. \emptyset | 2. $(q_0), (q_1), (q_2)$ |
| 3. $(q_0, q_1), (q_0, q_2), (q_1, q_2)$ | 4. (q_0, q_1, q_2) |

These subsets form a power set on Q , which is given by 2^Q .

The procedure for finding whether a given string is accepted by the NFA, involves :

1. Tracing the various paths followed by the machine for the given string.
2. Finding the set of states reached from the starting state by applying the symbols of the string.
3. If the set of states obtained in step (2), contains a final state, the given string is accepted by the NFA.

The above procedure can be applied to any string by constructing a successor table.

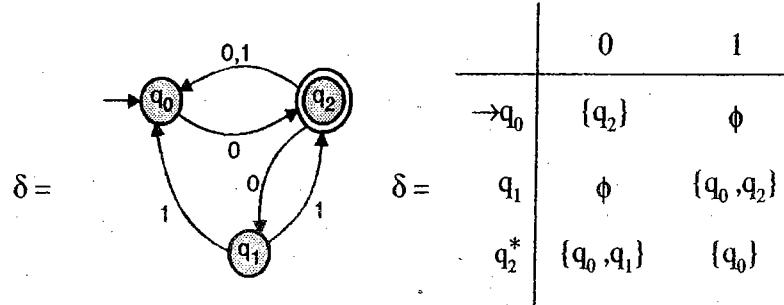


Fig. 2.3.5 : NFA to be converted to DFA

Algorithm for construction of successor table for the NFA of Fig. 2.3.5 is being explained, stepwise :

Step 1 : Starting state $\{q_0\}$ is the first subset:

0 – successor of q_0 is q_2 , i.e. $\delta(q_0, 0) \Rightarrow q_2$,

1 – successor of q_0 is ϕ , i.e. $\delta(q_0, 1) \Rightarrow \phi$.

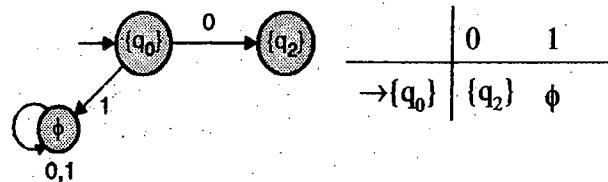


Fig. 2.3.6

Step 2 : A new subset $\{q_2\}$ is generated. Successor of the subset $\{q_2\}$ are generated.

0 – successor of $\{q_2\}$ are $\{q_0, q_1\}$,

1 – successor of $\{q_2\}$ are $\{q_0\}$.

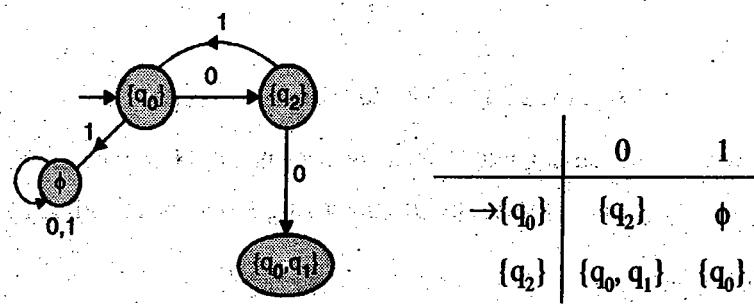


Fig. 2.3.7

Step 3 : A new subset $\{q_0, q_1\}$ is generated. Successors of the subset $\{q_0, q_1\}$ are generated.

$$\begin{aligned} 0 - \text{successor of } \{q_0, q_1\} &= \delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \\ &= \{q_2\} \cup \phi = \{q_2\} \end{aligned}$$

$$\begin{aligned} 1 - \text{successor of } \{q_0, q_1\} &= \delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \\ &= \phi \cup \{q_0, q_2\} = \{q_0, q_2\} \end{aligned}$$

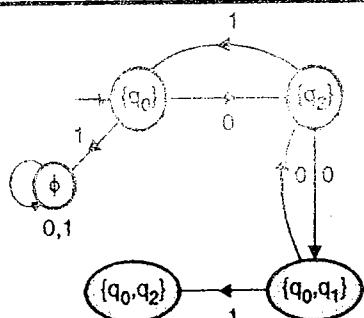


Fig. 2.3.8

| | 0 | 1 |
|----------------------|----------------|----------------|
| $\rightarrow\{q_0\}$ | $\{q_2\}$ | ϕ |
| $\{q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_0, q_1\}$ | $\{q_2\}$ | $\{q_0, q_2\}$ |

Step 4 : A new subset $\{q_0, q_2\}$ is generated successors of the subset $\{q_0, q_2\}$ are generated.

$$\begin{aligned}
 0 - \text{successor of } \{q_0, q_2\} &= \delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) \\
 &= \{q_2\} \cup \{q_2, q_1\} = \{q_0, q_1, q_2\} \\
 1 - \text{successor of } \{q_0, q_2\} &= \delta(\{q_0, q_2\}, 1) \\
 &= \delta(q_0, 1) \cup \delta(q_2, 1) = \phi \cup \{q_0\} = \{q_0\}
 \end{aligned}$$

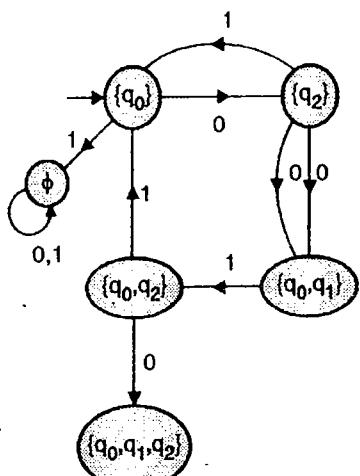
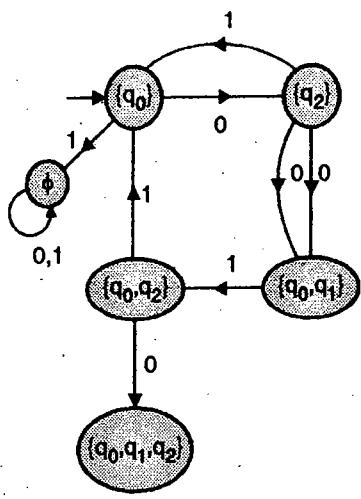


Fig. 2.3.9

| | 0 | 1 |
|----------------------|---------------------|----------------|
| $\rightarrow\{q_0\}$ | $\{q_2\}$ | ϕ |
| $\{q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_0, q_1\}$ | $\{q_2\}$ | $\{q_0, q_2\}$ |
| $\{q_0, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0\}$ |

Step 5 : A new subset $\{q_0, q_1, q_2\}$ is generated successors of the subset $\{q_0, q_1, q_2\}$ are generated.

$$\begin{aligned}
 0 - \text{successor of } \{q_0, q_1, q_2\} &= \delta(\{q_0, q_1, q_2\}, 0) = \delta(\{q_0, q_1\}, 0) \cup \delta(q_2, 0) \\
 &= \{q_2\} \cup \{q_0, q_1\} = \{q_0, q_1, q_2\} \\
 1 - \text{successor of } \{q_0, q_1, q_2\} &= \delta(\{q_0, q_1, q_2\}, 1) \\
 &= \delta(\{q_0, q_1\}, 1) \cup \delta(q_2, 1) = \{q_0, q_2\} \cup \{q_0\} = \{q_0, q_2\}
 \end{aligned}$$



| | 0 | 1 |
|----------------------|---------------------|----------------|
| $\rightarrow\{q_0\}$ | $\{q_2\}$ | ϕ |
| $\{q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_0, q_1\}$ | $\{q_2\}$ | $\{q_0, q_2\}$ |
| $\{q_0, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0\}$ |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_0\}$ |

Fig. 2.3.10

Since, a new subset is not generated the process of subset generation stops.

Step 6: q_2 is the final state in NFA of Fig. 2.3.5. Every subset containing q_2 should be taken as a final state.

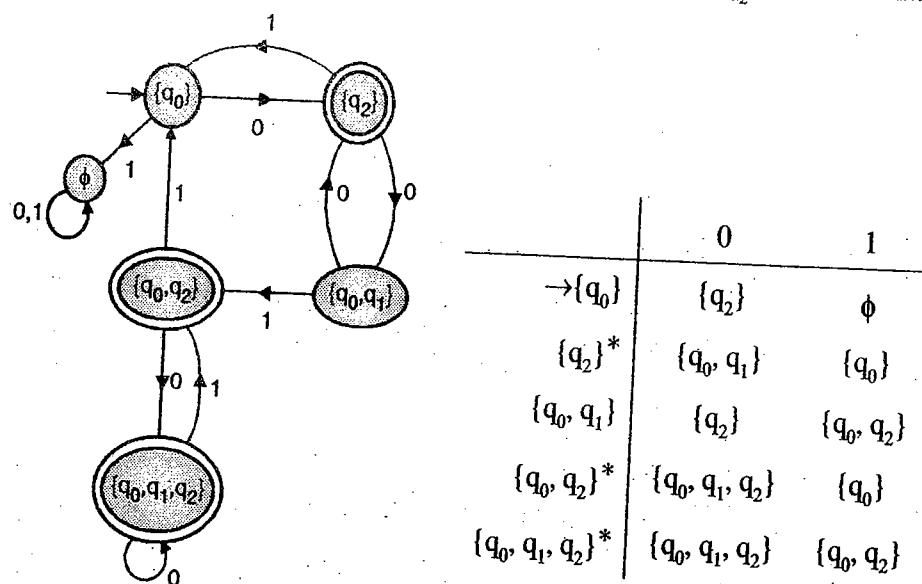


Fig. 2.3.11 : Final DFA after marking of accepting states

Syllabus Topic : Equivalence of NFA and DFA

Theorem 2.1 : A language L is accepted by some NFA if and only if it is accepted by some DFA.
OR

For every NFA, there exists an equivalent DFA.

Proof : This theorem has two parts :

1. If L is accepted by a DFA M_2 , then L is accepted by some NFA M_1 .
2. If L is accepted by an NFA M_1 , then L is accepted by some DFA M_2 .

First part can be proved trivially. Determinism is a case of non-determinism. Thus a DFA is also an NFA.

Second part of the theorem is proved below :

Construct M_2 from M_1 using subset generation algorithm as explained earlier. We can prove the theorem using induction on the length of ω .

Base case : Let $\omega = \epsilon$ with $|\omega| = 0$, where $|\omega|$ is length of ω .

Starting state for both NFA and DFA are taken as q_0 . When $\omega = \epsilon$, both DFA and NFA will be in q_0 . Hence, the base case is proved.

Assumption : Let us assume that both NFA and DFA are equivalent for every string of length n. We must show that the machines M_1 (NFA) and M_2 (DFA) are equivalent for strings of length $(n+1)$.

Let $\omega_{n+1} = \omega_n a$, where ω_n is a string of length n and ω_{n+1} is a string of length $(n+1)$. 'a' is an arbitrary alphabet from Σ .

$\delta_2(q_2, \omega_n) = \delta_2(q_0, \omega_n)$, where δ_2 is transition function of DFA (M_2) and δ , is transition function of NFA (M_1).

If the subset reached by NFA is given by $\{p_1, p_2, \dots, p_k\}$

then,

$$\delta_2(q_0, \omega_{n+1}) = \bigcup_{i=1}^k \delta_2(p_i, a) \quad \dots(i)$$



$$\text{or } \delta_2(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_1(p_i, a) \quad \dots \text{(ii)}$$

also, $\delta_2(q_0, \omega_n) = \{p_1, p_2, \dots, p_k\} \quad \dots \text{(iii)}$

from (i), (ii) and (iii) we get,

$$\begin{aligned} \delta_2(q_0, \omega_{n+1}) &= \delta_2(\delta_2(q_0, \omega_n), a) = \delta_2(\{p_1, p_2, \dots, p_k\}, a) \\ &= \bigcup_{i=1}^k \delta_1(p_i, a) = \delta_1(q_0, s_{n+1}) \end{aligned}$$

Thus, the result is true for $|\omega| = n + 1$, hence it is always true.

Example 2.3.12

Convert to a DFA the following NFA :

| | | |
|-----------------|--------|-------------|
| | 0 | 1 |
| $\rightarrow p$ | {p, q} | {p} |
| q | {r} | {r} |
| r | {s} | \emptyset |
| s^* | {s} | {s} |

Solution :

Table Ex. 2.3.12 : State transition table of DFA

| | 0 | 1 | |
|---------------------|--------------|-----------|--|
| $\rightarrow \{p\}$ | {p, q} | {p} | A new subset {p, q} is generated. |
| {p, q} | {p, q, r} | {p, r} | Two new subsets {p, q, r} and {p, r} are generated |
| {p, q, r} | {p, q, r, s} | {p, r} | A new subset {p, q, r, s} is generated |
| {p, r} | {p, q, s} | {p} | A new subset {p, q, s} is generated |
| $\{p, q, r, s\}^*$ | {p, q, r, s} | {p, r, s} | A new subset {p, r, s} is generated |
| $\{p, r, s\}^*$ | {p, q, s} | {p, s} | A new subset {p, s} is generated |
| $\{p, s\}^*$ | {p, q, s} | {p, s} | |

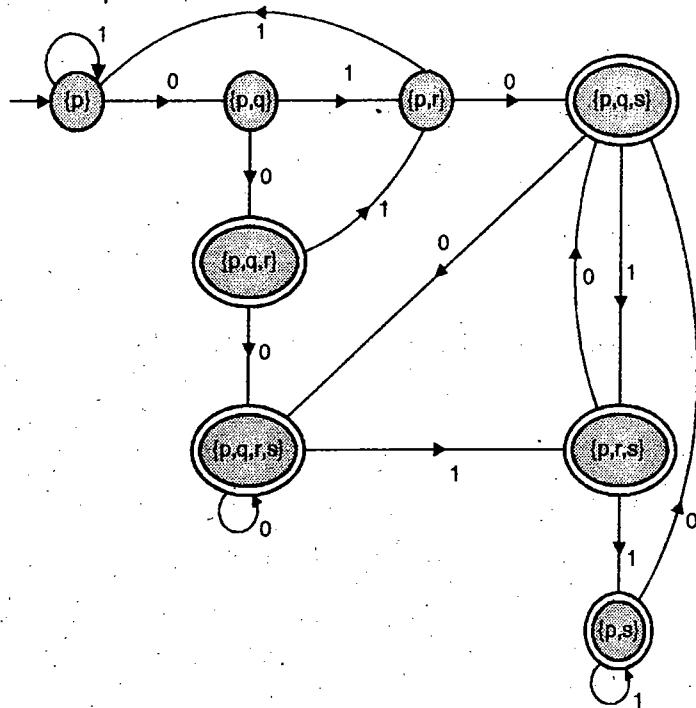


Fig. Ex. 2.3.12 : State transition diagram of DFA

- Subsets are generated recursively.
- Every subset is selected for generation of successor subsets.
- Every subset containing s (final state of the given NFA) should be marked as a final state.

Example 2.3.13 SPPU Dec 13 8 Marks

Convert to a DFA the following NFA.

| | 0 | 1 |
|---------------------|-------------|------------|
| $\rightarrow \{p\}$ | $\{q, s\}$ | $\{q\}$ |
| q^* | $\{r\}$ | $\{q, r\}$ |
| r | $\{s\}$ | $\{p\}$ |
| s^* | \emptyset | $\{p\}$ |

Solution :

Step 1 : $\{p\}$ is taken as the first subset.

$$0\text{-Successor of } \{p\} = \delta(\{p\}, 0) = \{q, s\}$$

$$1\text{-Successor of } \{p\} = \delta(\{p\}, 1) = \{q\}$$

| | 0 | 1 |
|---------------------|------------|---------|
| $\rightarrow \{p\}$ | $\{q, s\}$ | $\{q\}$ |

Step 2 : Two new subsets $\{q, s\}$ and $\{q\}$ have been generated. Successors of $\{q, s\}$ and $\{q\}$ are calculated.

$$\delta(\{q, s\}, 0) = \delta(q, 0) \cup \delta(s, 0) = \{r\} \cup \emptyset = \{r\}$$

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

$$\delta(\{q\}, 0) = \{r\}$$

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

| | 0 | 1 |
|---------------------|------------|---------------|
| $\rightarrow \{p\}$ | $\{q, s\}$ | $\{q\}$ |
| $\{q, s\}$ | $\{r\}$ | $\{p, q, r\}$ |
| $\{q\}$ | $\{r\}$ | $\{q, r\}$ |

Step 3 : Three new subsets $\{r\}$, $\{p, q, r\}$ and $\{q, r\}$ are generated.

Their successors are calculated.

$$\delta(\{r\}, 0) = \{s\}$$

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

$$\begin{aligned}\delta(\{p, q, r\}, 0) &= \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) = \{q, s\} \cup \{r\} \cup \{s\} \\ &= \{q, r, s\}\end{aligned}$$

$$\begin{aligned}\delta(\{p, q, r\}, 1) &= \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1) = \{q\} \cup \{q, r\} \cup \{p\} \\ &= \{p, q, r\}\end{aligned}$$

$$\delta(\{q, r\}, 0) = \delta(q, 0) \cup \delta(r, 0) = \{r\} \cup \{s\} = \{r, s\}$$

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



| | 0 | 1 |
|------------------------|---------------|---------------|
| $\rightarrow\{\cdot\}$ | $\{q, s\}$ | $\{q\}$ |
| $\{q, s\}$ | $\{r\}$ | $\{p, q, r\}$ |
| $\{q\}$ | $\{r\}$ | $\{q, r\}$ |
| $\{r\}$ | $\{s\}$ | $\{p\}$ |
| $\{q, r\}$ | $\{r, s\}$ | $\{p, q, r\}$ |
| $\{p, q, r\}$ | $\{q, r, s\}$ | $\{p, q, r\}$ |

Step 4 : Three new subsets $\{s\}$, $\{r, s\}$ and $\{q, r, s\}$ are generated. Their successors are calculated.

$$\delta(\{s\}, 0) = \emptyset$$

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

$$\delta(\{r, s\}, 0) = \{s\}$$

$$\delta(\{r, s\}, 1) = \{p\}$$

$$\delta(\{q, r, s\}, 0) = \{r, s\}$$

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

| | 0 | 1 |
|--------------------|---------------|---------------|
| $\rightarrow\{p\}$ | $\{q, s\}$ | $\{q\}$ |
| $\{q, s\}$ | $\{r\}$ | $\{p, q, r\}$ |
| $\{q\}$ | $\{r\}$ | $\{q, r\}$ |
| $\{r\}$ | $\{s\}$ | $\{p\}$ |
| $\{q, r\}$ | $\{r, s\}$ | $\{p, q, r\}$ |
| $\{p, q, r\}$ | $\{q, r, s\}$ | $\{p, q, r\}$ |
| $\{s\}$ | \emptyset | $\{p\}$ |
| $\{r, s\}$ | $\{s\}$ | $\{p\}$ |
| $\{q, r, s\}$ | $\{r, s\}$ | $\{p, q, r\}$ |

Step 5 : Every subset containing q or s is declared as a final state. Final DFA is shown in Fig. Ex. 2.3.13.

Table Ex. 2.3.13 : State transition table of DFA

| | 0 | 1 |
|--------------------|---------------|---------------|
| $\rightarrow\{p\}$ | $\{q, s\}$ | $\{q\}$ |
| $\{q, s\}^*$ | $\{r\}$ | $\{p, q, r\}$ |
| $\{q\}^*$ | $\{r\}$ | $\{q, r\}$ |
| $\{r\}$ | $\{s\}$ | $\{p\}$ |
| $\{q, r\}^*$ | $\{r, s\}$ | $\{p, q, r\}$ |
| $\{p, q, r\}^*$ | $\{q, r, s\}$ | $\{p, q, r\}$ |
| $\{s\}^*$ | \emptyset | $\{p\}$ |
| $\{r, s\}^*$ | $\{s\}$ | $\{p\}$ |
| $\{q, r, s\}^*$ | $\{r, s\}$ | $\{p, q, r\}$ |

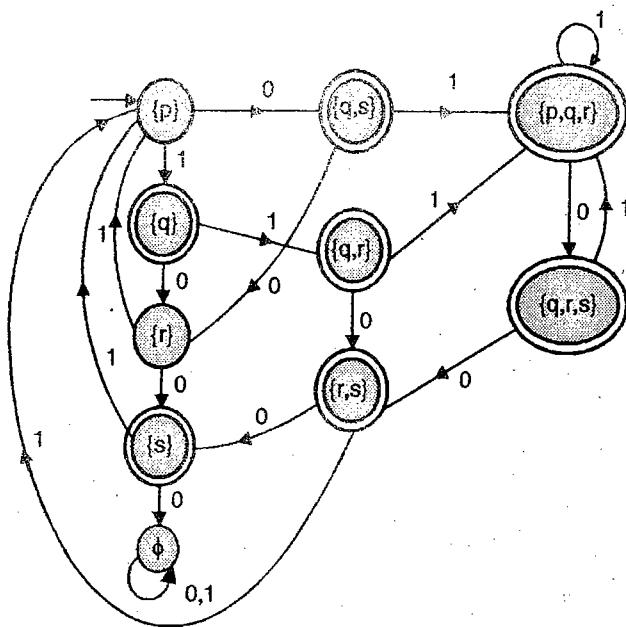


Fig. Ex. 2.3.13 : State transition diagram of the DFA

Example 2.3.14 SPPU - May 12, 8 Marks

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

| | 0 | 1 |
|-----------------|-------------|-------------|
| $\rightarrow p$ | {p, q} | {p} |
| q | {r, s} | {t} |
| r | {p, r} | {t} |
| s^* | \emptyset | \emptyset |
| t^* | \emptyset | \emptyset |

Solution :

Step 1 : {p} is taken as the first subset.

$$0\text{-Successor of } \{p\} = \delta(\{p\}, 0) = \{p, q\}$$

$$1\text{-Successor of } \{p\} = \delta(\{p\}, 1) = \{p\}$$

Step 2 : The new subsets {p, q} is generated. Successors of {p, q} are calculated.

$$\delta(\{p, q\}, 0) = \delta(p, 0) \cup \delta(q, 0) = \{p, q\} \cup \{r, s\} = \{p, q, r, s\}$$

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

Step 3 : Two new subsets {p, q, r, s} and {p, t} are generated. Their successors are calculated.

$$\begin{aligned}\delta(\{p, q, r, s\}, 0) &= \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) \cup \delta(s, 0) \\ &= \{p, q\} \cup \{r, s\} \cup \{p, r\} \cup \emptyset = \{p, q, r, s\}\end{aligned}$$

$$\begin{aligned}\delta(\{p, q, r, s\}, 1) &= \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1) \cup \delta(s, 1) \\ &= \{q\} \cup \{t\} \cup \{t\} \cup \emptyset = \{p, t\}\end{aligned}$$

$$\delta(\{p, t\}, 0) = \delta(p, 0) \cup \delta(t, 0) = \{p, q\} \cup \emptyset = \{p, q\}$$

$$\delta(\{p, t\}, 1) = \delta(p, 1) \cup \delta(t, 1) = \{p\} \cup \emptyset = \{p\}$$

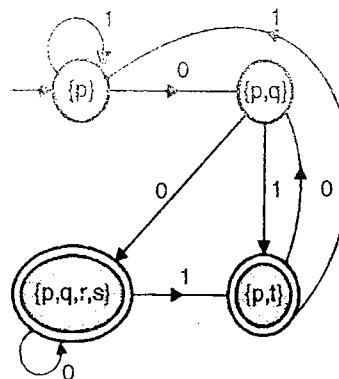
No, new subset is generated. Every subset containing either s or t is marked as a final state.



Informal Description: Strings over $\{0, 1\}$ with second digit from the end is 0.

| | 0 | 1 |
|---------------------|------------------|------------|
| $\rightarrow \{p\}$ | $\{p, q\}$ | $\{p\}$ |
| $\{p, q\}$ | $\{p, q, r, s\}$ | $\{p, t\}$ |
| $\{p, q, r, s\}^*$ | $\{p, q, r, s\}$ | $\{p, t\}$ |
| $\{p, t\}^*$ | $\{p, q\}$ | $\{p\}$ |

(a) State table



(b) State diagram

Fig. Ex. 2.3.14 : Final DFA for Example 2.3.14

Example 2.3.15

Construct a NFA that accepts a set of all strings over $\{a, b\}$ ending in aba. Use this NFA to construct DFA accepting the same set of strings.

Solution :

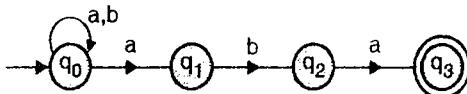


Fig. Ex. 2.3.15(a) : Non-deterministic finite automata for Example 2.3.15

Non-determinism should be utilized to full extent while designing an NFA. A string of length n , ending in aba can be recognized by the NFA given in Fig. Ex. 2.3.15(a). First $n-3$ characters can be absorbed by the state q_0 by making a guess. On guessing the last three characters as aba, the machine can make a transition from q_0 to q_3 .

NFA to DFA conversion :

Step 1 : $\{q_0\}$ is taken as first subset

$$\text{a-successor of } \{q_0\} = \delta(q_0, a) = \{q_0, q_1\}$$

$$\text{b-successor of } \{q_0\} = \delta(q_0, b) = \{q_0\}$$

Step 2 : A new subset $\{q_0, q_1\}$ is generated. Successors of $\{q_0, q_1\}$ are calculated.

$$\begin{aligned}\delta(\{q_0, q_1\}, a) &= \delta(q_0, a) \cup \delta(q_1, a) \\ &= \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}\end{aligned}$$

$$\begin{aligned}\delta(\{q_0, q_1\}, b) &= \delta(q_0, b) \cup \delta(q_1, b) \\ &= \{q_0\} \cup \{q_2\} = \{q_0, q_2\}\end{aligned}$$

Step 3 : A new subset $\{q_0, q_2\}$ is generated. Successors of $\{q_0, q_2\}$ are calculated.

$$\delta(\{q_0, q_2\}, a) = \delta(q_0, a) \cup \delta(q_2, a) = \{q_0, q_1\} \cup \{q_3\} = \{q_0, q_1, q_3\}$$

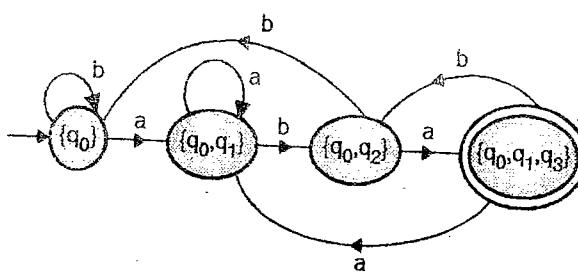
$$\delta(\{q_0, q_2\}, b) = \delta(q_0, b) \cup \delta(q_2, b) = \{q_0\} \cup \emptyset = \{q_0\}$$

Step 4 : A new subset $\{q_0, q_1, q_3\}$ is generated. Successors of $\{q_0, q_1, q_3\}$ are calculated.

$$\begin{aligned}\delta(\{q_0, q_1, q_3\}, a) &= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a) \\ &= \{q_0, q_1\} \cup \emptyset \cup \emptyset = \{q_0, q_1\}\end{aligned}$$

$$\begin{aligned}\delta(\{q_0, q_1, q_3\}, b) &= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b) \\ &= \{q_0\} \cup \{q_2\} \cup \emptyset = \{q_0, q_2\}\end{aligned}$$

No, new subset is generated. Every subset containing q_3 is marked as a final state.



(b) State diagram of the DFA

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

(c) State table of the DFA

Fig. Ex. 2.3.15

Example 2.3.16 SPPU - May 16, 6 Marks

Obtain a DFA equivalent to the NFA.

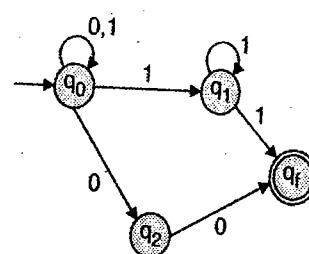


Fig. Ex. 2.3.16

Solution :

Stepwise construction of DFA :

Step 1 : Transition from the state q_0

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

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

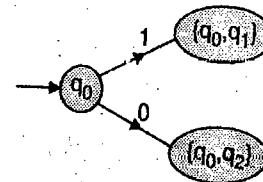


Fig. Ex. 2.3.16 (a)

Step 2 : Transitions from the state $\{q_0, q_2\}$

$$\begin{aligned}\delta(\{q_0, q_2\}, 0) &= \delta\{q_0, 0\} \cup \delta\{q_2, 0\} \\ &= \{q_0, q_2\} \cup q_f = \{q_0, q_2, q_f\}\end{aligned}$$

$$\begin{aligned}\delta(\{q_0, q_2\}, 1) &= \delta\{q_0, 1\} \cup \delta\{q_2, 1\} \\ &= \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}\end{aligned}$$

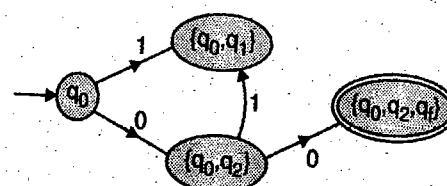
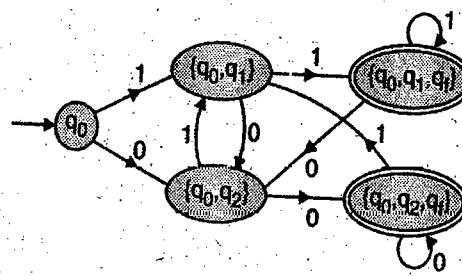


Fig. Ex. 2.3.16 (b)

Step 3 : Transitions from the state $\{q_0, q_1\}$

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

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



Further, there are no transitions from q_f and hence,

$$\delta(q_0, q_1, q_f) = \delta(q_0, q_1) \text{ and}$$

$$\delta(q_0, q_2, q_f) = \delta(q_0, q_2)$$

Fig. Ex. 2.3.16 (c)

**Example 2.3.17**

Construct a NFA and its equivalent DFA for accepting a language defined over $\Sigma = \{0, 1\}$ such that each string has two consecutive zeroes followed by 1.

Solution :

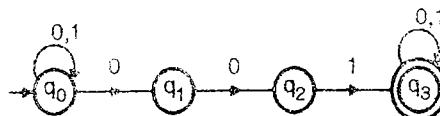


Fig. Ex. 2.3.17

A string of length n , having two consecutive zeroes followed by 1 can be recognized by the NFA. Symbols before 001 can be absorbed by the state q_0 by making a guess. On guessing 001, the machine can make a transition from q_0 to q_3 .

NFA to DFA conversion

| | 0 | 1 | |
|----------------------------|--------------------------|----------------|--|
| $\rightarrow \{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ | A new subset $\{q_0, q_1\}$ is generated |
| $\{q_0, q_1\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0\}$ | A new subset $\{q_0, q_1, q_2\}$ is generated |
| $\{q_0, q_1, q_2\}$ | $\{q_0, q_1, q_2\}$ | $\{q_0, q_3\}$ | A new subset $\{q_0, q_3\}$ is generated |
| $\{q_0, q_3\}^*$ | $\{q_0, q_1, q_3\}$ | $\{q_0, q_3\}$ | A new subset $\{q_0, q_1, q_3\}$ is generated |
| $\{q_0, q_1, q_3\}^*$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_3\}$ | A new subset $\{q_0, q_1, q_2, q_3\}$ is generated |
| $\{q_0, q_1, q_2, q_3\}^*$ | $\{q_0, q_1, q_2, q_3\}$ | $\{q_0, q_3\}$ | |

Transition behaviour of $\{q_0, q_3\}$, $\{q_0, q_1, q_3\}$ and $\{q_0, q_1, q_2, q_3\}$ is identical. Therefore, these three states can be merged into a single state with resultant DFA as shown in Fig. Ex. 2.3.17(a) :

| | 0 | 1 | |
|--|---|---|--|
| $\rightarrow \{q_0\}$ as $\rightarrow A$ | B | A | |
| $\{q_0, q_1\}$ as B | C | A | |
| $\{q_0, q_1, q_2\}$ as C | C | D | |
| $\{q_0, q_3\}$ as D* | D | D | |

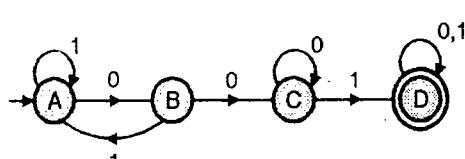


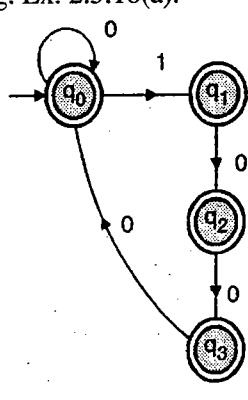
Fig. Ex. 2.3.17(a)

Example 2.3.18

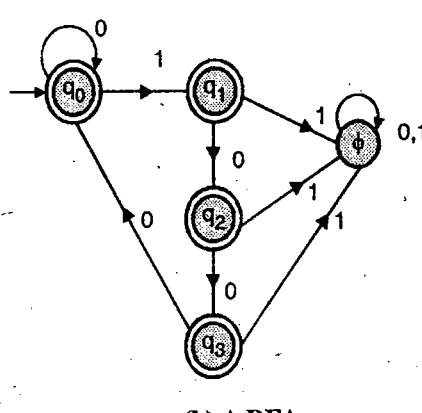
Construct an NFA and then equivalent DFA accepting strings over $\{0, 1\}$, whose every block of 4 consecutive symbols, contain at least 3 zeros.

Solution : This problem is fully deterministic in nature. Technically, a DFA with ϕ transitions not mentioned can be called a NFA.

NFA is given in Fig. Ex. 2.3.18(a).



(a) A NFA



(b) A DFA

Fig. Ex. 2.3.18

From Fig. Ex. 2.3.18(a), it is clear that between any two 1's there will be at least three 0's. Thus every block of 4 consecutive symbols will contain at least three 0's. Every state is a final state as the failure cases have not been shown.

Above NFA can be converted into a DFA by introducing the dead state. DFA is given in Fig. Ex. 2.3.18(b).

Example 2.3.19

Construct a NFA and then equivalent DFA accepting strings over $\{0, 1\}$, which accepts the set of all strings of zeros (i.e. 0's) and ones (i.e. 1's) with at most one pair of consecutive zeros and at most one pair of consecutive ones (i.e. 1's).

Solution : This problem is fully deterministic in nature. Technically, a DFA with ϕ transitions not mentioned can be called as NFA.

NFA is given in Fig. Ex. 2.3.19(a).

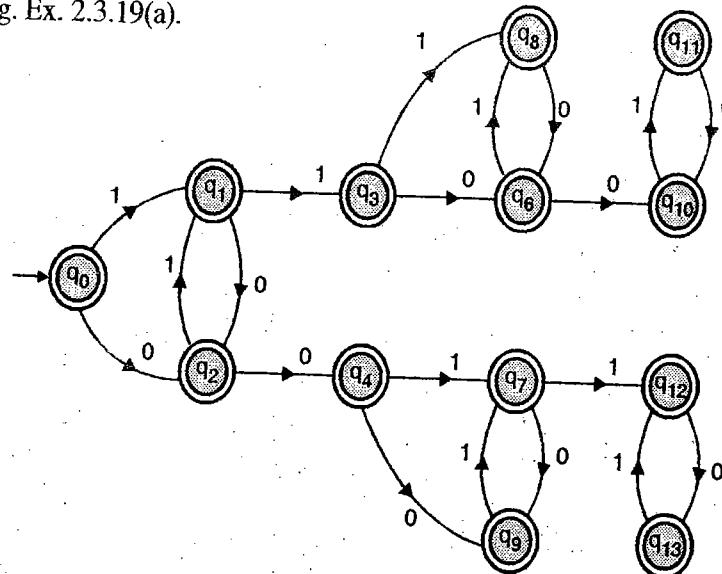


Fig. Ex. 2.3.19(a) : A NFA

Meaning of various states :

q_3 = A pair of 11 has been seen

q_4 = A pair of 00 has been seen

$q_{10}, q_{12} \rightarrow$ A pair of 00 and a pair of 11 have been seen

NFA in Fig. Ex. 2.3.19(a) can be converted into a DFA by introducing an explicit dead state.

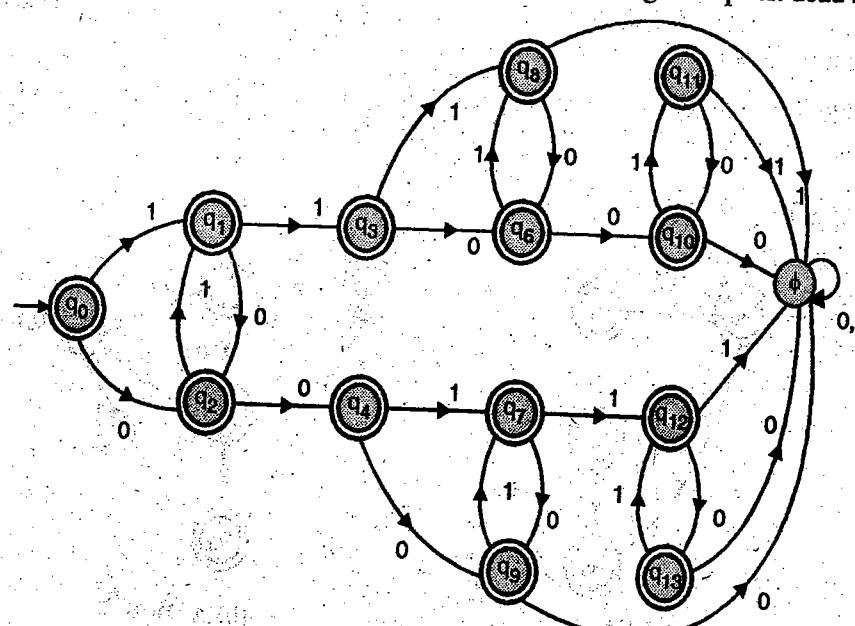


Fig. Ex. 2.3.19(b) : A DFA

**Example 2.3.20**

Construct an NFA and then equivalent DFA accepting string over $\{0, 1\}$, for accepting all possible strings of zero and ones not containing 101 as substring.

Solution : This problem is fully deterministic in nature as technically, a DFA with ϕ transitions not mentioned can be called as NFA.

- State q_1 is for previous character as 1.
- State q_2 is for previous two characters as 10.
- An input of 1 in state q_2 will cause the machine to enter a failure state.
- NFA can be converted into a DFA by introducing an explicit failure state. The equivalent DFA is given in Fig. Ex. 2.3.20(b).

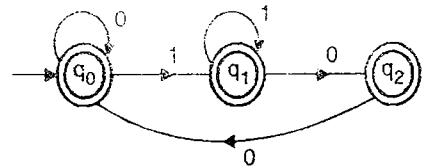


Fig. Ex. 2.3.20(a) : A NFA

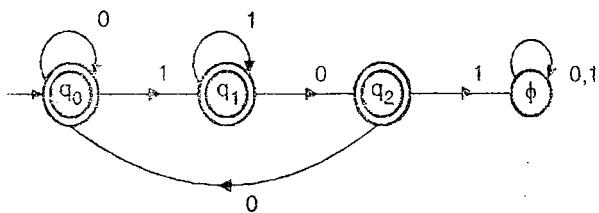


Fig. Ex. 2.3.20(b) : A DFA

Example 2.3.21

Construct a NFA and then equivalent DFA accepting strings over $\{0, 1\}$, for accepting all possible strings of zeros and ones which do not contain 011 as substring.

Solution :

This problem is fully deterministic in nature. Technically, a DFA with ϕ transitions not mentioned can be called as NFA.

- State q_1 is for previous character as 0.
- State q_2 is for previous two characters as 01.
- An input of 1 in state q_2 will cause the machine to enter a failure state.
- NFA can be converted into a DFA by introducing an explicit failure state.

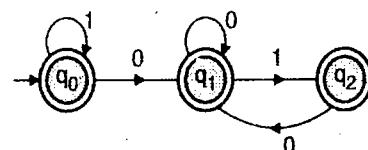


Fig. Ex. 2.3.21(a) : A NFA

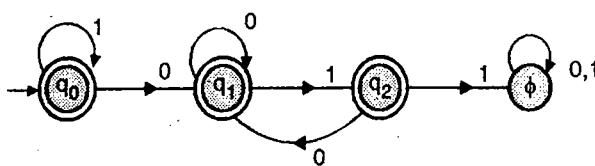


Fig. Ex. 2.3.21(b) : An equivalent DFA

Example 2.3.22 SPPU - Dec. 2012. 8 Marks

Design an NFA to accept set of all strings which end with 00. Where $I = \{0, 1\}$. Convert this NFA into its equivalent DFA.

Solution :

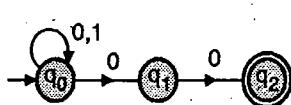
1. NFA accepting strings ending in 00

Fig. Ex. 2.3.22(a)

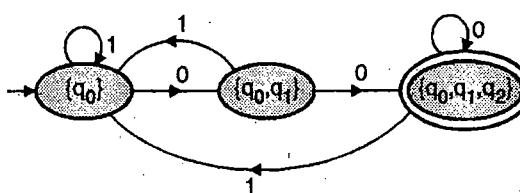
2. DFA from NFA (using direct method)

Fig. Ex. 2.3.22 (b)

Example 2.3.23

Construct a DFA equivalent to the NFA shown in Fig. Ex. 2.3.23 :

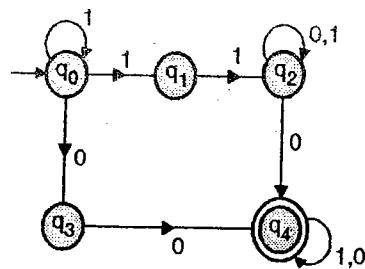


Fig. Ex. 2.3.23

Solution : Construction of subsets

Step 1 : $\delta(\{q_0\}, 0) = \{q_3\}$

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

Step 2 : $\delta(\{q_3\}, 0) = \{q_4\}$

$$\delta(\{q_3\}, 1) = \emptyset$$

$$\delta(\{q_0, q_1\}, 0) = \{q_3\}$$

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

Step 3 : $\delta(\{q_4\}, 1) = \{q_4\}$

$$\delta(\{q_4\}, 0) = \{q_4\}$$

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

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

Step 4 : $\delta(\{q_2, q_3, q_4\}, 0) = \{q_2, q_4\}$

$$\delta(\{q_2, q_3, q_4\}, 1) = \{q_2, q_4\}$$

Step 5 : $\delta(\{q_2, q_4\}, 0) = \{q_2, q_4\}$

$$\delta(\{q_2, q_4\}, 1) = \{q_2, q_4\}$$

DFA is given in Fig. Ex. 2.3.23(a).

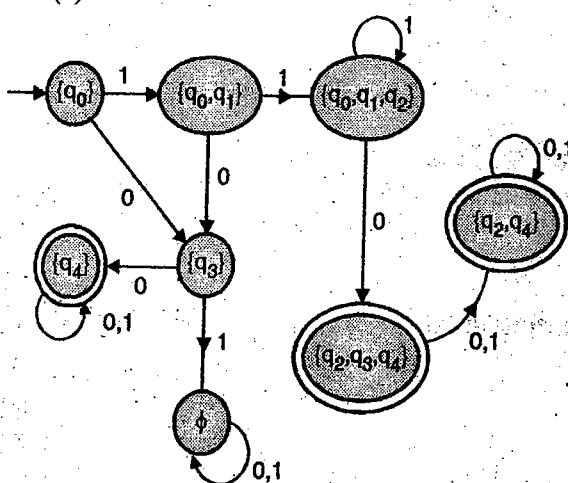


Fig. Ex. 2.3.23(a) : State diagram of the DFA for Example 2.3.23

Example 2.3.24

Let $m = (\{q_1, q_3, q_4\}, \{0, 1\}, \delta, \{q_1\}, \{q_3\})$ is NFA where δ is given as :

$$\delta(q_1, 0) = \{q_2, q_3\}$$

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



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

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

$$\delta(q_3, 0) = \{q_2\}$$

$$\delta(q_3, 1) = \{q_1, q_2\}$$

Construct the transition diagram corresponding to NFA and find and draw its equivalent DFA, show all intermediate steps also.

Solution : Transition diagram is given in Fig. Ex. 2.3.24(a).

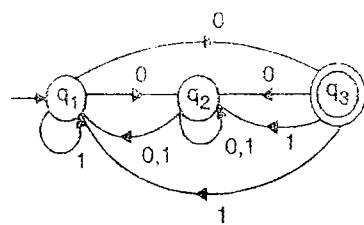


Fig. Ex. 2.3.24(a) : Transition of the given NFA

Subset construction

Step 1 :

$$\delta(\{q_1\}, 0) = \{q_2, q_3\}$$

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

Step 2 :

$$\delta(\{q_2, q_3\}, 0) = \{q_1, q_2\}$$

$$\delta(\{q_2, q_3\}, 1) = \{q_1, q_2\}$$

Step 3 :

$$\delta(\{q_1, q_2\}, 0) = \{q_1, q_2, q_3\}$$

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

Step 4 :

$$\delta(\{q_1, q_2, q_3\}, 0) = \{q_1, q_2, q_3\}$$

$$\delta(\{q_1, q_2, q_3\}, 1) = \{q_1, q_2\}$$

Equivalent DFA is given in Fig. Ex. 2.3.24(b).

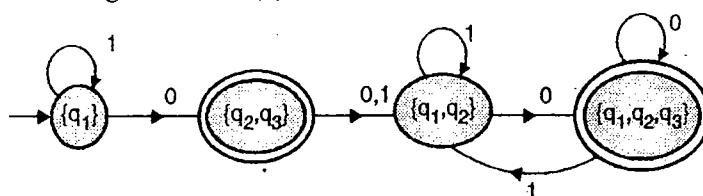


Fig. Ex. 2.3.24(b) : Equivalent DFA

Example 2.3.25 : M ($\{q_1, q_2, q_3\}$, $\{0, 1\}$, δ , q_1 , $\{q_3\}$) is a non-deterministic finite automaton where δ is given by.

$$\delta(q_1, 0) = \{q_2, q_3\} \quad \delta(q_1, 1) = \{q_1\}$$

$$\delta(q_2, 0) = \{q_1, q_2\} \quad \delta(q_2, 1) = \emptyset$$

$$\delta(q_3, 0) = \{q_2\} \quad \delta(q_3, 1) = \{q_1, q_2\}$$

Construct an equivalent DFA.

Solution : Subset construction

Step 1 :

$$\delta(\{q_1\}, 0) = \{q_2, q_3\}$$

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

Step 2 :

$$\delta(\{q_2, q_3\}, 0) = \{q_1, q_2\}$$

$$\delta(\{q_2, q_3\}, 1) = \{q_1, q_2\}$$

Step 3 :

$$\delta(\{q_1, q_2\}, 0) = \{q_1, q_2, q_3\}$$

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

$$\delta(\{q_1, q_2, q_3\}, 0) = \{q_1, q_2, q_3\}$$

$$\delta(\{q_1, q_2, q_3\}, 1) = \{q_1, q_2\}$$

DFA is given in Fig. Ex. 2.3.25.

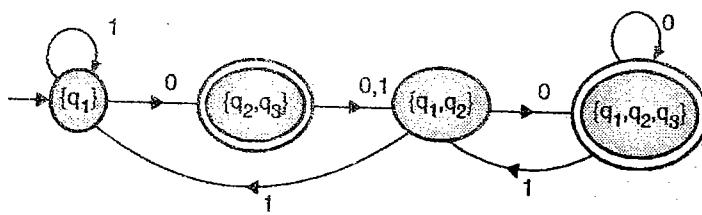


Fig. Ex. 2.3.25 : Equivalent DFA

Example 2.3.26

Let $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ be an NFA with :

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

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

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

Find the corresponding DFA.

Solution :

Subset construction

Step 1 :

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

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

Step 2 :

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

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

Step 3 :

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

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

An equivalent DFA is given in Fig. Ex. 2.3.26.

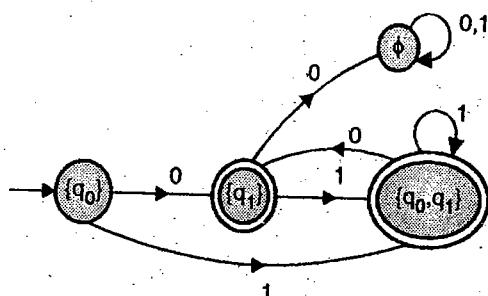


Fig. Ex. 2.3.26 : An equivalent DFA

Example 2.3.27

Convert NFA shown in Fig. Ex. 2.3.27 to its corresponding DFA.

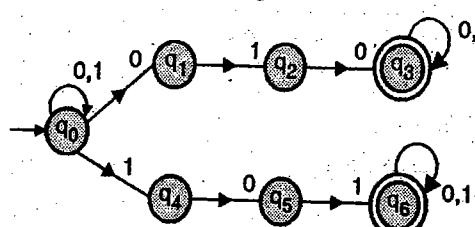


Fig. Ex. 2.3.27

Solution :

Step 1 : q_3 and q_6 have identical NFA and they can be merged into a single state q_3 .

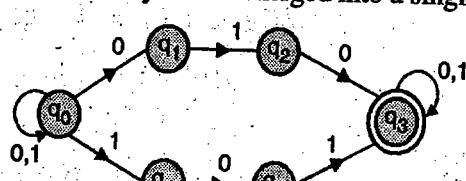


Fig. Ex. 2.3.27(a)

Step 2 :

We start with the subset $\{ q_0 \}$

0-successor of $\{ q_0 \} = \{ q_0, q_1 \}$

1-successor of $\{ q_1 \} = \{ q_0, q_4 \}$

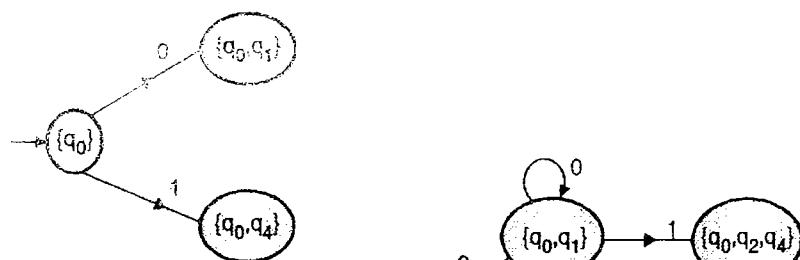


Fig. Ex. 2.3.27(b)

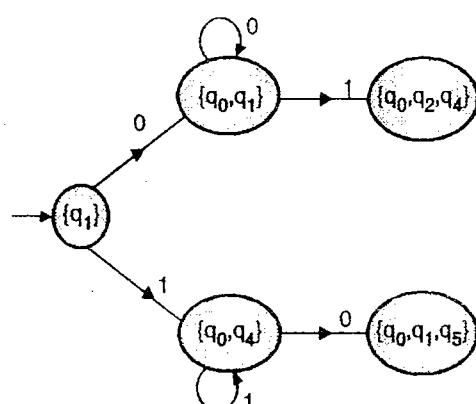
Step 3 : Two new subsets $\{ q_0, q_1 \}$ and $\{ q_0, q_4 \}$ are generated. Their successors are calculated.

Fig. Ex. 2.3.27(c)

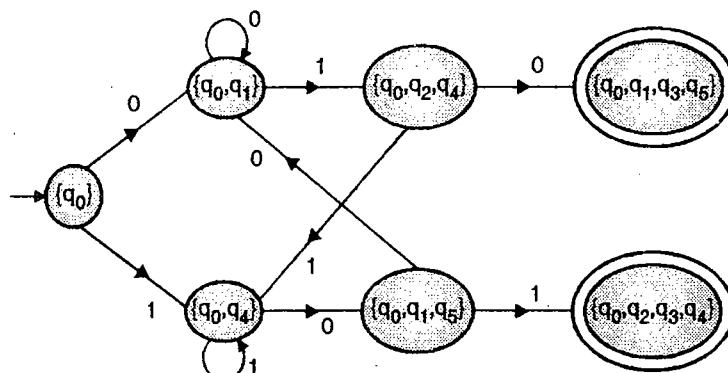
Step 4 : Two new subsets $\{ q_0, q_2, q_4 \}$ and $\{ q_0, q_1, q_5 \}$ are generated. Their successors are calculated.

Fig. Ex. 2.3.27(d)

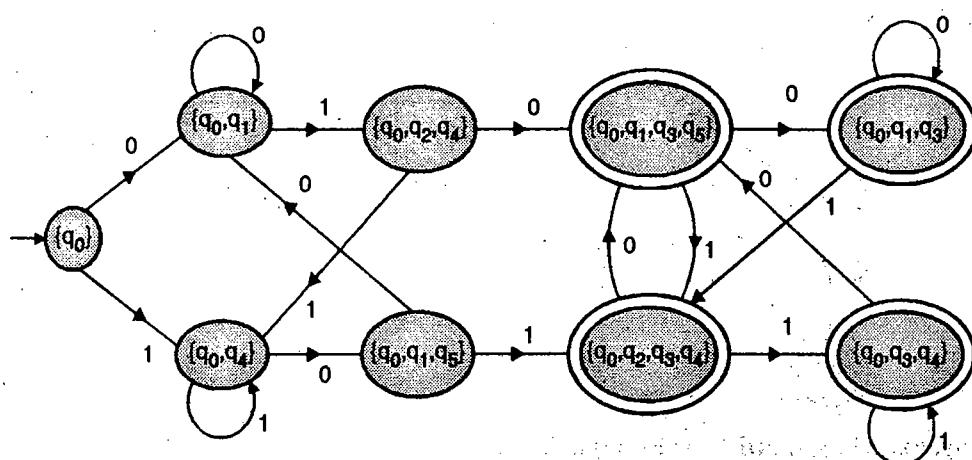
Step 5 : Two new subsets $\{ q_0, q_1, q_3, q_5 \}$ and $\{ q_0, q_2, q_3, q_4 \}$ are generated. Their successors are calculated.

Fig. Ex. 2.3.27(e)

Step 6 : Following states can be combined into a single state.

These states are $\{ q_0, q_1, q_3, q_5 \}$, $\{ q_0, q_1, q_3 \}$, $\{ q_0, q_2, q_3, q_4 \}$, $\{ q_0, q_3, q_4 \}$

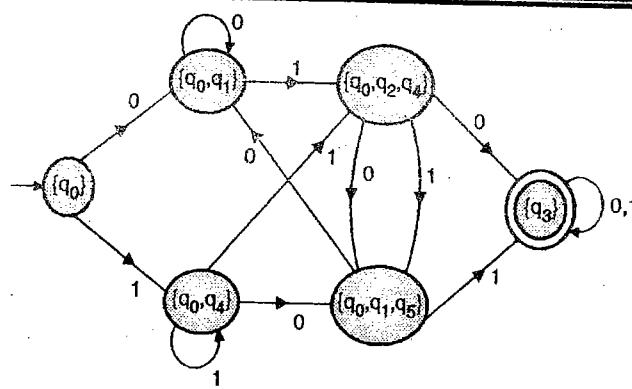


Fig. Ex. 2.3.27(f) : Final DFA

Example 2.3.28

Convert NFA shown in Fig. Ex. 2.3.28 to its corresponding DFA.

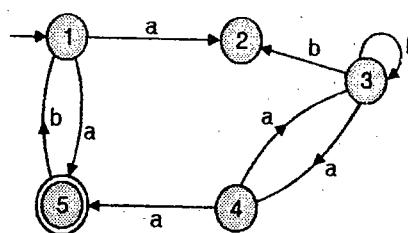


Fig. Ex. 2.3.28

Solution :

Step 1 : Starting state {1} is taken as the first subset. Its successors are calculated.

$$\delta(\{1\}, a) = \{2, 5\}; \quad \delta(\{1\}, b) = \emptyset$$

Step 2 : A new subset {2, 5} is generated. Its successors are calculated.

$$\delta(\{2, 5\}, a) = \delta(2, a) \cup \delta(5, a) = \emptyset \cup \emptyset = \emptyset$$

$$\delta(\{2, 5\}, b) = \delta(2, b) \cup \delta(5, b) = \emptyset \cup \{1\} = \{1\}$$

A subset containing 5 is marked as a final state. The equivalent DFA is given in Fig. Ex. 2.3.28(a).

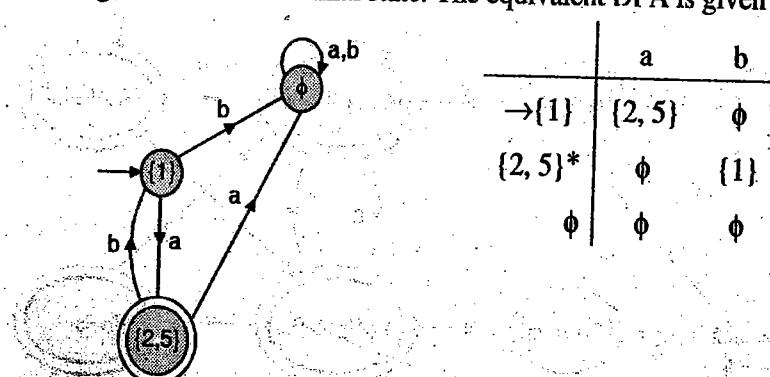


Fig. Ex. 2.3.28(a) : DFA for Example 2.3.28

Example 2.3.29

Convert the NFA shown in Fig. Ex. 2.3.29 to DFA.

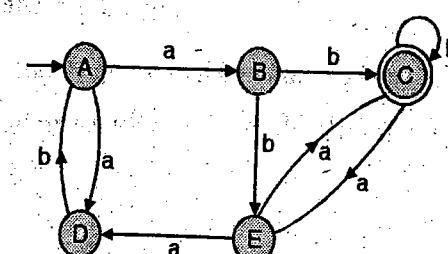


Fig. Ex. 2.3.29



Solution :

In some cases where number of states is large, it is convenient to derive subsets using a state transition table. State transition table for the given NFA is shown in Table Ex. 2.3.29(a).

Table Ex. 2.3.29 : State transition table for NFA of Example 2.3.29

| | a | b |
|-----------------|--------|--------|
| $\rightarrow A$ | {B, D} | ϕ |
| B | ϕ | {C, E} |
| C* | {E} | {C} |
| D | ϕ | {A} |
| E | {C, D} | ϕ |

Step 1 : Starting state {A} is taken as the first subset. Its successors are calculated.

$$\delta(\{A\}, a) = \{B, D\}$$

$$\delta(\{A\}, b) = \phi$$

Step 2 : A new subset {B, D} is generated. Its successors are calculated

$$\delta(\{B, D\}, a) = \delta(B, a) \cup \delta(D, a) = \phi \cup \phi = \phi$$

$$\delta(\{B, D\}, b) = \delta(B, b) \cup \delta(D, b) = \{C, E\} \cup \{A\} = \{A, C, E\}$$

Step 3 : A new subset {A, C, E} is generated. Its successors are calculated.

$$\begin{aligned}\delta(\{A, C, E\}, a) &= \delta(A, a) \cup \delta(C, a) \cup \delta(E, a) \\ &= \{B, D\} \cup \{E\} \cup \{C, D\} = \{B, C, D, E\}\end{aligned}$$

$$\begin{aligned}\delta(\{A, C, E\}, b) &= \delta(A, b) \cup \delta(C, b) \cup \delta(E, b) \\ &= \phi \cup \{C\} \cup \phi = \{C\}\end{aligned}$$

Step 4 : Two new subsets {B, C, D, E} and {C} are generated. Their successors are calculated.

$$\begin{aligned}\delta(\{B, C, D, E\}, a) &= \delta(B, a) \cup \delta(C, a) \cup \delta(D, a) \cup \delta(E, a) \\ &= \phi \cup \{E\} \cup \phi \cup \{C, D\} = \{C, D, E\}\end{aligned}$$

$$\begin{aligned}\delta(\{B, C, D, E\}, b) &= \delta(B, b) \cup \delta(C, b) \cup \delta(D, b) \cup \delta(E, b) \\ &= \{C, E\} \cup \{C\} \cup \{A\} \cup \phi = \{A, C, E\}\end{aligned}$$

$$\delta(\{C\}, a) = \{E\}$$

$$\delta(\{C\}, b) = \{C\}$$

Step 5 : Two new subsets {C, D, E} and {E} are generated. Their successors are calculated.

$$\delta(\{C, D, E\}, a) = \{E\} \cup \phi \cup \{C, D\} = \{C, D, E\}$$

$$\delta(\{C, D, E\}, b) = \{C\} \cup \{A\} \cup \phi = \{A, C\}$$

$$\delta(\{E\}, a) = \{C, D\}$$

$$\delta(\{E\}, b) = \phi$$

Step 6 : Two new subsets {A, C} and {C, D} are generated. Their successors are calculated.

$$\delta(\{C, D\}, a) = \{E\} \cup \phi = \{E\}$$

$$\delta(\{C, D\}, b) = \{C\} \cup \{A\} = \{A, C\}$$

$$\delta(\{A, C\}, a) = \{B, D\} \cup \{E\} = \{B, D, E\}$$

$$\delta(\{A, C\}, b) = \phi \cup \{C\} = \{C\}$$

Step 7 : A new subset $\{B, D, E\}$ is generated. Its successors are calculated.

$$\delta(\{B, D, E\}, a) = \phi \cup \phi \cup \{C, D\} = \{C, D\}$$

$$\delta(\{B, D, E\}, b) = \{C, E\} \cup \{A\} \cup \phi = \{A, C, E\}$$

Every subset containing C is marked as final state. Final DFA is given in Fig. Ex. 2.3.29(a).

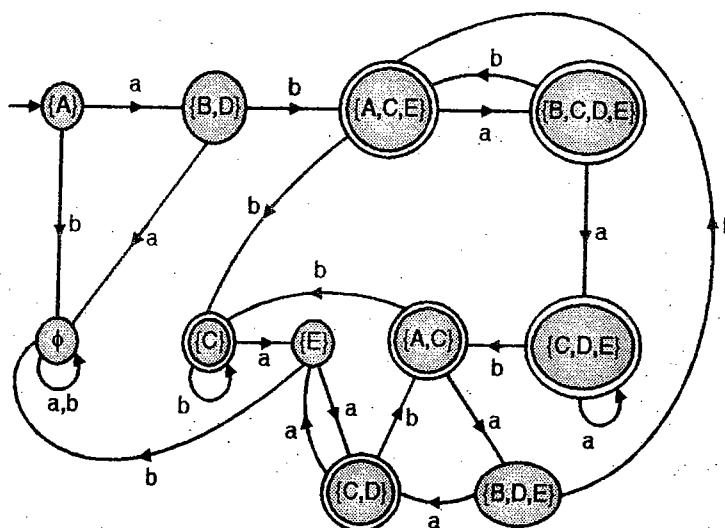


Fig. Ex. 2.3.29(a) : Final DFA for Example 2.3.29

Example 2.3.30

Convert the following NFA to its equivalent DFA.

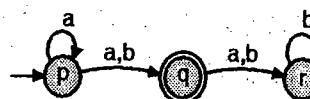


Fig. Ex. 2.3.30

Solution :

The state r is a dead state and it can be removed.

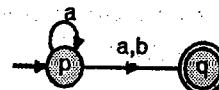


Fig. Ex. 2.3.30(a)

NFA to DFA

Step 1 : Writing transition from the state p.

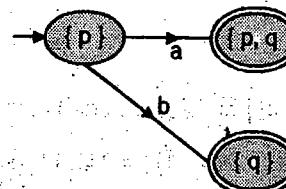


Fig. Ex. 2.3.30(b)

Step 2 : Writing transitions from the states $\{p, q\}$ and $\{q\}$.

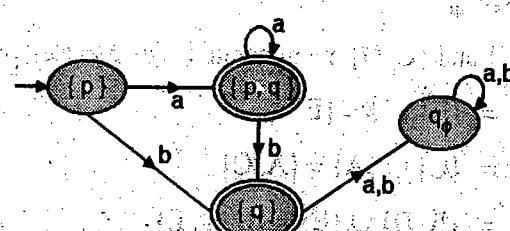


Fig. Ex. 2.3.30(c)

$$\begin{aligned}\delta(\{q\}, a) &= \phi, \delta(\{q\}, b) = \phi \\ \delta(\{p, q\}, a) &= \delta(p, a) \cup \delta(q, a) = \{p, q\} \cup \phi = \{p, q\} \\ \delta(\{p, q\}, b) &= \delta(p, b) \cup \delta(q, b) = \{q\} \cup \phi = \{q\}\end{aligned}$$

Example 2.3.31

Convert the following NFA into equivalent DFA

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

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

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

Solution : The state transition table for the above NFA is as given below :

| | 0 | 1 |
|-------------------|----------------|----------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_1\}$ |
| q_1^* | - | $\{q_0, q_1\}$ |

Step 1 : Transitions from the starting state q_0 .

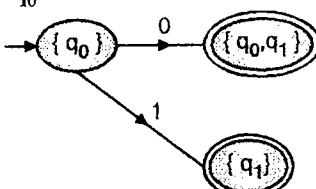


Fig. Ex. 2.3.31

Step 2 : Transitions from q_1 .

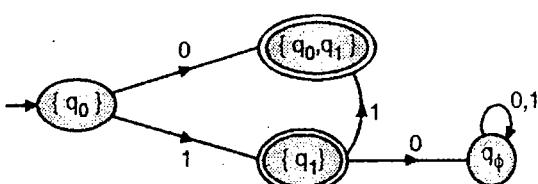


Fig. Ex. 2.3.31(a)

Step 3 : Transitions from $\{q_0, q_1\}$

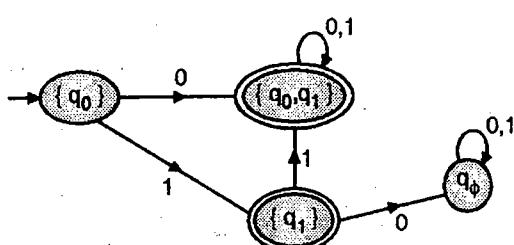


Fig. Ex. 2.3.31(b)

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

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

Example 2.3.32 SPPU - May 15, 6 Marks

Construct NFA accepting language represented by $0^*1^*2^*$ and convert it into DFA.

Solution :

ϵ - NFA for $0^*1^*2^*$ is given Fig. Ex. 2.3.32(A)(a)

Removing ϵ -transitions.

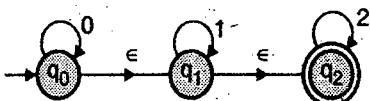


Fig. Ex. 2.3.32(A)(a)

Step 1 : Transitions from q_2 are also duplicated on q_1 . q_1 is made a final state.

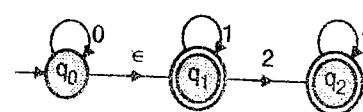


Fig. Ex. 2.3.32(A)(b)

Step 2 : Transitions from q_1 are duplicated on q_0 . q_0 is made a final state.

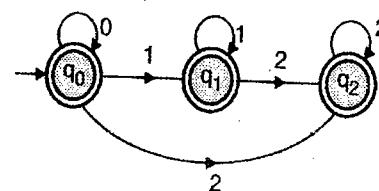


Fig. Ex. 2.3.32(c)

We can obtain a DFA by adding the failure state.

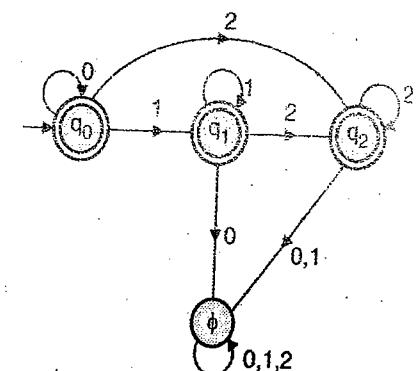


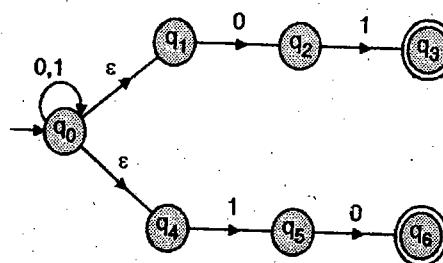
Fig. Ex. 2.3.32(d)

Syllabus Topic : NFA with Epsilon Moves

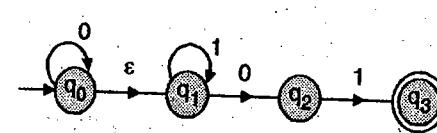
2.3.4 NFA with ϵ -Transitions

ϵ stands for a null symbol. A ϵ transition allows transition on ϵ (or no input), the empty string. This implies that a machine can make a transition without any input.

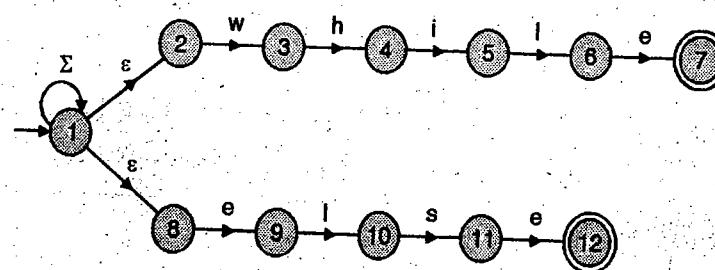
- When finding the string described by a path containing arc with label ϵ , the ϵ symbols are discarded.
- The use of ϵ -transition may simplify a transition graph by reducing number of labelled arcs.



(a) Strings ending in 01 or 10



(b) Starting with zero or more 0's followed by zero or more 1's and ending in 01



(c) Recognizing reverse words 'while' or 'else'

Fig. 2.3.12 : Few sample ϵ -NFAs

Syllabus Topic : Conversion of NFA with Epsilon moves to NFA

2.3.4.1 Equivalence of ϵ -NFA and NFA

An NFA with ϵ -move can be converted into an equivalent NFA without ϵ -move. We can always find the equivalence between systems with ϵ -move and without ϵ -move. This can be understood with the help of an example.

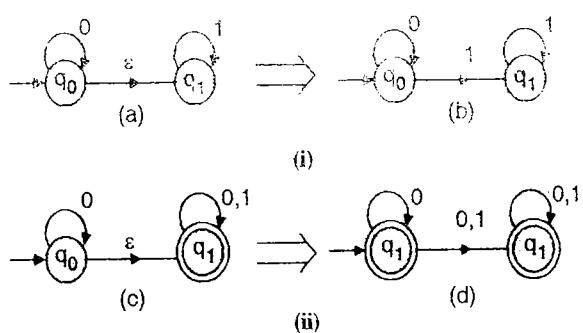


Fig. 2.3.13 : NFA with ϵ -move converted to NFA without ϵ -move

If there is an ϵ -move between any two states q_i and q_j , then ϵ -move can be removed as follows :

- (1) All moves from q_j should also originate from q_i . For example : If there is move from q_j to q_k , then we must add a move on q_i to q_k .
- (2) If q_j is a final state then make q_i as a final state.

Example 2.3.33 SPPU - May 12, 8 Marks

Use direct method to find an equivalent NFA without ϵ -move for the NFA given below :

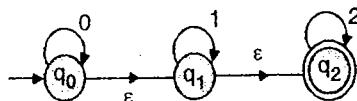


Fig. Ex. 2.3.33 : NFA with ϵ -move

Solution :

Step 1 : Transitions from q_2 are also duplicated on q_1 . q_1 is made a final state.

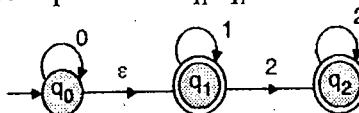


Fig. Ex. 2.3.33 (a) : ϵ -move between q_1 and q_2 is removed

Step 2 : Transition from q_1 are also duplicated on q_0 . q_0 is made a final state.

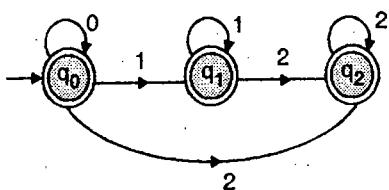


Fig. Ex. 2.3.33 (b) : Final NFA without ϵ -move after removing ϵ -move between q_0 and q_1

Example 2.3.34 SPPU – Aug. 15, 6 Marks

Convert the given NFA- ϵ to an NFA.

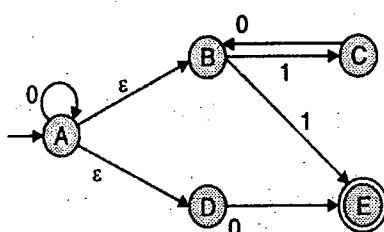


Fig. Ex. 2.3.34(a)

**Solution :**

- All moves from B will also originate from A as $\delta(A, \epsilon) = B$
 - All moves from D will also originate from A as $\delta(A, \epsilon) = D$.
- \therefore The required NFA is :

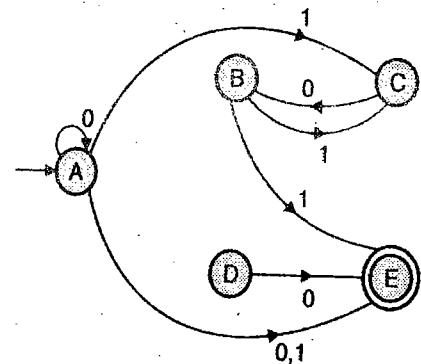


Fig. Ex. 2.3.34 (b)

The state D can be removed as it is not accessible.

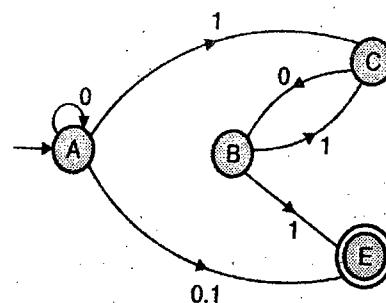


Fig. Ex. 2.3.34 (b)

2.3.4.2 The Formal Notation for an ϵ -NFA

SPPU - Dec. 15

University Question

Q. Write formal definition of NFA - ϵ .

(Dec. 2015, 3 Marks)

A non deterministic finite automaton M with ϵ -transition is given by :

Where, Q = A finite set of states

Σ = is an alphabet

$q_0 = q_0 \in Q$ is the initial/start state.

$F = F \subseteq Q$ is a set of final/accepting states

$\delta = A$ transition function from $Q \times (\Sigma \cup \{\epsilon\})$ to the power set of Q i.e. to 2^Q .

The transition function δ also includes transitions on ϵ . The NFA given in Fig. 2.3.14(a) can be formally represented as :

$$M = ((q_1, q_2, q_3), \{1, 2, 3\}, \delta, q_1, \{q_3\})$$

where δ is defined by the transition table/diagram in Fig. 2.3.14(b).

| | ϵ | 1 | 2 | 3 |
|-------------------|-------------|-------------|-------------|-------------|
| $\rightarrow q_1$ | $\{q_2\}$ | $\{q_1\}$ | \emptyset | \emptyset |
| q_2 | $\{q_3\}$ | \emptyset | $\{q_2\}$ | \emptyset |
| q_3^* | \emptyset | \emptyset | \emptyset | $\{q_3\}$ |

Fig. 2.3.14 : An NFA used for formal notation

2.3.4.3 ϵ -Closures

SPPU - Dec. 14, Dec. 15

University Question

Q. Write the formal definition of ϵ -closure
a. Define ϵ -closure.(Dec. 2014, 2 Marks)
(Dec. 2015, 2 Marks)



ϵ -closure of a state q_i is the set of states including q_i where q_i can reach by any number of ϵ -moves of the given non-deterministic finite automata.

ϵ -closure of q_i

- ϵ -closure of a state q_i , includes q_i .
- Set of states reachable from q_i on ϵ -move.
- Set of states reachable from existing states in ϵ -closure, using ϵ -move, and so on.

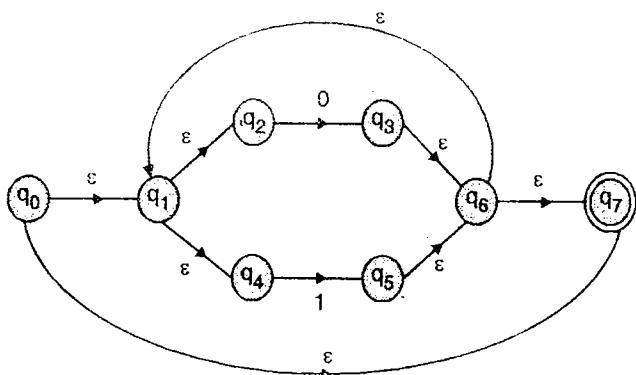


Fig. 2.3.15 : NFA considered for calculation of ϵ -closures

ϵ -closures of various states in Fig. 2.3.15 are given below :

ϵ -closure of $q_0 = \{q_0, q_1, q_2, q_4, q_7\}$

There are ϵ -moves from q_0 to q_1 ,

q_0 to q_7 ,

q_1 to q_2 ,

q_1 to q_4 .

ϵ -closure of $q_1 = \{q_1, q_2, q_4\}$

There are ϵ -moves from q_1 to q_4 ,

q_1 to q_2 .

ϵ -closure of $q_2 = \{q_2\}$

There is no ϵ -move from q_2 .

ϵ -closure of $q_3 = \{q_3, q_6, q_7, q_1, q_2, q_4\}$

There are ϵ -moves from q_3 to q_6 ,

q_6 to q_7 ,

q_6 to q_1 ,

q_1 to q_2 ,

q_1 to q_4 .

ϵ -closure of $q_4 = \{q_4\}$

There is no ϵ -move from q_4 .

ϵ -closure of $q_5 = \{q_5, q_6, q_7, q_1, q_2, q_4\}$

There are ϵ -moves from q_5 to q_6 ,

q_6 to q_7 ,

q_6 to q_1 ,

q_1 to q_2 ,

q_1 to q_4 .

ϵ -closure of $q_6 = \{q_7, q_1, q_2, q_4\}$

There are ϵ -moves from q_6 to q_7 ,

q_6 to q_1 ,

q_1 to q_2 ,

q_1 to q_4 .

ϵ -closure of $q_7 = \{q_7\}$

There is no ϵ -move from q_7 .

ϵ -closure of various states are summarised below :

| State | ϵ -closure |
|-------|------------------------------------|
| q_0 | $\{q_0, q_1, q_2, q_4, q_7\}$ |
| q_1 | $\{q_1, q_2, q_4\}$ |
| q_2 | $\{q_2\}$ |
| q_3 | $\{q_3, q_6, q_7, q_1, q_2, q_4\}$ |
| q_4 | $\{q_4\}$ |
| q_5 | $\{q_5, q_6, q_7, q_1, q_2, q_4\}$ |
| q_6 | $\{q_7, q_1, q_2, q_4\}$ |
| q_7 | $\{q_7\}$ |

Example 2.3.35

A transition table is given for another with NULL with seven states.

| q | $\delta(q, a)$ | $\delta(q, b)$ | $\delta(q, \epsilon)$ |
|---|----------------|----------------|-----------------------|
| 1 | {5} | \emptyset | {4} |
| 2 | {1} | \emptyset | \emptyset |
| 3 | \emptyset | {2} | \emptyset |
| 4 | \emptyset | {7} | {3} |
| 5 | \emptyset | \emptyset | {1} |
| 6 | \emptyset | {5} | {4} |
| 7 | {6} | \emptyset | \emptyset |

- (a) Draw a transition diagram. (b) Calculate $\delta^*(1, ba)$

Solution :

- (a) Transition diagram is in Fig. Ex. 2.3.35(a).

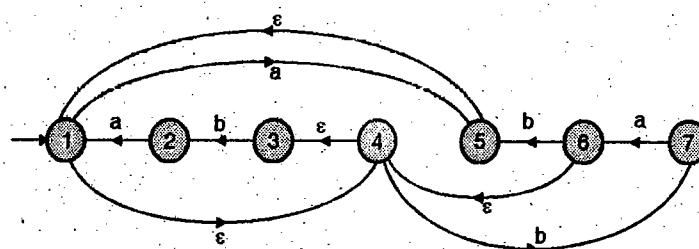


Fig. Ex. 2.3.35(a) : State transition diagram

- (b) Calculate $\delta^*(1, ba)$

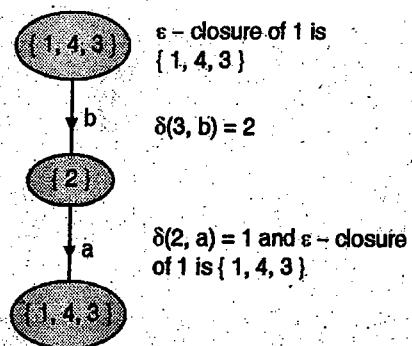


Fig. Ex. 2.3.35(b)

$$\therefore \delta^*(1, ba) = \{1, 4, 3\}$$

**Syllabus Topic : Conversion of NFA With Epsilon Moves to DFA****2.3.4.4 ϵ -NFA to DFA**

ϵ -NFA to DFA conversion is also based on subset construction as in case of NFA to DFA. ϵ -closure is calculated for every state in the subset. That is the subset is extended by ϵ -closure of every state in the subset.

Let us compute the extended transition (δ_ϵ) for state q_2 on input 0 for the ϵ -NFA in Fig. 2.3.15.

$$\delta_\epsilon(q_2, 0) = \epsilon\text{-closure}(q_3) = \{q_3, q_6, q_7, q_1, q_2, q_4\}$$

Example 2.3.36

Find an equivalent DFA for the ϵ -NFA given in Fig. Ex. 2.3.36.

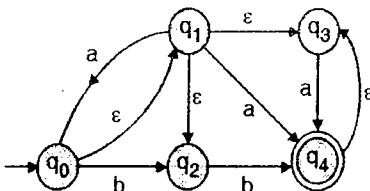


Fig. Ex. 2.3.36

Solution :**Step 1** : ϵ -closure of states.

| State | ϵ -closure | Comment |
|-------|--------------------------|---|
| q_0 | $\{q_0, q_1, q_2, q_3\}$ | There are ϵ -moves from q_0 to q_1 , q_1 to q_2 , q_1 to q_3 . |
| q_1 | $\{q_1, q_2, q_3\}$ | There are ϵ -moves from q_1 to q_2 , q_1 to q_3 |
| q_2 | $\{q_2\}$ | There is no ϵ -move from q_2 . |
| q_3 | $\{q_3\}$ | There is no ϵ -move from q_3 . |
| q_4 | $\{q_4, q_3\}$ | There is an ϵ -move from q_4 to q_3 . |

Step 2 : ϵ -closure of the initial state q_0 is taken as the first subset.

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_3\}$$

$$\begin{aligned}
 \text{a-successor of } \{q_0, q_1, q_2, q_3\} &= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a)) \\
 &= \epsilon\text{-closure}(\emptyset \cup \{q_0, q_4\} \cup \emptyset \cup \{q_4\}) \\
 &= \epsilon\text{-closure}(\{q_0, q_4\}) = \epsilon\text{-closure}\{q_0\} \cup \epsilon\text{-closure}\{q_4\} \\
 &= \{q_0, q_1, q_2, q_3\} \cup \{q_3, q_4\} = \{q_0, q_1, q_2, q_3, q_4\}
 \end{aligned}$$

$$\begin{aligned}
 \text{b-successor of } \{q_0, q_1, q_2, q_3\} &= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b)) \\
 &= \epsilon\text{-closure}(\{q_2\} \cup \emptyset \cup \{q_4\} \cup \emptyset) \\
 &= \epsilon\text{-closure}(\{q_2, q_4\}) = \epsilon\text{-closure}\{q_2\} \cup \epsilon\text{-closure}\{q_4\} \\
 &= \{q_2\} \cup \{q_3, q_4\} = \{q_2, q_3, q_4\}
 \end{aligned}$$

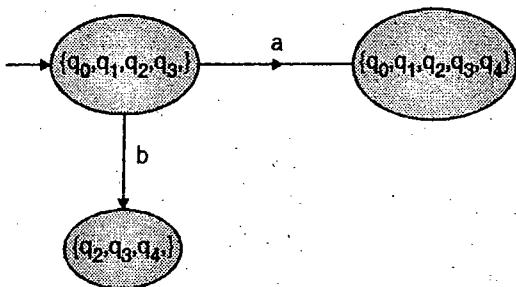


Fig. Ex. 2.3.36(a)

Step 3 : Two new subsets $\{q_2, q_3, q_4\}$ and $\{q_0, q_1, q_2, q_3, q_4\}$ have been generated. Their successors are calculated.

$$\begin{aligned} \text{a-successor of } \{q_2, q_3, q_4\} &= \varepsilon\text{-closure}(\delta(q_2, a) \cup \delta(q_3, a) \cup \delta(q_4, a)) \\ &= \varepsilon\text{-closure}(\emptyset \cup \{q_4\} \cup \emptyset) = \varepsilon\text{-closure}(q_3, q_4) \\ &= \varepsilon\text{-closure}(q_3) \cup \varepsilon\text{-closure}(q_4) \\ &= \{q_3\} \cup \{q_3, q_4\} = \{q_3, q_4\} \end{aligned}$$

$$\begin{aligned} \text{b-successor of } \{q_2, q_3, q_4\} &= \varepsilon\text{-closure}(\delta(q_2, b) \cup \delta(q_3, b) \cup \delta(q_4, b)) \\ &= \varepsilon\text{-closure}(\{q_4\} \cup \emptyset \cup \emptyset) \\ &= \varepsilon\text{-closure}(q_4) = \{q_3, q_4\} \end{aligned}$$

$$\begin{aligned} \text{a-successor of } \{q_0, q_1, q_2, q_3, q_4\} &= \text{a-successor of } \{q_0, q_1, q_2, q_3\} \cup \text{a-successor of } \{q_4\} \\ &= \{q_0, q_1, q_2, q_3, q_4\} \cup \emptyset = \{q_0, q_1, q_2, q_3, q_4\} \end{aligned}$$

$$\begin{aligned} \text{b-successor of } \{q_0, q_1, q_2, q_3, q_4\} &= \text{b-successor of } \{q_0, q_1, q_2, q_3\} \cup \text{b-successor of } \{q_4\} \\ &= \{q_2, q_3, q_4\} \cup \emptyset = \{q_2, q_3, q_4\} \end{aligned}$$

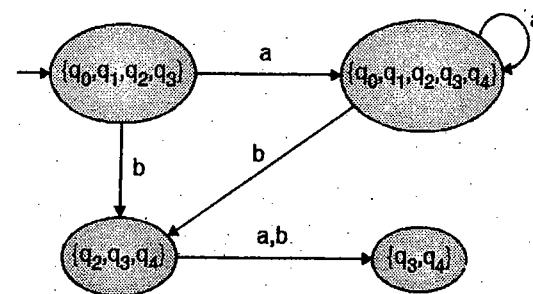


Fig. Ex. 2.3.36(b)

Step 4 : A new subset $\{q_3, q_4\}$ has been generated. Successors of $\{q_3, q_4\}$ are calculated.

$$\begin{aligned} \text{a-successor of } \{q_3, q_4\} &= \varepsilon\text{-closure}(\delta(q_3, a) \cup \delta(q_4, a)) \\ &= \varepsilon\text{-closure}(\{q_4\} \cup \emptyset) = \varepsilon\text{-closure}(q_4) = \{q_3, q_4\} \\ \text{b-successor of } \{q_3, q_4\} &= \varepsilon\text{-closure}(\delta(q_3, b) \cup \delta(q_4, b)) \\ &= \varepsilon\text{-closure}(\emptyset \cup \emptyset) = \emptyset \end{aligned}$$

No further subsets are generated. Every subset containing q_4 (final state in Fig. Ex. 2.3.36) is marked as final state.

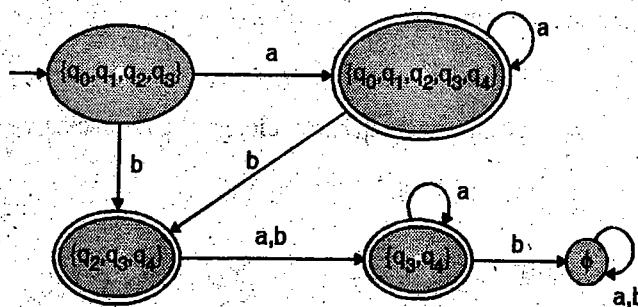


Fig. Ex. 2.3.36(c) : Final DFA

Example 2.3.37 [SPPU-JUNE/2013]

Consider the following ϵ -NFA

| | ϵ | a | B | c |
|-----------------|-------------|-----|-------------|-------------|
| $\rightarrow p$ | \emptyset | {p} | {q} | {r} |
| q | {p} | {q} | {r} | \emptyset |
| r^* | {q} | {r} | \emptyset | {p} |

- (a) Compute the ϵ -closure of each state.
- (b) Convert the automaton to a DFA.

Solution :

- (a) ϵ -closure of states

| State | ϵ -closure | Comment |
|-------|---------------------|--|
| p | {p} | There is no ϵ -move from p |
| q | {p, q} | There is an ϵ -move from q to p. |
| r | {p, q, r} | There are ϵ -moves from r to q, q to p. |

- (b) Conversion of NFA to DFA

Step 1 : ϵ -closure of the initial state p is taken as the first subset

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

$$\text{a-successor of } \{p\} = \epsilon\text{-closure}(\delta(p, a)) = \epsilon\text{-closure}(p) = p$$

$$\text{b-successor of } \{p\} = \epsilon\text{-closure}(\delta(p, b)) = \epsilon\text{-closure}(q) = \{p, q\}$$

$$\begin{aligned}\text{c-successor of } \{p\} &= \epsilon\text{-closure}(\delta(p, c)) \\ &= \epsilon\text{-closure}(r) = \{p, q, r\}\end{aligned}$$

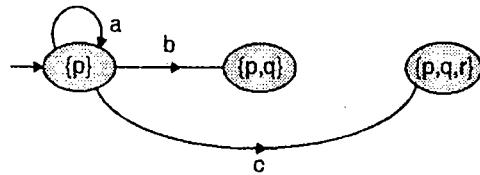


Fig. Ex. 2.3.37

Step 2 : Two new subsets $\{p, q\}$ and $\{p, q, r\}$ are generated. Their successors are calculated.

$$\begin{aligned}\text{a-successor of } \{p, q\} &= \epsilon\text{-closure}(\delta(p, a) \cup \delta(q, a)) = \epsilon\text{-closure}(\{p\} \cup \{q\}) \\ &= \epsilon\text{-closure}(\{p, q\}) = \epsilon\text{-closure}(p) \cup \epsilon\text{-closure}(q) \\ &= \{p\} \cup \{p, q\} = \{p, q\}\end{aligned}$$

$$\begin{aligned}\text{b-successor of } \{p, q\} &= \epsilon\text{-closure}(\delta(p, b) \cup \delta(q, b)) = \epsilon\text{-closure}(\{q\} \cup \{r\}) \\ &= \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r) \\ &= \{p, q\} \cup \{p, q, r\} = \{p, q, r\}\end{aligned}$$

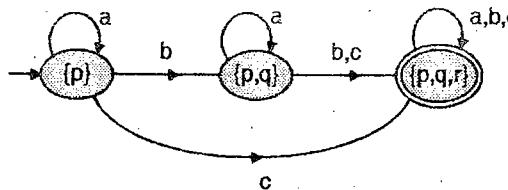
$$\begin{aligned}\text{c-successor of } \{p, q\} &= \epsilon\text{-closure}(\delta(p, c) \cup \delta(q, c)) \\ &= \epsilon\text{-closure}(\{r\} \cup \emptyset) = \{p, q, r\}\end{aligned}$$

$$\begin{aligned}\text{a-successor of } \{p, q, r\} &= \text{a-successor of } \{p, q\} \cup \text{a-successor of } \{r\} \\ &= \{p, q\} \cup \epsilon\text{-closure}(r) \\ &= \{p, q\} \cup \{p, q, r\} = \{p, q, r\}\end{aligned}$$

$$\text{b-successor of } \{p, q, r\} = \text{b-successor of } \{p, q\} \cup \text{b-successor of } \{r\} = \{p, q, r\}$$

$$\begin{aligned}\text{c-successor of } \{p, q, r\} &= \text{c-successor of } \{p, q\} \cup \text{c-successor of } \{r\} \\ &= \{p, q, r\} \cup \epsilon\text{-closure}(p) = \{p, q, r\}\end{aligned}$$

No new subsets are generated. Each subset containing r (final state in given (ϵ -NFA)) is marked as final state.



(a) State transition diagram of final DFA

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

(b) State transition table of final DFA

Fig. Ex. 2.3.37

Example 2.3.38

Consider the following ϵ -NFA.

| | ϵ | a | b | c |
|-----------------|-------------|-------------|-------------|-------------|
| $\rightarrow p$ | {q, r} | \emptyset | {q} | {r} |
| q | \emptyset | {p} | {r} | {p, q} |
| r* | \emptyset | \emptyset | \emptyset | \emptyset |

- (a) Compute the ϵ -closure of each state.
 (b) Convert the automation of a DFA.

Solution :

(a) ϵ -closure of states

| State | ϵ -closure | Comment |
|-------|---------------------|---|
| p | {p, q, r} | There is an ϵ -move from p to {q, r} |
| q | {q} | There is no ϵ -move from q. |
| r | {r} | There is no ϵ -move from r. |

(b) Conversion of NFA to DFA

Step 1 : ϵ -closure of the initial state p is taken as the first subset.

$$\epsilon\text{-closure}(p) = \{p, q, r\}$$

Successor of {p, q, r} are given by :

$$\begin{aligned} \text{a-successor of } \{p, q, r\} &= \epsilon\text{-closure}(\delta(p, q, r), a) \\ &= \epsilon\text{-closure}(\delta(p, a) \cup \delta(q, a) \cup \delta(r, a)) \\ &= \epsilon\text{-closure}(\emptyset \cup \{p\} \cup \emptyset) = \epsilon\text{-closure}(\{p\}) = \{p, q, r\} \end{aligned}$$

$$\begin{aligned} \text{b-successor of } \{p, q, r\} &= \epsilon\text{-closure}[\delta(\{p, q, r\}, b)] \\ &= \epsilon\text{-closure}(\delta(p, b) \cup \delta(q, b) \cup \delta(r, b)) \\ &= \epsilon\text{-closure}(\{q\} \cup \{r\} \cup \emptyset) = \epsilon\text{-closure}(\{q, r\}) \\ &= \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r) = \{q\} \cup \{r\} = \{q, r\} \end{aligned}$$

$$\begin{aligned} \text{c-successors of } \{p, q, r\} &= \epsilon\text{-closure}(\delta(\{p, q, r\}, c)) \\ &= \epsilon\text{-closure}(\delta(p, c) \cup \delta(q, c) \cup \delta(r, c)) \\ &= \epsilon\text{-closure}(\{r\} \cup \{p, q\} \cup \emptyset) = \epsilon\text{-closure}(\{p, q, r\}) \\ &= \epsilon\text{-closure}(p) \cup \epsilon\text{-closure}(q) \cup \epsilon\text{-closure}(r) \\ &= \{p, q, r\} \cup \{q\} \cup \{r\} = \{p, q, r\} \end{aligned}$$

Step 2 : A new subset $\{q, r\}$ is generated and its successors are calculated.

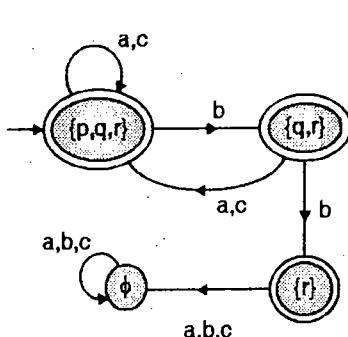
$$\begin{aligned} \text{a successor of } \{q, r\} &= \epsilon\text{-closure}(\delta(\{q, r\}, a)) \\ &= \epsilon\text{-closure}(\delta(q, a) \cup \delta(r, a)) \\ &= \epsilon\text{-closure}(\{p\} \cup \emptyset) = \epsilon\text{-closure}(p) = \{p, q, r\} \end{aligned}$$

$$\begin{aligned} \text{b-successors of } \{q, r\} &= \epsilon\text{-closure}(\delta(\{q, r\}, b)) \\ &= \epsilon\text{-closure}(\delta(q, b) \cup \delta(r, b)) \\ &= \epsilon\text{-closure}(\{r\} \cup \emptyset) = \epsilon\text{-closure}(\{r\}) = \{r\} \end{aligned}$$

$$\begin{aligned} \text{c-successor of } \{q, r\} &= \epsilon\text{-closure}(\delta(\{q, r\}, c)) = \epsilon\text{-closure}(\delta(q, c) \cup \delta(r, c)) \\ &= \epsilon\text{-closure}(\{p, q\} \cup \emptyset) = \{p, q, r\} \end{aligned}$$

Step 3 : A new subset $\{r\}$ is generated successor of $\{r\}$ are \emptyset .

Every subset containing r is marked as a final state. Final DFA is given in Fig. Ex. 2.3.38.



(a) State diagram

| | a | b | c |
|----------------------------|-------------|-------------|-------------|
| $\rightarrow\{p, q, r\}^*$ | {p, q, r} | {q, r} | {p, q, r} |
| $\{q, r\}^*$ | {p, q, r} | {r} | {p, q, r} |
| $\{r\}^*$ | \emptyset | \emptyset | \emptyset |
| \emptyset | \emptyset | \emptyset | \emptyset |

(b) State table

Fig. Ex. 2.3.38 : Final DFA for Example 2.3.38

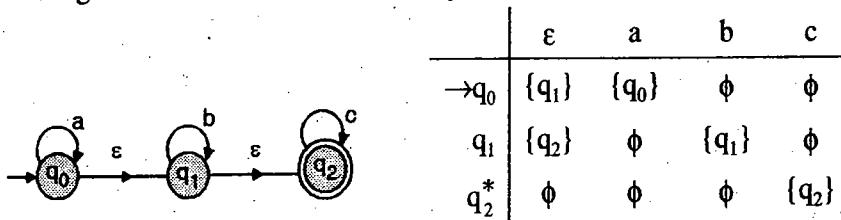
Example 2.3.39 SPPU – Dec. 14. 6 Marks

Design ϵ -NFA for the following languages. Try to use ϵ -transitions to simplify the design

- (a) The set of strings consisting of zero or more a's followed by zero or more b's, followed by zero or more c's.
- (b) The set of strings that consist of either 01 repeated zero or more times or 010 repeated zero or more times.
- (c) The set of strings of 0's and 1's such that at least one of the last ten positions is 1.

Solution :

- (a) The set of strings consisting of zero or more a's followed by zero or more b's, followed by zero or more c's.



| | ϵ | a | b | c |
|-------------------|-------------|-------------|-------------|-------------|
| $\rightarrow q_0$ | {q_1} | {q_0} | \emptyset | \emptyset |
| q_1 | {q_2} | \emptyset | {q_1} | \emptyset |
| q_2^* | \emptyset | \emptyset | \emptyset | {q_2} |

Fig. Ex. 2.3.39 : ϵ -NFA for Example 2.3.39(a)

Meaning of various states :

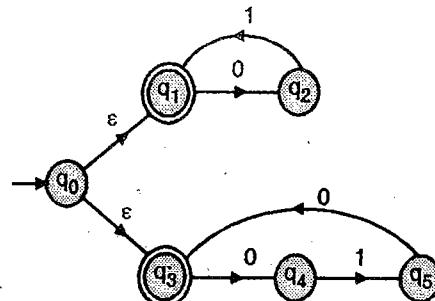
$q_0 \rightarrow q_0$ will generate zero or more a's

$q_1 \rightarrow q_1$ will generate zero or more b's

$q_2 \rightarrow q_2$ will generate zero or more c's



- (b) The set of strings that consist of either 01 repeated zero or more times or 010 repeated zero or more times.



(a) State diagram

| | ϵ | 0 | 1 |
|-------------------|----------------|-----------|-----------|
| $\rightarrow q_0$ | $\{q_1, q_3\}$ | ϕ | ϕ |
| q_1^* | ϕ | $\{q_2\}$ | ϕ |
| q_2 | ϕ | ϕ | $\{q_1\}$ |
| q_3^* | ϕ | $\{q_4\}$ | ϕ |
| q_4 | ϕ | ϕ | $\{q_5\}$ |
| q_5 | ϕ | $\{q_3\}$ | ϕ |

(b) State table

Fig. Ex. 2.3.39 : ϵ -NFA for Example 2.3.39(b)

- o Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_1 \dots q_2 \rightarrow q_1$ is for generation of zero or more representation of 01.
 - o Path $q_0 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \dots \rightarrow q_3$ is for generation of zero or more representation of 010.
- (c) The set of strings of 0's and 1's such that at least one of the last ten positions is 1.

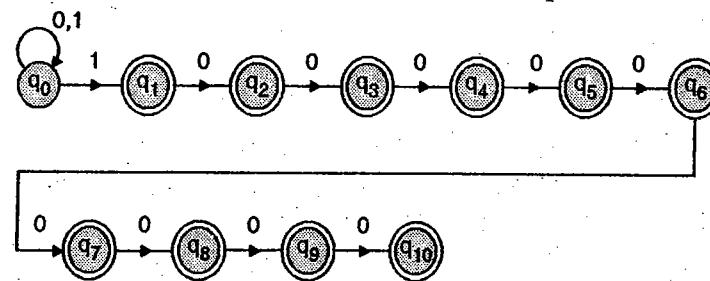


Fig. Ex. 2.3.39(c) : NFA for Example 2.3.39(c)

- o State q_1 is for accepting strings with last character is 1.
- o Similarly, states q_2 to q_{10} are for accepting strings whose second position to 10th position from the end contain 1.

Example 2.3.40

Convert the following NFA- ϵ to its equivalent DFA.

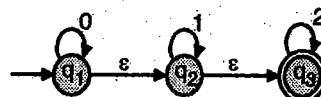


Fig. Ex. 2.3.40

Solution :

Step 1 : Finding ϵ -closure

| State | ϵ -closure |
|-------|---------------------|
| q_1 | $\{q_1, q_2, q_3\}$ |
| q_2 | $\{q_2, q_3\}$ |
| q_3 | $\{q_3\}$ |

Step 2 : Initial state is taken as the ϵ -closure of q_1 . Transitions are written from it.

$$\begin{aligned}\delta(\{q_1, q_2, q_3\}, 0) &= \text{ε-closure}(\delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)) \\ &= \text{ε-closure}(q_1 \cup \phi \cup \phi) = \{q_1, q_2, q_3\} \\ \delta(\{q_1, q_2, q_3\}, 1) &= \text{ε-closure}(\delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1))\end{aligned}$$

$$\begin{aligned}
 &= \epsilon\text{-closure}(\phi \cup q_2 \cup \phi) = \{q_2, q_3\} \\
 \delta(\{q_1, q_2, q_3\}, 2) &= \epsilon\text{-closure}(\delta(q_1, 2) \cup \delta(q_2, 2) \cup \delta(q_3, 2)) \\
 &= \epsilon\text{-closure}(\phi \cup \phi \cup q_3) = \{q_3\}
 \end{aligned}$$

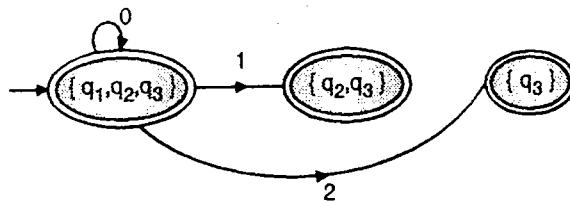


Fig. Ex. 2.3.40(a)

Step 3 : Transitions from $\{q_2, q_3\}$ and $\{q_3\}$ are calculated.

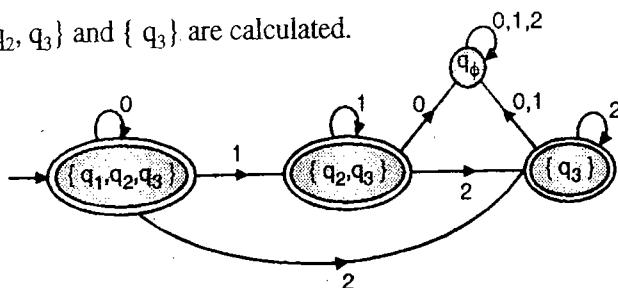


Fig. Ex. 2.3.40(b)

Example 2.3.41

Design an FSM to accept those strings having 101 or 110 as substring.

Solution : The required FSM is given below.

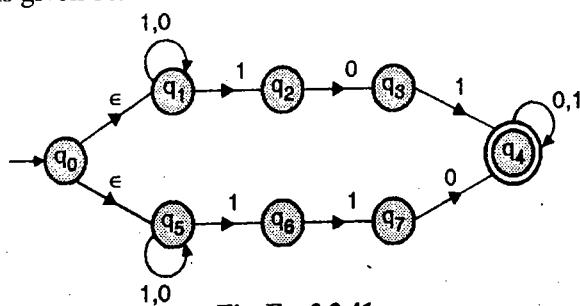


Fig. Ex. 2.3.41

- The path from $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$ is for the substring 101.
- The path from $q_0 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_4$ is for the substring 110.

2.3.5 Difference between NFA and DFA

SPPU – Dec. 12

University Question

Q. Give the difference between NFA and DFA.

(Dec. 2012, 8 Marks)

| Parameter | NFA | DFA |
|----------------|---|--|
| Transition | Non-deterministic. | Deterministic |
| No. of states. | NFA has fewer number of states. | More, if NFA contains Q states then the corresponding DFA will have $\leq 2^Q$ states. |
| Power | NFA is as powerful as a DFA | DFA is as powerful as an NFA |
| Design | Easy to design due to non-determinism. | Relatively, more difficult to design as transitions are deterministic. |
| Acceptance | It is difficult to find whether $w \in L$ as there are several paths. Backtracking is required to explore several parallel paths. | It is easy to find whether $w \in L$ as transitions are deterministic. |

2.4 Finite Automata as Output Devices

Finite automata that we have discussed so far have a limited capability of either accepting a string or rejecting a string. Acceptance of a string was based on the reachability of a machine from starting state to final state. Finite automata can also be used as an output device.

- Such machines do no have final state/states.
- Machine generates an output on every input. The value of the output is a function of current state and the current input.

Such machines are characterised by two behaviours :

1. State transition function (δ)
2. Output function (λ)

State transition function (δ) is also known as STF.

Output function (λ) is also known as machine function (MAF).

$$\delta : \Sigma \times Q \rightarrow Q$$

$$\lambda : \Sigma \times Q \rightarrow O \text{ [for Mealy machine]}$$

$$\lambda : Q \rightarrow O \text{ [for Moore machine]}$$

There are two types of automata with outputs :

1. Mealy machine : Output is associated with transition.

$$\lambda : \Sigma \times Q \rightarrow O$$

Set of output alphabet O can be different from the set of input alphabet Σ

2. Moore machine : Output is associated with state.

$$\lambda : Q \rightarrow O$$

2.4.1 A Sample Mealy Machine

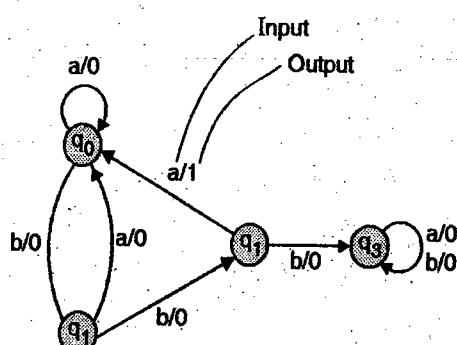


Fig. 2.4.1 : State diagram of a Mealy machine

State transition function (δ) (or STF) :

| | a | b |
|-------------------|-------|-------|
| $\rightarrow q_0$ | q_0 | q_1 |
| q_1 | q_0 | q_2 |
| q_2 | q_0 | q_3 |
| q_3 | q_3 | q_3 |

Fig. 2.4.2 : State transition function for Mealy machine of Fig. 2.4.1



Output function (λ) (or MAF) :

| | a | b |
|-------------------|---|---|
| $\rightarrow q_0$ | 0 | 0 |
| q_1 | 0 | 0 |
| q_2 | 1 | 0 |
| q_3 | 0 | 0 |

Fig. 2.4.3 : Output function for mealy machine of Fig. 2.4.1

State table for both δ and λ (both STF and MAF) :

| | A | b |
|-------------------|---------|---------|
| $\rightarrow q_0$ | $q_0/0$ | $q_1/0$ |
| q_1 | $q_0/0$ | $q_2/0$ |
| q_2 | $q_0/1$ | $q_3/0$ |
| q_3 | $q_3/0$ | $q_3/0$ |

↓ → Output
 → Next state

Fig. 2.4.4 : State table depicting both transition and output behavior of mealy machine of Fig. 2.4.1

- An arc from state q_i in a mealy machine is associated with :
 1. Input alphabet $\in \Sigma$
 2. An output alphabet $\in O$.
- An arc marked as 'a/0' in Fig. 2.4.1 implies that :
 1. a is in input
 2. 0 is an output.
- State transition behavior and output behavior of a mealy machine can be shown separately as in Fig. 2.4.2 and 2.4.3; or they can be combined together as in Fig. 2.4.4.

Syllabus Topic : Definition and Construction of Mealy Machines

2.4.2 Formal Definition of a Mealy Machine

A mealy machine M is defined as :

$$M = (Q, \Sigma, O, \delta, \lambda, q_0)$$

Where,

Q = A finite set of states.

Σ = A finite set of input alphabet

O = A finite set of output alphabet

δ = A transition function $\Sigma \times Q \rightarrow Q$

λ = An output function $\Sigma \times Q \rightarrow O$

$q_0 = q_0 \in Q$ is an initial state.

2.4.3 A Sample Moore Machine

SPPU – May 12

University Question

Q. Explain Moore's algorithm with the help of example.

(May 2012, 4 Marks)

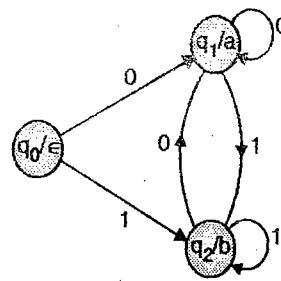


Fig. 2.4.5 : State diagram of a Moore machine

State transition function (δ) :

| | 0 | 1 | State | Output |
|-------------------|-------|-------|-------------------|------------|
| $\rightarrow q_0$ | q_1 | q_2 | $\rightarrow q_0$ | ϵ |
| q_1 | q_1 | q_2 | q_1 | a |
| q_2 | q_1 | q_2 | q_2 | b |

Fig. 2.4.6 : State transition function for Moore
machine of Fig. 2.4.5Fig. 2.4.7 : Output function for Moore machine
of Fig. 2.4.5State table for both δ and λ :

| State | Input | | Output |
|-------------------|-------|-------|------------|
| | 0 | 1 | |
| $\rightarrow q_0$ | q_1 | q_2 | ϵ |
| q_1 | q_1 | q_2 | a |
| q_2 | q_1 | q_2 | b |

Fig. 2.4.8 : State table depicting both transition and output
behavior of Moore machine of Fig. 2.4.5

- Output in a Moore machine is associated with a state and not with transition.

Syllabus Topic : Definition and Construction of Moore Machines

2.4.4 Formal Definition of a Moore Machine

SPPU – May 12

University Question

- (Q) Explain Moore's algorithm with the help of example.

(May 2012, 4 Marks)

A Moore machine M is defined as :

$$M = \{Q, \Sigma, O, \delta, \lambda, q_0\}$$

where,

Q = A finite set of states

 Σ = A finite set of input alphabet

O = A finite set of output alphabet

 δ = A transition function $\Sigma \times Q \rightarrow Q$ λ = An output function $Q \rightarrow O$ $q_0 = q_0 \in Q$ is an initial state.

Example 2.4.1

Give Mealy and Moore machine for the following :

From input Σ^* , where $\Sigma = \{0, 1, 2\}$ print the residue modulo 5 of the input treated as ternary (base 3).

Solution :

(a) Mealy machine :

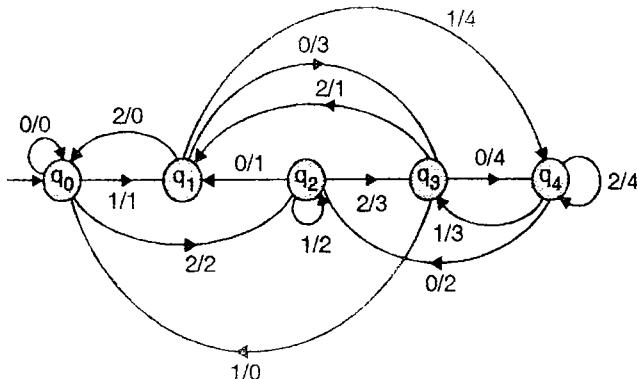


Fig. Ex. 2.4.1(a) : Mealy machine

Meaning of various states is :

q_0 – Running remainder is 0

q_1 – Running remainder is 1

q_2 – Running remainder is 2.

q_3 – Running remainder is $3 = (10)_3$

q_4 – Running remainder is $4 = (11)_3$

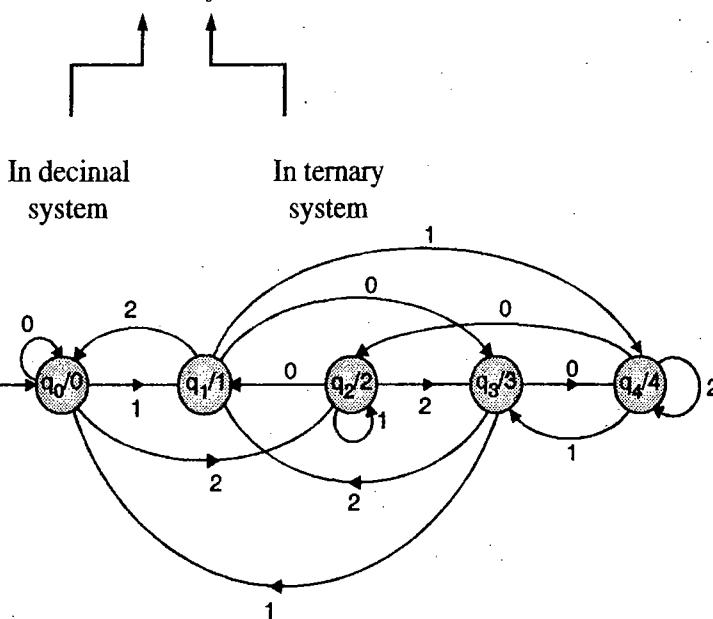


Fig. Ex. 2.4.1(b) : Moore machine

Example 2.4.2 SPPU - Dec. 12, 6 Marks

Draw the transition matrix and transition graph for a binary adder.

OR

Construct FSM for Binary Adder.

Solution : The machine will have two states :

q_0 – corresponding to previous carry as 0

q_1 – corresponding to previous carry as 1.

- Machine will transit from q_0 to q_1 if the previous carry is 0 and the next carry is 1.

- Machine will transit from q_1 to q_0 if the previous carry is 1 and the next carry is 0.
- Next input will be one of the followings :
 $\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- Sum of carry will be calculated in a usual way.

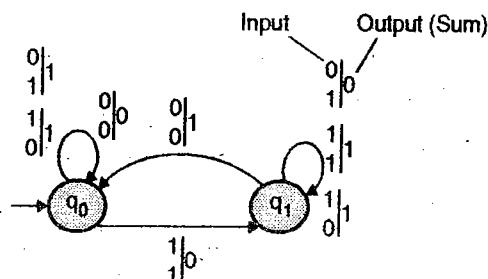


Fig. Ex. 2.4.2(a) : Transition graph

| | $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ | |
|-------|--|--|--|--|--------------|
| q_0 | $q_0, 0$ | $q_0, 1$ | $q_0, 1$ | $q_1, 0$ | ← carry is 0 |
| q_1 | $q_0, 1$ | $q_1, 0$ | $q_1, 0$ | $q_1, 1$ | ← carry is 1 |

↓ ↓ ↓ ↓
 Output Output Output Output
 ↓ ↓ ↓ ↓
 Next state Next state Next state Next state

Fig. Ex. 2.4.2(b) : Transition matrix

Example 2.4.3 : Show MAF, STF, Transition graph and Transition matrix for a divisibility by 3 tester assuming ternary input.

Solution : A divisibility by 3 tester will have three states. These states will be based on running remainder. The next remainder will be a function of current remainder and the next input. A ternary system has three alphabets.

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

q_0 = current remainder is 0

q_1 = current remainder is 1

q_2 = current remainder is 2.

Next remainder as a function of current remainder (state) and the next input is given below.

| Current state (current remainder) | Next input | Next remainder (next state) |
|-----------------------------------|------------|-----------------------------|
|-----------------------------------|------------|-----------------------------|

$0 (q_0)$

$$0 \quad 00 \bmod 10 = 0 \quad q_0$$

$$1 \quad 01 \bmod 10 = 1 \quad q_1$$

$$2 \quad 02 \bmod 10 = 2 \quad q_2$$

$$(10)_3 = (3)_{10}$$

$1 (q_1)$

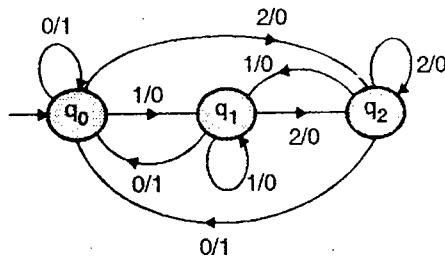
$$0 \quad 10 \bmod 10 = 0 \quad q_0$$

$$1 \quad 11 \bmod 10 = 1 \quad q_1$$

$$2 \quad 12 \bmod 10 = 2 \quad q_2$$

| Current state (current remainder) | Next input | Next remainder (next state) |
|-----------------------------------|--------------------|-----------------------------|
| 2 (q_2) | 0 20 mod 10 = 0 | q_0 |
| | 1 21 mod 10 = 1 | q_1 |
| | 2 22 mod 10 = 2 | q_2 |

Machine will give an output 1 if the current remainder is 0, output will be 0 otherwise.



(a) State transition diagram

| Current state | Input | | |
|-------------------|---------|---------|---------|
| | 0 | 1 | 2 |
| $\rightarrow q_0$ | $q_0/1$ | $q_1/0$ | $q_2/0$ |
| q_1 | $q_0/1$ | $q_1/0$ | $q_2/0$ |
| q_2 | $q_0/1$ | $q_1/0$ | $q_2/0$ |

(b) State transition table

| | 0 | 1 | 2 |
|-------------------|---|---|---|
| $\rightarrow q_0$ | 1 | 0 | 0 |
| q_1 | 1 | 0 | 0 |
| q_2 | 1 | 0 | 0 |

(c) MAF

| | 0 | 1 | 2 |
|-------------------|-------|-------|-------|
| $\rightarrow q_0$ | q_0 | q_1 | q_2 |
| q_1 | q_0 | q_1 | q_2 |
| q_2 | q_0 | q_1 | q_2 |

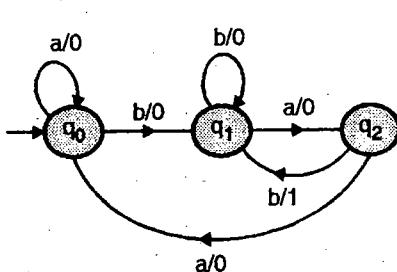
(d) STF

Fig. Ex. 2.4.3

Example 2.4.4

Design a Mealy machine that gives an output of 1 if the input string ends in bab.

Solution :



(a) State diagram

| | a | b |
|-------------------|----------|----------|
| $\rightarrow q_0$ | $q_0, 0$ | $q_1, 0$ |
| q_1 | $q_2, 0$ | $q_1, 0$ |
| q_2 | $q_0, 0$ | $q_1, 1$ |

(b) State table

Fig. Ex. 2.4.4

Meaning of various states :

q_0 – Start state

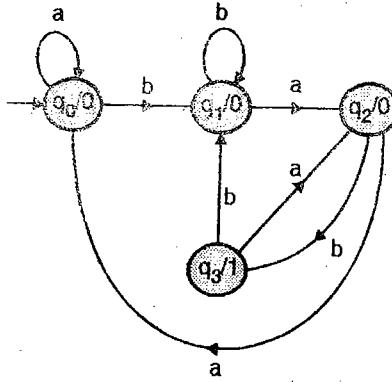
q_1 – Previous character is b

q_2 – Preceding two character are ba

A transition from q_2 to q_1 will make the preceding three characters as bab and hence the output 1.

Example 2.4.5

Design a Moore machine that gives an output of 1 if the input string ends in bab.

Solution :

(a) State diagram

| Present state | Input | | Output |
|-------------------|-------|-------|--------|
| | a | b | |
| $\rightarrow q_0$ | q_0 | q_1 | 0 |
| q_1 | q_2 | q_1 | 0 |
| q_2 | q_0 | q_3 | 0 |
| q_3 | q_2 | q_1 | 1 |

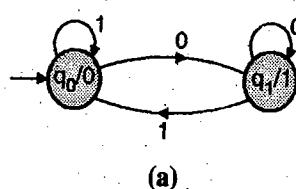
(b) State table

Fig. Ex. 2.4.5 : Moore machine for the Example 2.4.5

Meaning of various states :

 q_0 – Start state q_1 – Previous character is b q_2 – Preceding two character are ab q_3 – Previous character is bA transition from q_2 to q_3 will make the preceding three characters as bab hence the output 1.**Example 2.4.6**

Design a Moore machine for the 1's complement of binary number.

Solution :

(a)

| | 0 | 1 | Output |
|-------------------|-------|-------|--------|
| $\rightarrow q_0$ | q_1 | q_0 | 0 |
| q_1 | q_1 | q_0 | 1 |

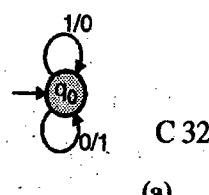
(b)

Fig. Ex. 2.4.6

- Next input as 0, sends the machine to state q_1 with output of 1. Complement of 0 is 1.
- Next input as 1, sends the machine to state q_0 with output of 0. Complement of 1 is 0.

Example 2.4.7

Design a Mealy machine to generate 1's complement of a given binary number.

Solution :

(a)

| | 0 | 1 |
|-------------------|----------|----------|
| $\rightarrow q_0$ | $q_0, 1$ | $q_0, 0$ |

(b)

Fig. Ex. 2.4.7

- Every input is simply complemented in state q_0 .

Example 2.4.8

Design a Moore machine for generating output 1 if input of binary sequence 1 is preceded with exactly two zeros.

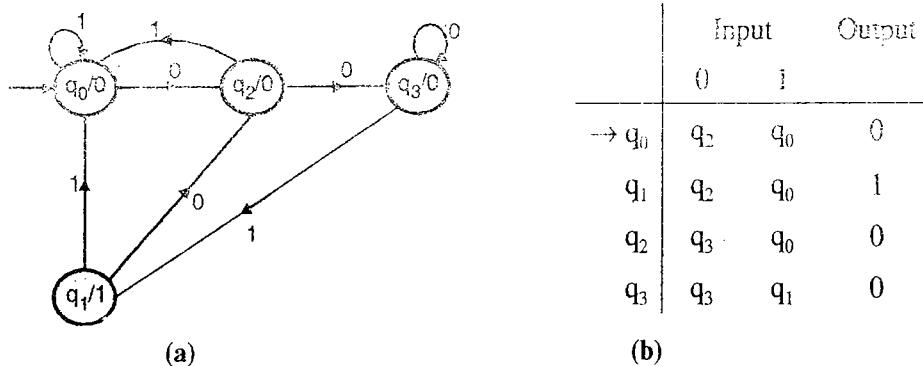
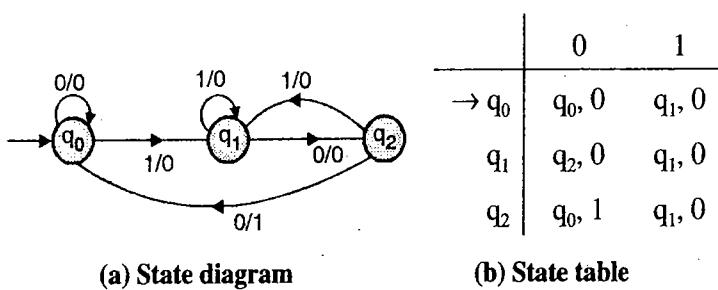
Solution :

Fig. Ex. 2.4.8

Meaning of various states :

 q_0 – initial state, associated with output 0 q_2 – preceding symbol is 0 of sequence 001, associated with output 0. q_3 – preceding two symbols as 00 of sequence 001, associated with output 0. q_1 – preceding three symbols as 001 of sequence 001, associated with output 1.**Example 2.4.9**

Design a mealy machine for a binary input sequence such that if the sequence ends with 100 the output is 1 otherwise output is 0.

Solution :

(a) State diagram

(b) State table

Fig. Ex. 2.4.9

Meaning of various states :

 q_0 – start state q_1 – previous symbol is 1 q_2 – preceding two symbols are 10A transition from q_2 to q_0 will make the preceding three symbol as 100 and hence the output 1.**Example 2.4.10**

Design a mealy machine for incrementing the value of any binary number by one. The output should also be a binary number with value one more than the given number.

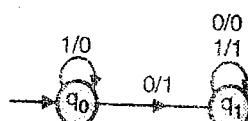
Solution :

When a binary number is incremented then trailing 1's become 0's and the first zero immediately before trailing 1's become 1.

For example, if we add 1 to 0101111 then the incremented value will be 0110000.

$$\begin{array}{r}
 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
 & + & & & & & & \\
 \hline
 & 0 & 1 & 1 & 0 & 0 & 0 & 0
 \end{array}$$

The mealy machine is shown in Fig. Ex. 2.4.10.



Input should be applied, starting from the rightmost symbol.

Fig. Ex. 2.4.10

| | | |
|-------------------|----------|----------|
| | 0 | 1 |
| $\rightarrow q_0$ | $q_1, 1$ | $q_0, 0$ |
| q_1 | $q_1, 0$ | $q_1, 1$ |

Example 2.4.11

Design a Moore machine for the following problem :

Input alphabet = {0, 1, 2}

Output alphabet = {a, b, c}

If the sum of 3 consecutive number is 0, 1, 4; output a.

If the sum is 2, 3; output b.

If the sum is 5, 6; output c.

Solution : Sum of three consecutive symbols will depend on :

1. Preceding two symbols 2. Next input.

Preceding two symbols could be :

| Symbols | State |
|---------|----------|
| 00 | q_{00} |
| 01 | q_{01} |
| 02 | q_{02} |
| 10 | q_{10} |
| 11 | q_{11} |
| 12 | q_{12} |
| 20 | q_{20} |
| 21 | q_{21} |
| 22 | q_{22} |

The next state and the output is given below :

1. An input 0 in state q_{00} will make sum as 0 and next state as q_{00}
2. An input 1 in state q_{00} will make sum as 1 and next state as q_{01}
3. An input 2 in state q_{00} will make sum as 2 and next state as q_{02}
4. An input 0 in state q_{01} will make sum as 1 and next state as q_{10}
5. An input 1 in state q_{01} will make sum as 2 and next state as q_{11}
6. An input 2 in state q_{01} will make sum as 3 and next state as q_{12}
- ⋮
- ⋮
25. An input 0 in state q_{22} will make sum as 4 and next state as q_{20}
26. An input 1 in state q_{22} will make sum as 5 and next state as q_{21}
27. An input 2 in state q_{22} will make sum as 6 and next state as q_{22}



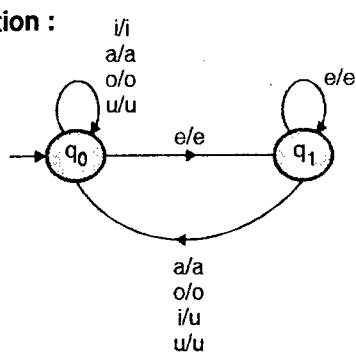
State transition table,

| | 0 | 1 | 2 |
|-------------------|------------|------------|------------|
| $\rightarrow q_0$ | q_{00}/a | q_{01}/a | q_{02}/b |
| q_{01} | q_{10}/a | q_{11}/b | q_{12}/b |
| q_{02} | q_{20}/b | q_{21}/b | q_{22}/a |
| q_{10} | q_{00}/a | q_{01}/b | q_{02}/b |
| q_{11} | q_{10}/b | q_{11}/b | q_{12}/a |
| q_{12} | q_{20}/b | q_{21}/a | q_{22}/c |
| q_{20} | q_{00}/b | q_{01}/b | q_{02}/a |
| q_{21} | q_{10}/b | q_{11}/a | q_{12}/c |
| q_{22} | q_{20}/a | q_{21}/c | q_{22}/c |

Example 2.4.12

Design a Mealy machine that will read sequences made up of vowels of English language, it will give output in same sequences, except in cases where a 'i' follows 'e', it will be changed to 'u'.

Solution :



(a) State diagram

| Current state | Next state output | | | | |
|-------------------|-------------------|----------|----------|----------|----------|
| | a | e | i | o | u |
| $\rightarrow q_0$ | q_0, a | q_1, e | q_0, i | q_0, o | q_0, u |
| q_1 | q_0, a | q_1, e | q_0, u | q_0, o | q_0, u |

(b) State table

Fig. Ex. 2.4.12

- Machine must convert every occurrence of ei into eu. In other cases, output will be same as input.
- State q_1 implies that the preceding character is e.
- An input i in stat q_1 will generate an output u.

Example 2.4.13

Give the Mealy and Moore 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".

Solution :

(a) Design of Mealy machine

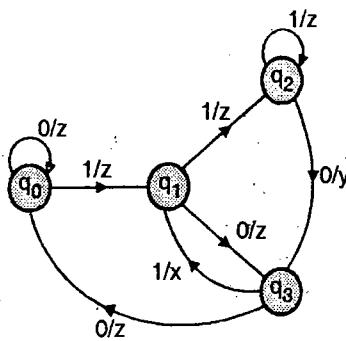


Fig. Ex. 2.4.13(a) : State diagram of Mealy machine

- The mealy machine must be designed to remember the preceding two symbols particularly '11' and '10'.

- The state q_2 is for preceding two symbols as '11'.
- The state q_3 is for preceding two symbols as '10'.
- An input of 0 in state q_2 will mean a sequence of 110 (hence the output y), and it will make preceding two symbols as 10 (q_3 stands for preceding two symbols as 10).
- Similar argument can be given for the state q_3 .

(b) Design of Moore machine

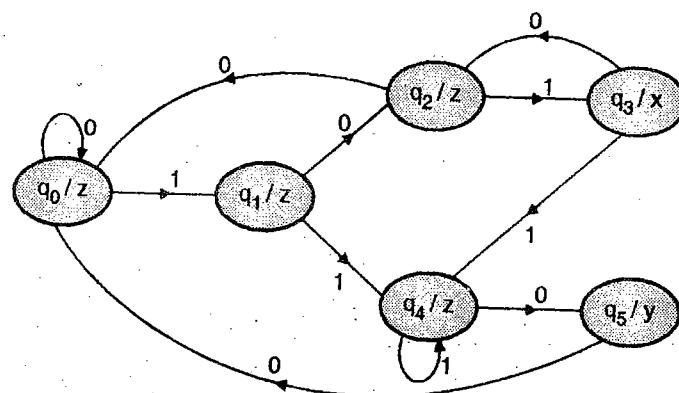


Fig. Ex. 2.4.13(b) : State diagram of the Moore machine

Example 2.4.14

Design a Moore and Mealy machine for a binary input sequence such that if it has a substring 110 the machine outputs A, if it has a substring 101 machine outputs B, otherwise outputs C.

Solution :

(a) Design of Mealy machine

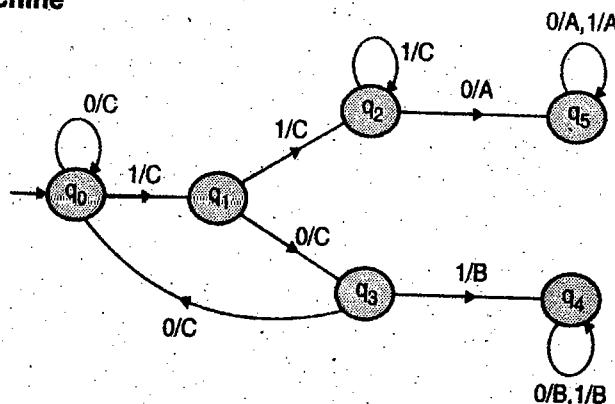


Fig. Ex. 2.4.14(a) : State diagram of the Mealy machine

Meaning of various states :

q_0 – Start state

q_1 – Preceding character is 1

q_2 – Preceding two characters are 11 and it needs a 0 to complete the sequence 110

q_3 – Preceding two characters are 10 and it needs a 1 to complete the sequence 101.

(b) Design of Moore machine

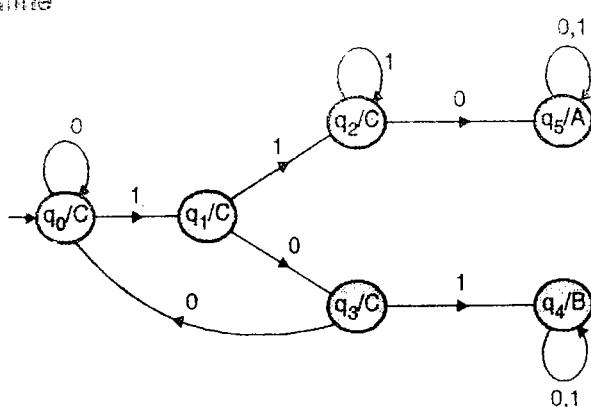


Fig. Ex. 2.4.14(b) : State diagram of the Moore machine

Example 2.4.15

Design finite state machine that compares two binary numbers to determine whether they are equal and which of two is larger.

Solution :

Let the two numbers are A and B.

$$\text{where, } A = a_0, a_{n-1} \dots a_i$$

$$B = b_0, b_{n-1} \dots b_i$$

Numbers are entered bitwise, starting from the least significant digit. If there is a mismatch at the i^{th} bit position then we can conclude the following.

$$A > B \quad \text{if } a_i > b_i$$

$$A < B \quad \text{if } a_i < b_i$$

Machine generates the following output :

$$A > B \dots +$$

$$A < B \dots -$$

$$A = B \dots 0$$

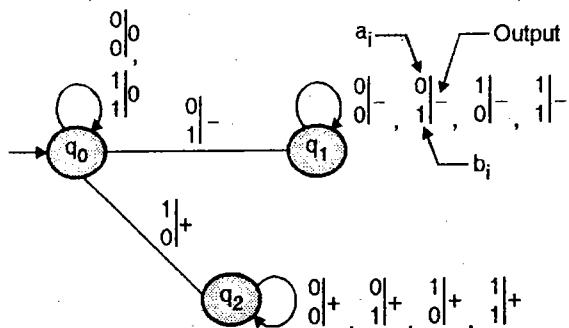


Fig. Ex. 2.4.15 : State diagram of the Mealy machine

Example 2.4.16

Design a Moore Machine for checking divisibility by 3 of a given decimal number (residue of 3).

Solution : The decimal system has 10 alphabet.

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The running remainder when a given decimal number is divided by 3 could be :

0 → associated state is q_0 .

1 → associated state is q_1 .

2 → associated state is q_2 .

The required Moore machine is given Fig. Ex. 2.4.16.

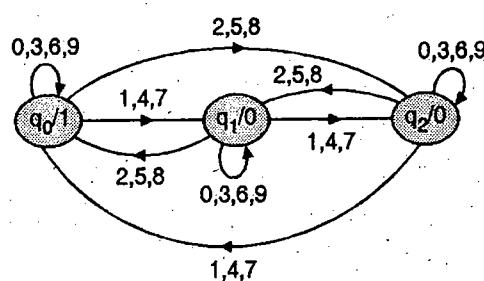


Fig. Ex. 2.4.16



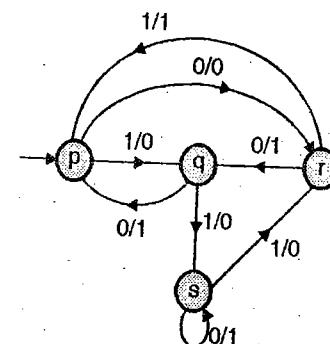
Syllabus Topic : Conversion of Mealy Machine into a Moore Machine

2.4.5 Conversion of a Mealy Machine into a Moore Machine

A Mealy machine can be transformed into an equivalent Moore machine so that both the machines show the same output behaviour for any input sequence. Conversion process is being explained with the help of a Mealy machine shown in Fig. 2.4.9.

| Present state | (Next state, output) | |
|-----------------|----------------------|---------|
| | Input 0 | Input 1 |
| $\rightarrow p$ | r, 0 | q, 0 |
| q | p, 1 | s, 0 |
| r | q, 1 | p, 1 |
| s | s, 1 | r, 0 |

(a) State table



(b) State diagram

Fig. 2.4.9 : Mealy machine considered for conversion into an equivalent Moore machine

Step 1 : Splitting of states :

It may be necessary to split some of the states of a Mealy machine to get an equivalent Moore machine. Splitting is based on incoming lines into a state and the associated outputs. We must split a state to handle different output values associated with incoming lines.

- There are two incoming lines into the state p.
 - From q to p on input 0, giving an output of 1.
 - From r to p on input 1, giving an output of 1.
 In every case output is 1 and hence splitting is not required.
- There are two incoming lines into the state q.
 - From p to q on input 1, giving an output of 0.
 - From r to q on input 0, giving an output of 1.

Since there are two different outputs, we must split q into two different states q_0 with output 0 and q_1 with output 1.

$$q \xrightarrow{\text{split}} \begin{array}{l} q_0/0 - \text{output 0} \\ q_1/1 - \text{output 1} \end{array}$$

- There are two incoming lines into the state r.
 - From p to r on input 0, giving an output of 0.
 - From s to r on input 1, giving an output of 0.
 In every case output is 0 and hence splitting is not required.
- There are two incoming lines into the states S.
 - From q to s on input 1, giving an output of 0.
 - From s to s on input 0, giving an output of 1.

Since there are two different outputs, we must split s into two different states, s_0 with output 0 and s_1 with output 1.

$$s \xrightarrow{\text{split}} \begin{array}{l} s_0/0 - \text{output 0} \\ s_1/1 - \text{output 1} \end{array}$$



Step 2 : Drawing of an equivalent Moore machine.

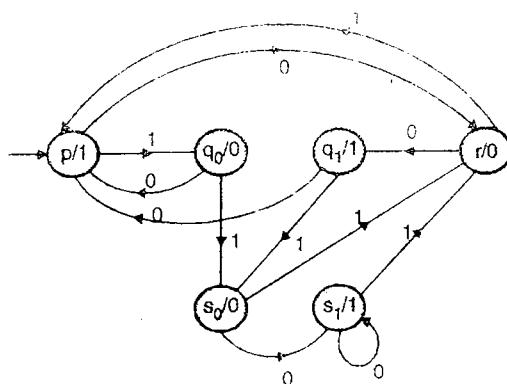


Fig. 2.4.9(c) : State diagram of Moore machine

- Every transition into q with output 0 is directed to q_0 .
- Every transition into q with output 1 is directed to q_1 .
- Every transition into s with output 0 is directed to s_0 .
- Every transition into s with output 1 is directed to s_1 .
- Every transition from q is duplicated both from q_0 and q_1 .
- Every transition from s is duplicated both from s_0 and s_1 .

| Transitions for Mealy Machine | Equivalent transitions for Moore Machine |
|-------------------------------|--|
| $\delta(p, 0) = r, 0$ | $\delta(p, 0) = r$ |
| $\delta(p, 1) = q, 0$ | $\delta(p, 1) = q_0$ q_0 is associated with output 0 |
| $\delta(q, 0) = p, 1$ | $\delta(q_0, 0) = p$ $\delta(q_1, 0) = p$ Transition is duplicated for both q_0 and q_1 |
| $\delta(q, 1) = s, 0$ | $\delta(q_0, 1) = s_0$ $\delta(q_1, 1) = s_0$ Transition is duplicated for both q_0 and q_1 . s_0 is associated with output 0. |
| $\delta(r, 0) = q, 1$ | $\delta(r, 0) = q_1$ q_1 is associated with output |
| $\delta(r, 1) = p, 1$ | $\delta(r, 1) = p$ |
| $\delta(s, 0) = s, 1$ | $\delta(s_0, 0) = s_1$ $\delta(s_1, 0) = s_1$ Transition is duplicated for both s_0 and s_1 . s_1 is associated with output 1. |
| $\delta(s, 1) = r, 0$ | $\delta(s_0, 1) = r$ $\delta(s_1, 1) = r$ Transition is duplicated for both s_0 and s_1 . |

State transition table for the Moore machine is given in Fig. 2.4.9(d).

| Present state | Next state | | Output |
|-----------------|------------|-------|--------|
| | 0 | 1 | |
| $\rightarrow p$ | r | q_0 | 1 |
| q_0 | p | s_0 | 0 |
| q_1 | p | s_0 | 1 |
| r | q_1 | p | 0 |
| s_0 | s_1 | r | 0 |
| s_1 | s_1 | r | 1 |

Fig. 2.4.9(d) : State transition table

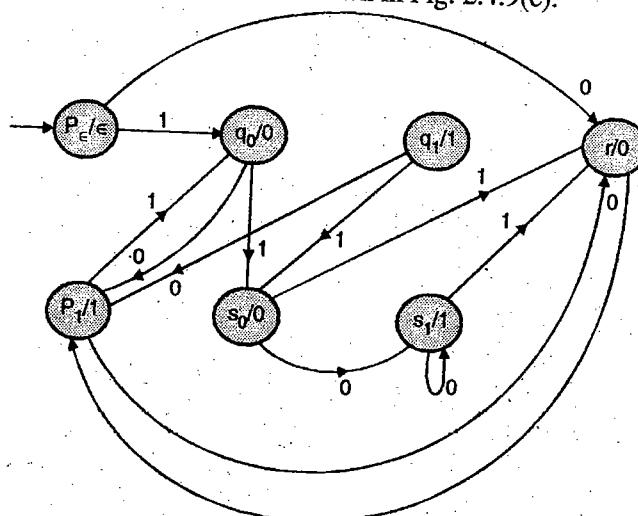
Step 3 : Moore machine of Fig. 2.4.9(c) produces an output 1 without any input. Thus, the Moore machine gives an output 1 on a zero length sequence. But the Mealy machine will produce no output for a zero length sequence.

To handle the above problem, we must split the start state p into two states :

p_e – Giving no output

p_1 – Giving an output 1.

Final Moore machine is drawn in Fig. 2.4.9(e).



(e) State diagram

| Present state | Next state | | Output |
|-------------------|------------|-------|------------|
| | 0 | 1 | |
| $\rightarrow p_e$ | r | q_0 | ϵ |
| p_1 | r | q_0 | 1 |
| q_0 | p_1 | s_0 | 0 |
| q_1 | p_1 | s_0 | 1 |
| r | q_1 | p_1 | 0 |
| s_0 | s_1 | r | 0 |
| s_1 | s_1 | r | 1 |

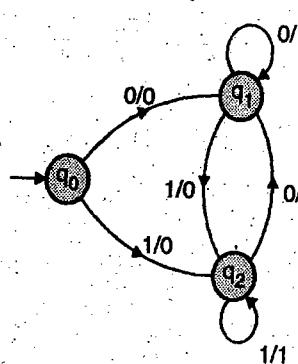
(f) State table

Fig. 2.4.9 : Final Moore machine

Example 2.4.17

Construct a Mealy machine that accept strings ending in '00' and '11'. Convert the same to a Moore machine.

Solution : (i) Construction of Mealy machine.



(a) State transition diagram of the Mealy machine

| Current state | (Next state, Output) | |
|-------------------|----------------------|----------|
| | 0 | 1 |
| $\rightarrow q_0$ | $q_1, 0$ | $q_2, 0$ |
| q_1 | $q_1, 1$ | $q_2, 0$ |
| q_2 | $q_0, 0$ | $q_1, 1$ |

(b) State transition table of the Mealy machine

Fig. Ex. 2.4.17



The Mealy machine is shown in the Fig. Ex. 2.4.17. Interpretation of various states is given below.

q_0 = Initial state

q_1 = Preceding input is 0, 0 as next input will complete the sequence 00.

q_2 = Preceding input is 1, 1 as next input will complete the sequence 11.

(ii) Conversion to a Moore machine :

Step 1 : Splitting of states

(i) There is no incoming line into the state q_0 . Output associated with q_0 will be ϵ .

(ii) There are three incoming lines into the state q_1 .

(a) q_0 to q_1 on input 0, with output 0.

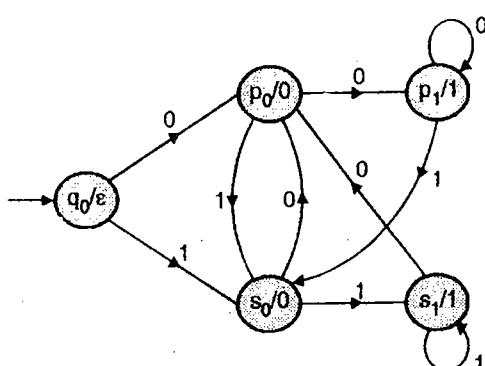
(b) q_1 to q_1 on input 0, with output 1.

(c) q_2 to q_1 on input 0, with output 0.

Since there are two different outputs, we must split q_0 into two different states p_0 with output 0 and p_1 with output 1.

(iii) Similarly, we must split q_2 into two different states s_0 with output 0 and s_1 with output 1.

Step 2 : Drawing of an equivalent Moore machine.



| Current state | Input | | Output |
|---------------|-------|-------|------------|
| | 0 | 1 | |
| q_0 | p_0 | s_0 | ϵ |
| p_0 | p_1 | s_0 | 0 |
| p_1 | p_1 | s_0 | 1 |
| s_0 | p_0 | s_1 | 0 |
| s_1 | p_0 | s_1 | 1 |

(c) State transition diagram of the Moore machine

(d) State transition table of the Moore machine

Fig. Ex. 2.4.17

| Transition in Mealy machine | Corresponding transitions in Moore machine |
|-----------------------------|--|
| $\delta(q_0, 0) = q_1, 0$ | $\delta(q_0, 0) = p_0$, p_0 is associated with output 0. |
| $\delta(q_0, 1) = q_2, 0$ | $\delta(q_0, 1) = s_0$, s_0 is associated with output 0. |
| $\delta(q_1, 1) = q_1, 1$ | $\delta(p_0, 0) = p_1$ $\delta(p_1, 0) = p_1$ Transition is duplicated for both p_0 and p_1 . p_1 is associated with output 1 |
| $\delta(q_1, 1) = q_2, 0$ | $\delta(p_0, 1) = s_0$ $\delta(p_1, 1) = s_0$ Transition is duplicated for both p_0 and p_1 . s_0 is associated with output 0. |
| $\delta(q_2, 0) = q_1, 0$ | $\delta(s_0, 0) = p_0$ $\delta(s_1, 0) = p_0$ Transition is duplicated for both s_0 and s_1 . p_0 is associated with output 0. |



| Transition in Mealy machine | Corresponding transitions in Moore machine |
|-----------------------------|--|
| $\delta(q_2, 1) = q_2, 1$ | $\delta(s_0, 1) = s_1$ $\delta(s_1, 1) = s_1$ Transition is duplicated for both s_0 and s_1 . s_1 is associated with output 1. |

Example 2.4.18

Consider the following Mealy machine. Convert it to Moore.

| Present state | Next state | | | |
|-------------------|------------|--------|-------|--------|
| | a = 0 | | a = 1 | |
| | State | Output | State | Output |
| $\rightarrow q_1$ | 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 :**Step 1 :** Splitting of states.

- (i) Incoming lines into q_1 always give an output 1.
- (ii) Incoming lines into q_2 give both 0 and 1 as output.

We must split q_2 into two different states p_0 with output 0 and p_1 with output 1.

- (iii) Incoming lines into q_3 give both 0 and 1 as output.

We must split q_3 into two different states s_0 with output 0 and s_1 with output 1.

- (iv) Incoming lines into q_4 always give an output 1.

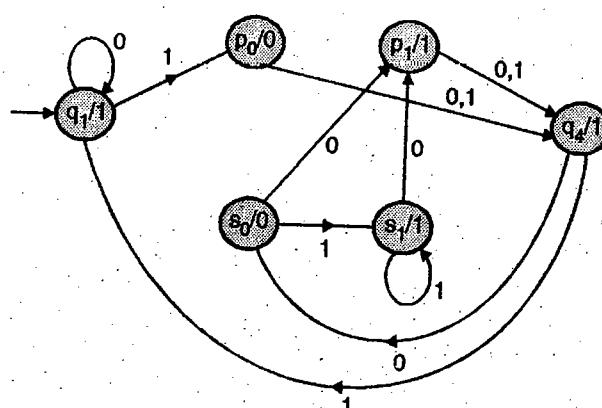
Step 2 : Drawing of an equivalent Moore machine.

Fig. Ex. 2.4.18(a) : Moore machine

Step 3 : Moore machine of Fig. Ex. 2.4.18(a) produces an output of 1 without any input. But the Mealy machine given in Example 2.4.18 does not produce any output for a zero-length input sequence.
To handle the above problem, we must split the start state q_1 into two states :

 r_e – Giving no output r_1 – Giving an output 1.

Final Moore machine is drawn in Fig. Ex. 2.4.18(b) and 2.4.18(c).

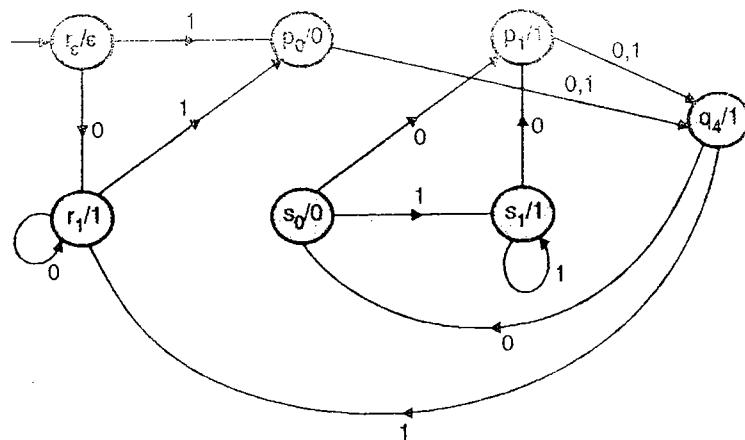


Fig. Ex. 2.4.18(b) : State transition diagram for an equivalent Moore machine

| Current state | 0 | 1 | Output |
|-------------------|-------|-------|------------|
| $\rightarrow r_e$ | r_1 | p_0 | ϵ |
| r_1 | r_1 | p_0 | 1 |
| p_0 | q_4 | q_4 | 0 |
| p_1 | q_4 | q_4 | 1 |
| s_0 | p_1 | s_1 | 0 |
| s_1 | p_1 | s_1 | 1 |

Fig. Ex. 2.4.18(c) : State transition table for an equivalent Moore machine

Example 2.4.19

Convert the Mealy machine shown in Fig. Ex. 2.4.19 to Moore machine.

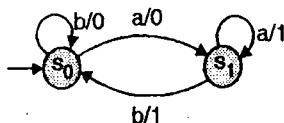


Fig. Ex. 2.4.19

Solution :

Step 1 : Splitting of states.

- Incoming lines into s_0 give both 0 and 1 as output. In addition, s_0 is an initial state and hence an ϵ -output condition should also be considered. s_0 is split into three states :
 - p_e with output ϵ ,
 - p_0 with output 0,
 - p_1 with output 1.
- Incoming lines into s_1 give both 0 and 1 as output. We must split s_1 into two different states.
 - q_0 with output 0,
 - q_1 with output 1.

Step 2 : Drawing of an equivalent Moore machine.

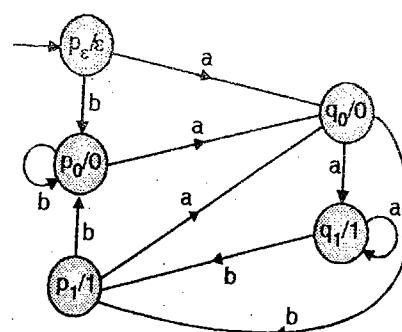


Fig. Ex. 2.4.19(a) : Moore machine for Example 2.4.19

- Transitions for s_0 are duplicated for p_e , p_0 and p_1 .
- Transitions for s_1 are duplicated for q_0 and q_1 .

Example 2.4.20 [SPPU - Dec. 16, 6 Marks]

Convert the Mealy machine shown in Fig. Ex.2.4.20 to its equivalent Moore machine.

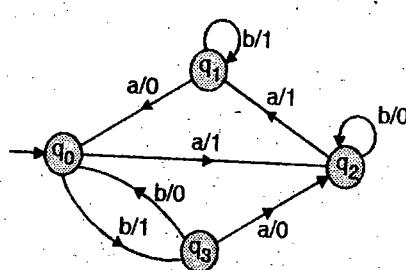


Fig. Ex. 2.4.20

Solution :

Step 1 : Splitting of states.

- (i) q_0 is associated with three different outputs (on the basis of incoming lines) ϵ , 0, 1. We must split q_0 into three different states :

A_ϵ with output ϵ (q_0 is a start state)

A_0 with output 0,

A_1 with output 1.

- (ii) q_1 is associated with two different outputs 0 and 1.

We must split q_1 into two different states :

B_0 with output 0,

B_1 with output 1.

- (iii) q_2 is associated with two different outputs 0 and 1.

We must split q_2 into two different states :

C_0 with output 0,

C_1 with output 1.

- (iv) q_3 is associated with two different outputs 0 and 1. We must split q_3 into two different states :

D_0 with output 0,

D_1 with output 1.



Step 2 : Drawing of an equivalent Moore machine.

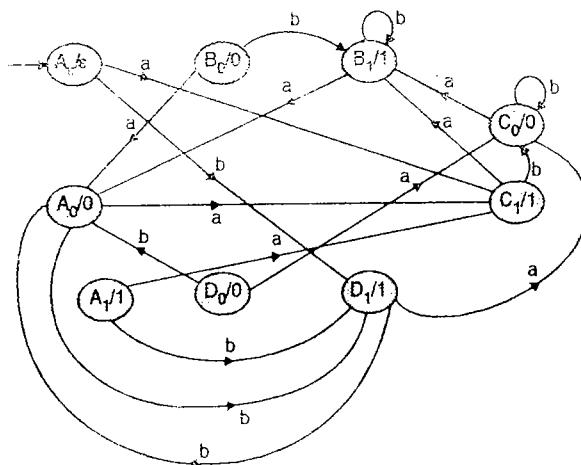


Fig. Ex. 2.4.20(a) : Mealy machine for Example 2.4.20

Syllabus Topic : Conversion of Moore Machine into a Mealy Machine

2.4.6 Conversion of a Moore Machine into a Mealy Machine

A Moore machine can be transformed to a corresponding Mealy machine in two steps. These steps are :

1. Construction of a trivial Mealy machine -
By moving output associated with a state to transitions entering into that state.
2. Minimization of the trivial Mealy machine obtained in step 1.

These two steps are being explained with the help of a Moore machine shown in Fig. 2.4.10(a).

| Present state | Next state | | Output |
|-----------------|------------|---|--------|
| | 0 | 1 | |
| $\rightarrow p$ | s | q | 0 |
| q | q | r | 1 |
| r | r | s | 0 |
| s | s | p | 0 |

Fig. 2.4.10(a) : Moore machine considered for conversion

Step 1 : Construction of trivial Mealy machine.

- (i) Moving the output associated p, which is 0, to transitions entering into state p, we get :

| | 0 | 1 | |
|-----------------|---|------|---|
| $\rightarrow p$ | s | q | - |
| q | q | r | 1 |
| r | r | s | 0 |
| s | s | p, 0 | 0 |

- (ii) Moving the output associated with q, which is 1 to transitions entering into state q, we get :

| | 0 | 1 | |
|-----------------|------|------|---|
| $\rightarrow p$ | s | q, 1 | - |
| q | q, 1 | r | - |
| r | r | s | 0 |
| s | s | p, 0 | 0 |

(iii) Moving the output associated with r, which is 0 to transitions entering into the state r, we get :

| | | | |
|-----------------|------|------|---|
| | 0 | 1 | |
| $\rightarrow p$ | s | q, 1 | - |
| q | q, 1 | r, 0 | - |
| r | r, 0 | s | - |
| s | s | p, 0 | 0 |

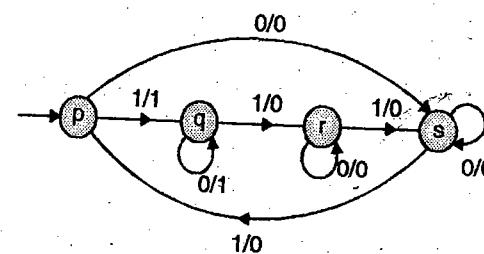
(iv) Moving the output associated with s, which is 0 to transitions entering the state s, we get :

| | | |
|-----------------|------|------|
| | 0 | 1 |
| $\rightarrow p$ | s, 0 | q, 1 |
| q | q, 1 | r, 0 |
| r | r, 0 | s, 0 |
| s | s, 0 | p, 0 |

Fig. 2.4.10(b) : A trivial Mealy machine

Step 2 : The trivial Mealy machine obtained in Fig. 2.4.10(b) can not be minimized further.
Final Mealy machine is given in Fig. 2.4.10(c) and 2.4.10(d).

| | 0 | 1 |
|-----------------|------|------|
| $\rightarrow p$ | s, 0 | q, 1 |
| q | q, 1 | r, 0 |
| r | r, 0 | s, 0 |
| s | s, 0 | p, 0 |



(c) Transition table for final Mealy machine

(d) Transition diagram for final Mealy machine

Fig. 2.4.10

Example 2.4.21

Change the given Moore machine into Mealy machine.

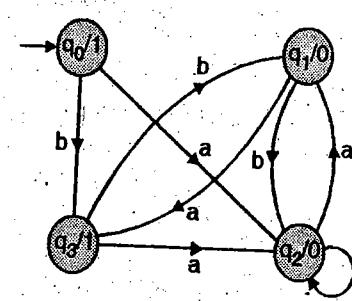


Fig. Ex. 2.4.21

Solution :

Step 1 : Construction of a trivial Mealy machine by moving output associated with a state to transitions entering that state, we get :

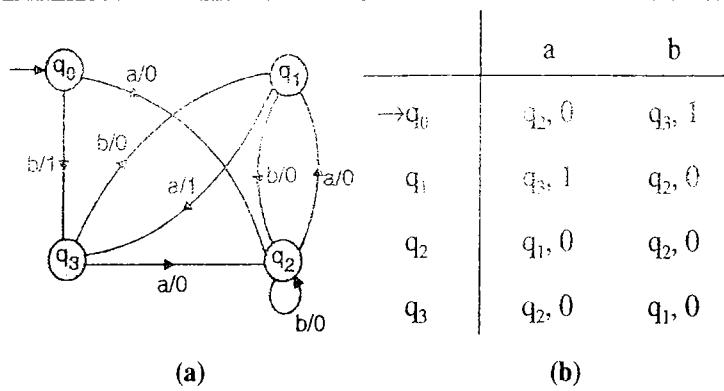


Fig. Ex. 2.4.21

Step 2 : Minimization of Mealy machine obtained in Fig. Ex. 2.4.21(a).

(i) 1-equivalent partitions P_1 based on outputs

$$P_1 = (q_0) (q_1) (q_2, q_3)$$

(ii) 2-equivalent partitions P_2 based on transition,

q_2 is mapped to block (q_1) on input 0.

q_3 is mapped to block (q_2, q_3) on input 0.

\therefore Behavior of q_2 is different from q_3 .

$$P_2 = (q_0) (q_1) (q_2) (q_3)$$

Mealy machine given in Fig. Ex. 2.4.21(a) is the final Mealy machine.

Example 2.4.22

Change the given Moore machine into Mealy machine.

| Present state | Next state | | Output |
|-------------------|------------|-------|------------|
| | 0 | 1 | |
| $\rightarrow p_0$ | r | q_0 | ϵ |
| p_1 | r | q_0 | 1 |
| q_0 | p_1 | s_0 | 0 |
| q_1 | p_1 | s_0 | 1 |
| r | q_1 | p_1 | 0 |
| s_0 | s_1 | r | 0 |
| s_1 | s_1 | r | 1 |

Solution :

Step 1: Construction of a trivial Mealy machine by moving output associated with a state to transitions entering that state, we get :

| Present state | (Next state, Output) | |
|-------------------|----------------------|----------|
| | 0 | 1 |
| $\rightarrow p_0$ | r, 0 | $q_0, 0$ |
| p_1 | r, 0 | $q_0, 0$ |
| q_0 | $p_1, 1$ | $s_0, 0$ |
| q_1 | $p_1, 1$ | $s_0, 0$ |
| r | $q_1, 1$ | $p_1, 1$ |
| s_0 | $s_1, 1$ | r, 0 |
| s_1 | $s_1, 1$ | r, 0 |

Fig. Ex. 2.4.22(a) : A trivial Mealy machine

Step 2 : Minimization of the trivial Mealy machine obtained in Fig. Ex. 2.4.22(a).

(i) 1-equivalent partitions P_1 based on outputs.

$$P_1 = (p_0, p_1) (q_0, q_1, s_0, s_1) (r)$$

(ii) 2-equivalent partitions p_2 based on transitions.

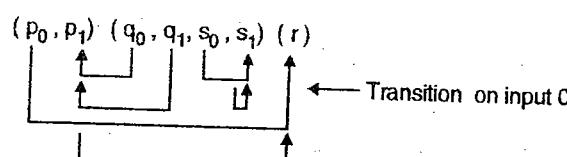


Fig. Ex. 2.4.22(b)

On input 0, q_0 and q_1 are mapped to block (p_0, p_1)

On input 0, s_0 and s_1 are mapped to block (q_0, q_1, s_0, s_1)

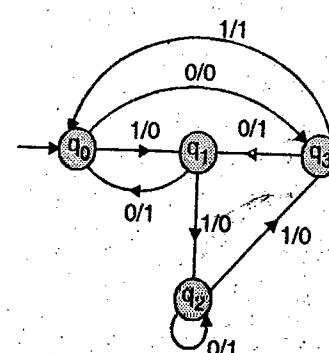
∴ Behavior of q_0, q_1 is different from s_0, s_1 .

$$P_2 = (p_0, p_1) (q_0, q_1) (s_0, s_1) (r)$$

Further partitioning is not possible. The minimum state Mealy machine is given in Fig. Ex. 2.4.22(c) and 2.4.22(d)

| Present state | (Next state, Output) | |
|--------------------------------|----------------------|----------|
| | 0 | 1 |
| $(p_0, p_1) = \rightarrow q_0$ | $q_3, 0$ | $q_1, 0$ |
| $(q_0, q_1) = q_1$ | $q_0, 1$ | $q_2, 0$ |
| $(s_0, s_1) = q_2$ | $q_2, 1$ | $q_3, 0$ |
| $(r) = q_3$ | $q_1, 1$ | $q_0, 1$ |

(c) State transition table of Mealy machine



(d) State transition diagram of Mealy machine

Fig. Ex. 2.4.22

Example 2.4.23

Consider the Moore machine described by the transition table given below. Construct corresponding Mealy machine.

| 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 :

Step 1 : Construction of a trivial Mealy machine by moving output associated with a state to transitions entering that state, we get :

| | 0 | 1 |
|-------------------|----------|----------|
| $\rightarrow q_1$ | $q_1, 0$ | $q_2, 0$ |
| q_2 | $q_1, 0$ | $q_3, 1$ |
| q_3 | $q_1, 0$ | $q_3, 1$ |

Step 2 : Minimization of the trivial Mealy machine obtained in step1.

States q_2 and q_3 are equivalent as they have the same output and transition behaviour.

The minimum state machine is given below :

| | | |
|--------------------|----------|----------|
| | 0 | 1 |
| $\rightarrow q_1$ | $q_1, 0$ | $q_2, 0$ |
| $(q_2, q_3) = q_2$ | $q_1, 0$ | $q_2, 1$ |

Example 2.4.24

Convert the following Moore machine to Mealy machine.

| State | Input | | Output |
|-------|-------|-------|--------|
| | a | b | |
| q_0 | q_1 | q_3 | 1 |
| q_1 | q_3 | q_1 | 0 |
| q_2 | q_0 | q_3 | 0 |
| q_3 | q_3 | q_2 | 1 |

Solution :

Step 1 : Construction of trivial mealy machine by moving output associated with states on incoming transitions.

| State | Input | |
|-------|---------|---------|
| | a | b |
| q_0 | $q_1/0$ | $q_3/1$ |
| q_1 | $q_3/1$ | $q_1/0$ |
| q_2 | $q_0/1$ | $q_3/1$ |
| q_3 | $q_3/1$ | $q_2/0$ |

Step 2 : Minimization of trivial mealy machine.

States are grouped on the basis of output.

$(q_0) (q_1, q_3) (q_2)$

The transition behaviour of q_1 and q_3 are different under the input 'b'. Hence the states q_1 and q_3 are distinguishable.

\therefore The final mealy machine is same as the trivial mealy machine and it is given Fig. Ex. 2.4.24.

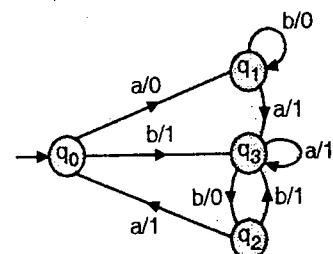


Fig. Ex. 2.4.24

Example 2.4.25 SPPU - May 15, 6 Marks

Construct Moore machine equivalent for the given Mealy machine.

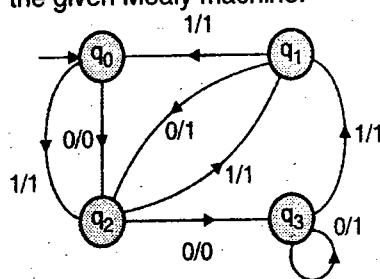


Fig. Ex. 2.4.25

Solution :

Step 1 : Splitting of states on the basis of outputs associated with incoming lines.

$q_0 - q_{01}$ for output 1 and $q_{0\epsilon}$ as it is an initial state.

$q_2 - q_{20}$ for output 0 and q_{21} for output 1.

$q_1 - q_{11}$ for output 1 for all incoming lines.

$q_3 - q_{30}$ for output 0 and q_{31} for output 1 for incoming lines.



Step 2 : Drawing of an equivalent moore machine.

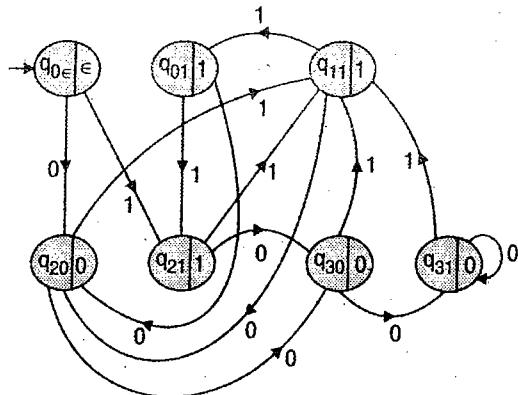


Fig. Ex. 2.4.25(a)

Example 2.4.26 SPPU - Aug. 15, 4 Marks

Convert following Moore machine into Mealy machine.

| Present State | Next State | | Output |
|---------------|------------|-------|--------|
| | a = 0 | a = 1 | |
| q0 | q3 | q1 | 1 |
| q1 | q2 | q3 | 0 |
| q2 | q2 | q1 | 0 |
| q3 | q1 | q0 | 1 |

Solution :

Step 1 : Construction of a trivial mealy machine by moving output associated with a state to transitions entering that state, we get

| | 0 | 1 |
|-------------------|----------|----------|
| $\rightarrow q_0$ | $q_3, 1$ | $q_1, 0$ |
| q_1 | $q_2, 0$ | $q_3, 1$ |
| q_2 | $q_2, 0$ | $q_1, 0$ |
| q_3 | $q_1, 0$ | $q_0, 1$ |

The above mealy machine cannot be minimized further.

\therefore The machine obtained in step 1 is the final mealy machine.

2.5 Minimization of a Mealy Machine

Minimization of a Mealy machine is almost similar to minimization of a DFA (Discussed in Chapter 3).

- In a DFA, initial grouping is based on accepting and non-accepting states.
- In a Mealy machine, initial grouping is based on outputs.

| Current state | (Next state, Output) | |
|-------------------|----------------------|----------|
| | 0 | 1 |
| $\rightarrow q_0$ | $q_4, 0$ | $q_3, 1$ |
| q_1 | $q_5, 0$ | $q_3, 0$ |
| q_2 | $q_4, 0$ | $q_1, 1$ |
| q_3 | $q_5, 0$ | $q_1, 0$ |
| q_4 | $q_2, 0$ | $q_5, 1$ |
| q_5 | $q_1, 0$ | $q_2, 0$ |

Fig. 2.5.1 : Mealy machine being considered for minimization



Minimization process is being explained step wise for the Mealy machine given in Fig. 2.5.1.

Step 1 : Finding 1-equivalent partitions P_1 based on outputs.

$$P_1 = (q_0, q_2, q_4) (q_1, q_3, q_5)$$

q_0, q_2, q_4 have the same output behaviour i.e. 0, 1.

q_1, q_3, q_5 have the same output behaviour i.e. 0, 0.

Step 2 : Finding 2-equivalent partitions P_2 based on transitions.

Behaviour of q_5 is different from q_1 and q_3 . 1-successor of q_5 is block 1. 1-successor of q_1 and q_3 is block 2.

$$P_2 = \begin{matrix} (q_0, q_2, q_4) & (q_1, q_3) & (q_5) \\ \text{Block 1} & \text{Block 2} & \text{Block 3} \end{matrix}$$

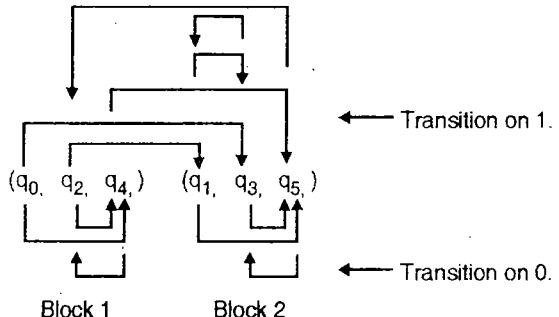


Fig. 2.5.1(a)

Step 3 : Finding 3-equivalent partitions P_3 based on transitions.

1-successor of q_0 and q_2 is the block (q_1, q_3)

1-successor of q_4 is the block (q_5) .

Behaviour of q_4 is different from q_0 and q_2 .

$$P_3 = (q_0, q_2) (q_4) (q_1, q_3) (q_5)$$

Further division is not possible. The minimum machine is drawn in Fig. 2.5.1(b) and 2.5.1(c).

| Present state | (Next state, Output) | |
|-----------------------|----------------------|----------|
| | 0 | 1 |
| (q_0, q_2) as p_0 | $p_1, 0$ | $p_2, 1$ |
| (q_4) as p_1 | $p_0, 0$ | $p_3, 1$ |
| (q_1, q_3) as p_2 | $p_3, 0$ | $p_2, 0$ |
| (q_5) as p_3 | $p_2, 0$ | $p_0, 0$ |

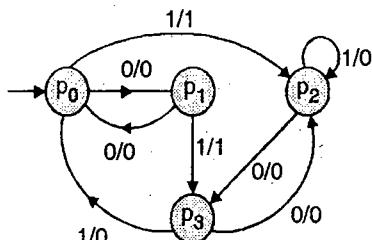


Fig. 2.5.1(b) : State table for minimum state Mealy machine

Fig. 2.5.1(c) : State diagram for minimum state Mealy machine

Example 2.5.1

Find the equivalent partition and corresponding reduced machine in standard form, for the following machine

| PS | NS, Z | |
|----|-------|-------|
| | X = 0 | X = 1 |
| A | F, 0 | B, 1 |
| B | G, 0 | A, 1 |
| C | B, 0 | C, 1 |
| D | C, 0 | B, 1 |
| E | D, 0 | A, 1 |
| F | E, 1 | F, 1 |
| G | E, 1 | G, 1 |

PS = Present state

NS = Next state

Z = Output

X = I/P

Solution :

Step 1 : 1-equivalent partitions P_1 based on outputs

$$P_1 = (A, B, C, D, E) (F, G)$$

Step 2 : 2-equivalent partitions P_2 based on transitions.

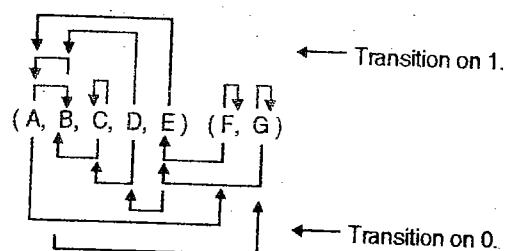


Fig. Ex. 2.5.1(a)

On input 0, A, B are mapped to block (F, G)

On output 0, C, D and E are mapped to block (A, B, C, D, E)

\therefore Behavior of A, B is different from C, D, E

$$P_2 = (A, B) (C, D, E) (F, G)$$

Step 3 : 3-equivalent partitions based on transition.

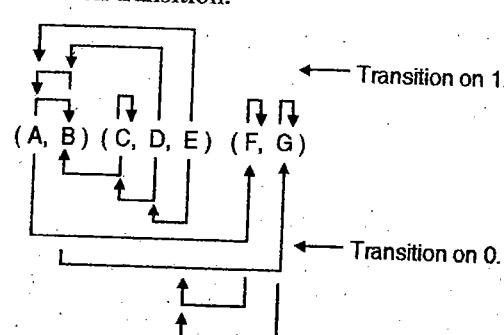


Fig. Ex. 2.5.1(b)

Behaviour of C is different from D, E.

$$P_3 = (A, B) (C) (D, E) (F, G)$$

Step 4 : 4-equivalent partitions based on transition.

On input 0, D is mapped to block (C)

On input 0, E is mapped to block (D, E)

\therefore Behaviour of D is different from E.

$$P_4 = (A, B) (C) (D) (E) (F, G)$$

Further division is not possible.

The minimum state Mealy machine is drawn in Fig. Ex. 2.5.1(e).

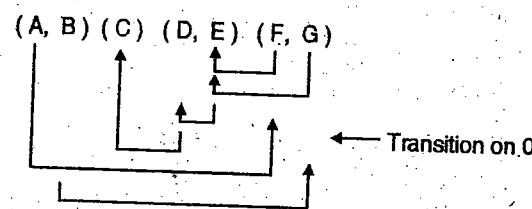
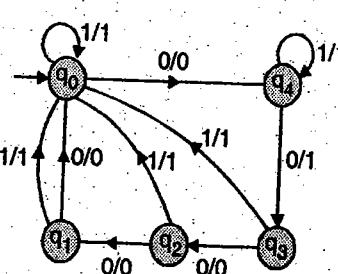


Fig. Ex. 2.5.1(c)

| Current state | (Next state, Output) | |
|----------------|----------------------|----------|
| | 0 | 1 |
| $(A, B) = q_0$ | $q_4, 0$ | $q_0, 1$ |
| $(C) = q_1$ | $q_0, 0$ | $q_1, 1$ |
| $(D) = q_2$ | $q_1, 0$ | $q_0, 1$ |
| $(E) = q_3$ | $q_2, 0$ | $q_0, 1$ |
| $(F, G) = q_4$ | $q_3, 1$ | $q_4, 1$ |

(d) State transition table



(e) State transition diagram

Fig. Ex. 2.5.1 : Minimum state Mealy machine for Example 2.5.1

Example 2.5.2 SPPU - Aug. 15, 6 Marks

Minimize the following automata.

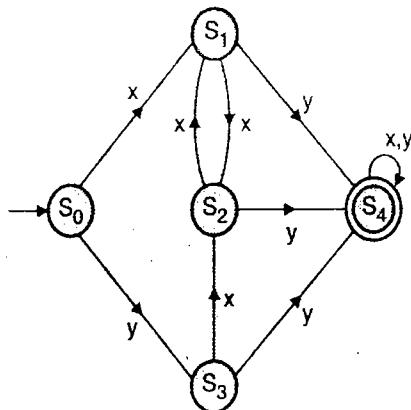


Fig. Ex. 2.5.2

Solution :

Step 1 : Finding 0-equivalence partitioning of states by putting final and non-final states into independent blocks.

$$p_0 = (s_0, s_1, s_2, s_3) \quad (s_4)$$

block 1 block 2

Step 2 : Finding 1-equivalence partitioning of states by considering transitions on 'x' and 'y'.

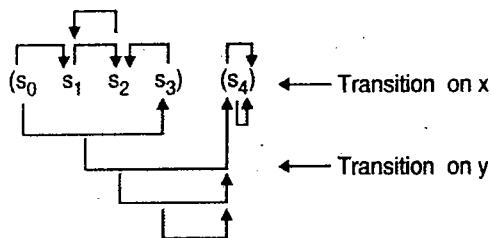


Fig. Ex. 2.5.2 (a)

The state s_0 is separable from the states (s_1, s_2, s_3) on the input 'y'.

$$p_1 = (s_0) (s_1, s_2, s_3) (s_4)$$

The three states (s_1, s_2, s_3) are not separable. The final automata is given below.

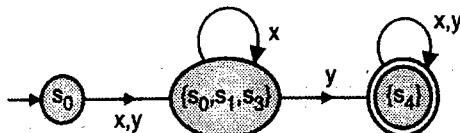


Fig. Ex. 2.5.2 (b)



CHAPTER

3

Regular Expressions (RE) and Languages

Syllabus

Definition and Identities of Regular Expressions, Construction of Regular Expression of the given L, Construction of Language from the RE, Construction of FA from the given RE using direct method, Conversion of FA to RE using Arden's Theorem, Pumping Lemma for RL, Closure properties of RLs, Applications of Regular Expressions.

Syllabus Topic : Construction of Regular Expression of the given L
3.1 Introduction

SPPU - Dec. 13

University Question

- Q.** Define regular expression with example.

(Dec. 2013, 2 Marks)

The set of strings accepted by finite automata is known as regular language. This language can also be described in a compact form using a set of operators.

These operators are :

- 1) $+$, union operator
- 2) \cdot , concatenation operator
- 3) $*$, star or closure operator.

An expression written using the set of operators ($+, \cdot, ^*$) and describing a regular language is known as regular expression. Regular expressions for some basic automata are given in Fig. 3.1.1.

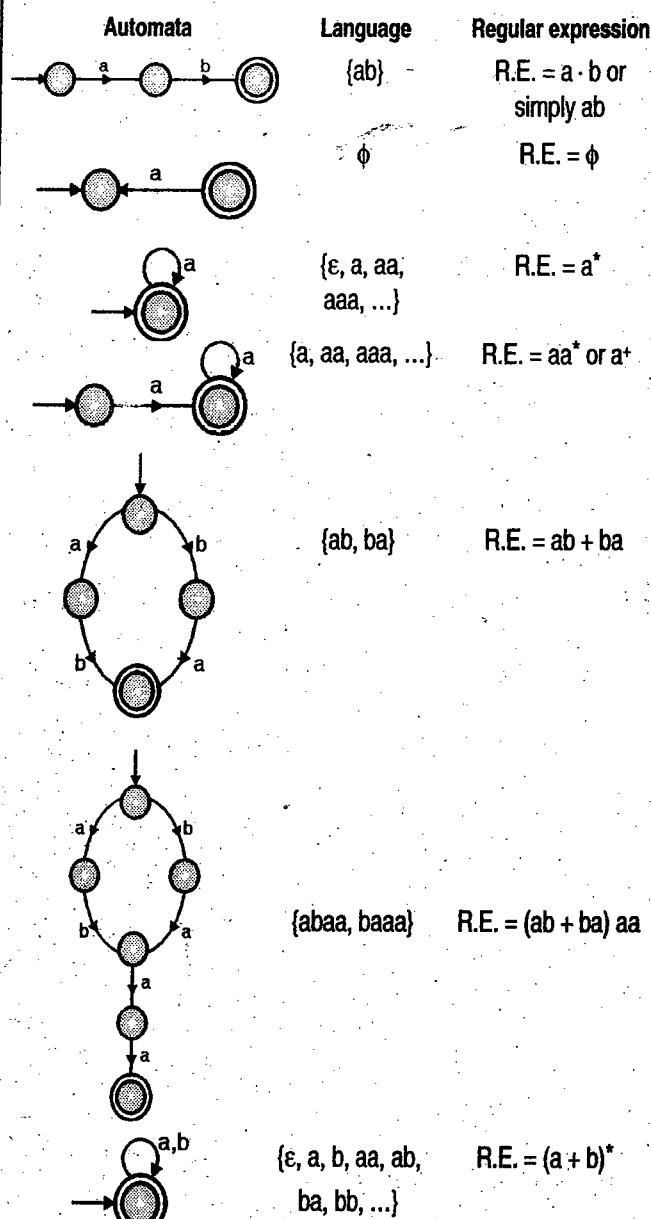
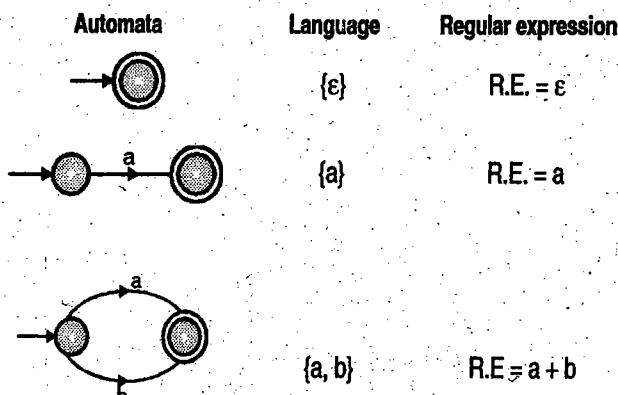


Fig. 3.1.1 : Examples on regular expression



If R_1 and R_2 are regular expressions then :

$R_1 + R_2$ is also regular,

$R_1 \cdot R_2$ is also regular,

R_1^* is also regular,

R_2^* is also regular,

R_1^+ is also regular.

$[R_1^+]$ stands for one or more occurrences of R_1

- 0^* stands for a language in which a word contains zero or more 0's.
- $(0 + 1)^*$ stands for a language in which a word ω contains any combination of 0's and 1's and $|\omega| \geq 0$.

Example 3.1.1

Write regular expression for the following languages.

- The set {1010}
- The set {10, 1010}
- The set {ε, 10, 01}
- The set {ε, 0, 00, 000, ...}
- The set {0, 00, 000, ...}
- The set of strings over alphabet {0, 1} starting with 0.
- The set of strings over alphabet {0, 1} ending in 1.
- The set of strings over alphabet {a, b} starting with a and ending in b.
- The set of strings recognized by $(a + b)^3$

Solution :

- (a) Regular expression,

R.E. = 1010 stands for the set {1010}

- (b) Regular expression,

R.E. = 10 + 1010

represent the set {10, 1010}

- (c) The set {ε, 10, 01} is represented by the regular expression,

R.E. = ε + 10 + 01

- (d) The set {ε, 0, 00, 000, ...} is represented by the regular expression,

R.E. = 0*

- (e) The set {0, 00, 000, ...} is represented by the regular expression,

R.E. = 0⁺ R.E. = 00*

00* represents 0{ε, 0, 00, 000, ...} = {0, 00, 000, ...}

- (f) The set of string starting with 0 is given by 0 followed by any combination of 0, 1.

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

(g) The regular expression for a set over alphabet {0, 1} ending in 1 is given by :

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

(h) The regular expression for a set of strings over alphabet {a, b} starting with a and ending in b is given by,

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

(i) $(a + b)^3$ stands for $(a + b)(a + b)(a + b) = aaa + aab + aba + abb + baa + bab + bba + bbb$.

Example 3.1.2

Write a regular expression to identify valid decimal integer constant for 'C' language. Justify RE with example.

Solution : The RE is being given for an unsigned decimal integer constant.

$(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)^*$

The first term ensures that the integer constant is of non-zero length. The second term can generate any number of digits.

Any valid integer constant like 1234 will be generated by the above RE.

Example 3.1.3

Write a regular expression to search .dat file (s) having starting character "p" and ending with "zw". Justify RE with example (s).

Solution : The required regular expression in UNIX will be written as P * z w · dat

- Star matches one or more characters.
- Star(*) is a wild character.
- The corresponding command in UNIX can be written as \$ ls -l p*zw.dat

3.2 Finite Automata Representing a Regular Expression

We can always construct an ε-NFA for recognizing a set of strings represented by a regular expression.

ε-NFA can be converted directly into a DFA.

OR

ε-NFA can be converted into an NFA without ε-moves which can be converted into a DFA.

A regular expression can be converted to an ε-NFA through recursive application of the operations +, ·, * on alphabet Σ, φ and ε.



Recursive nature of regular expressions

A regular expression $R = (1 + 0(10)^*)^*$ can be written as $R = P^*$, where $P = 1 + 0(10)^*$

P can be written as $P_1 + P_2$, where $P_1 = 1$ and $P_2 = 0(10)^*$.

P_2 can be written as $P_3 \cdot P_4$, where P_3 is 0 and P_4 is $(10)^*$.

P_4 can be written as P_5^* , where P_5 is 10.

P_5 can be written as $P_6 \cdot P_7$, where P_6 is 1 and P_7 is 0.

From the recursive nature of regular expression, we can conclude that :

If FAs for two regular expressions R_1 and R_2 are given and if we can construct composite FA for

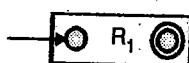
- (1) $R_1 + R_2$
- (2) $R_1 \cdot R_2$
- (3) R_1^*

Then we can construct finite state automata (FA) for any regular expression.

3.2.1 Composite Finite State Automata

Let us assume the existence of FAs for regular expressions R_1 and R_2 :

FA for R_1 is given by

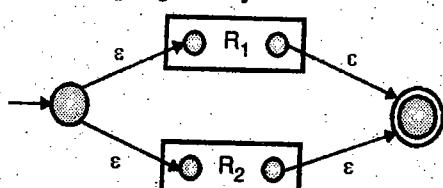


FA for R_2 is given by

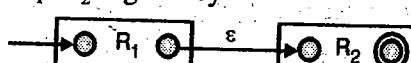


Each FA is assumed to have one final state.

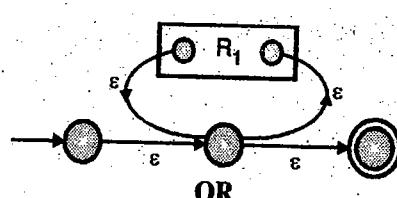
1. FA for $R_1 + R_2$ is given by



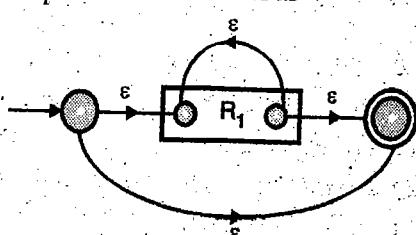
2. FA for $R_1 \cdot R_2$ is given by



3. FA for R_1^* is given by



FA for R_1^* can also be shown as



Finite automata for some primitive strings are shown in Fig. 3.2.1.

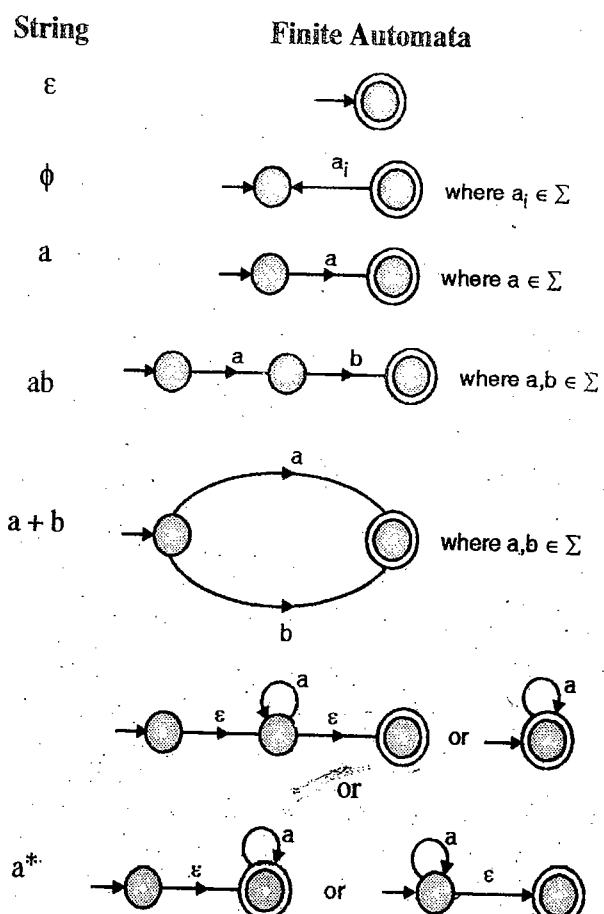


Fig. 3.2.1

Example 3.2.1

Write a note on Kleen's theorem.

Solution : Kleen's theorem says that if a language can be defined by any one of these three ways :

- (i) Transition Diagram
- (ii) Transition Table
- (iii) Regular Expression

Then, it can be defined by other two ways.

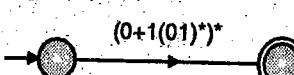
Example 3.2.2

Construct a finite automata for the regular expression

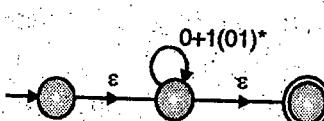
$$R = (0 + 1(01)^*)^*$$

Solution :

Step 1 :



Step 2 :



Step 3 :

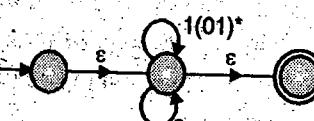
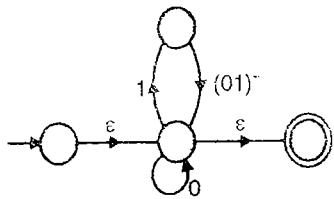


Fig. Ex. 3.2.2 : Contd....

Step 4 :



Step 5 :

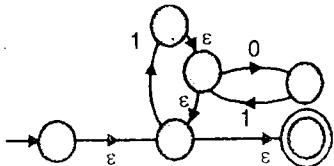
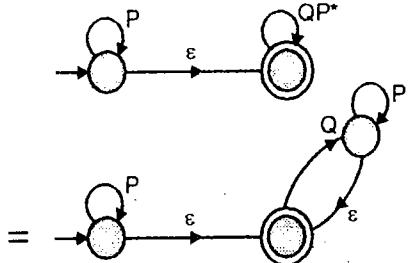


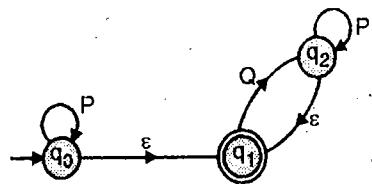
Fig. Ex. 3.2.2

Example 3.2.3Show that $P^*(QP^*)^* = (P + Q)^*$

Solution : A finite automata recognizing $P^*(QP^*)^*$ is given by :



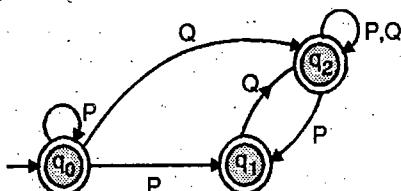
Naming the states, we get the Fig. Ex. 3.2.3.

Fig. Ex. 3.2.3 : A finite automata for $P^*(QP^*)^*$ ϵ -move from q_0 to q_1 can be removed by

1. Duplicating transitions of q_1 on q_0 .
2. Making q_0 as a final state.

 ϵ -move from q_2 to q_1 can be removed by

1. Duplicating transitions of q_1 on q_2 .
2. Making q_2 as a final state.

Fig. Ex. 3.2.3(a) : An equivalent finite automata without ϵ -moves

Since all the three states q_0 , q_1 and q_2 are final states (in Fig. Ex. 3.2.3(a)), they can be merged into a single state.

$$\begin{array}{c} \text{P}, \text{Q} \\ (q_0) \end{array} = (P + Q)^*$$

Fig. Ex. 3.2.3(b) : An equivalent DFA

Thus both expressions $P^*(QP^*)^*$ and $(P + Q)^*$ can be recognized by equivalent DFAs and hence we can infer that $P^*(QP^*)^* = (P + Q)^*$

Example 3.2.4 SPPU - Dec. 12, Dec. 13, 6 Marks

Prove or disprove the following for a regular expression.

- (i) $(r^*s^*)^* = (r+s)^*$ (ii) $R^* R = R^+$
 (iii) $(R^*)^* = R^*$

Solution :

- (i)
- $(r^*s^*)^* = (r+s)^*$

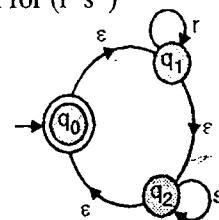
Step 1 : ϵ -NFA for $(r^*s^*)^*$ 

Fig. Ex. 3.2.4

Step 2 : ϵ -NFA to DFA ϵ -closure of $q_0 = \{q_0, q_1, q_2\}$ ϵ -closure of $q_1 = \{q_0, q_1, q_2\}$ ϵ -closure of $q_2 = \{q_0, q_1, q_2\}$

The equivalent DFA is given shown in Fig. Ex. 3.2.4(a).

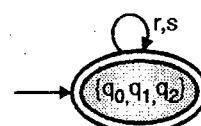


Fig. Ex. 3.2.4(a)

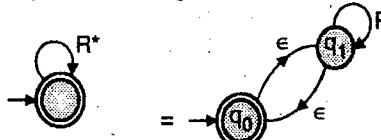
Step 3 : The regular expression for the DFA

$$= (r+s)^* = \text{R.H.S.}$$

- (ii)
- $R^* R = R^+$

$$\begin{aligned} R^* R &= \{\epsilon + R + RR + RRR + \dots\}R \\ &= \{R + RR + RRR + RRRR + \dots\} \\ &= R^+ \end{aligned}$$

- (iii)
- $(R^*)^* = R^*$
- : Drawing an FA for
- $(R^*)^*$
- , we get



Drawing an equivalent DFA, we get

 ϵ -closure of $q_0 = \{q_0, q_1\}$ ϵ -closure of $q_1 = \{q_0, q_1\}$

$$\therefore (R^*)^* = R^*$$





Syllabus Topic : Definition and Identities of Regular Expressions

3.3 Determination of Regular Expression

SPPU - May 15

University Question

Q. With examples define Regular Expression.
(May 2015, 2 Marks)

Formal definition : A regular expression over alphabet

$q = (a_1, a_2, \dots, a_n)$ is defined recursively as follows :

1. An empty set ϕ is a regular expression.
2. A null string ϵ is a regular expression.
3. An alphabet $a_i \in \Sigma$ is a regular expression.
4. If r_1 and r_2 are regular expression then :
 - (a) Their concatenation $r_1 \cdot r_2$ is a regular expression.
 - (b) Their union $r_1 + r_2$ is a regular expression.
 - (c) Closure of r_1 , i.e. r_1^* is a regular expression.
5. A regular expression is generated through a finite number of applications of above rules.

Syllabus Topic : Construction of Language from the RE

3.3.1 Language Generated by a Regular Expression

If r is a regular expression then the language generated by r is given by $L(r)$.

Example : If $r_1 = 0 + 1$ then $L(r_1) = \{0, 1\}$

If $r_2 = 0^*$ then $L(r_2) = \{\epsilon, 0, 00, 000, \dots\}$

If $r_3 = (0 + 1)^*$ then $L(r_3) = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$

3.3.2 Basic Properties of Regular Expressions

Some of the basic identities for regular expressions are given below. P, Q and R are regular expressions.

$$\phi + R = R \quad \dots(3.1)$$

$$\phi \cdot R = R \cdot \phi = \phi \quad \dots(3.2)$$

$$\epsilon \cdot R = R \cdot \epsilon = R \quad \dots(3.3)$$

$$\epsilon^* = \epsilon \quad \dots(3.4)$$

$$\phi^* = \epsilon \quad \dots(3.5)$$

$$R + R = R \quad \dots(3.6)$$

$$PQ + PR = P(Q + R) \quad \dots(3.7)$$

$$QP + RP = (Q + R)P \quad \dots(3.8)$$

$$R^* R^* = R^* \quad \dots(3.9)$$

$$RR^* = R^* R \quad \dots(3.10)$$

$$(R^*)^* = R^* \quad \dots(3.11)$$

$$\epsilon + RR^* = R^* \quad \dots(3.12)$$

$$(PQ)^* P = P(QP)^* \quad \dots(3.13)$$

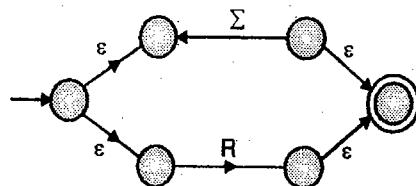
$$(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^* \quad \dots(3.14)$$

$$(P^* Q)^* = \epsilon + (P + Q)^* Q \quad \dots(3.15)$$

Proof for basic properties

$$1. \phi + R = R$$

Machine for $\phi + R$ is given by

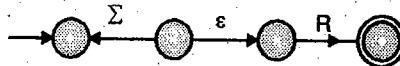


There is only one path from the start state to the final state, recognizing $\epsilon \cdot R \cdot \epsilon = R$.

$$\text{Hence, } \phi + R = R$$

$$2. \phi \cdot R = \phi$$

A composite machine for $\phi \cdot R$ is given by



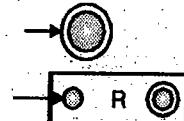
There is no way of reaching the final state from the start state.

$$\therefore \phi \cdot R = \phi$$

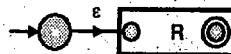
$$3. \epsilon \cdot R = R$$

FA for ϵ is given by

FA for R is given by



Composite FA for $\epsilon \cdot R$ is given by



This machine accepts R and only R .

$$\therefore \epsilon \cdot R = R$$

$$4. \epsilon + RR^* = R^*$$

$$\text{L.H.S.} = \epsilon + RR^*$$

$$= \epsilon + R\{\epsilon + R + RR + RRR + \dots\}$$

$$= \epsilon + R + RR + RRR + \dots = R^*$$

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

$$5. (PQ)^* P = P(QP)^*$$

$$\text{L.H.S.} = (PQ)^* P$$

$$= \{\epsilon + PQ + (PQ)^2 + (PQ)^3 + \dots\} P$$

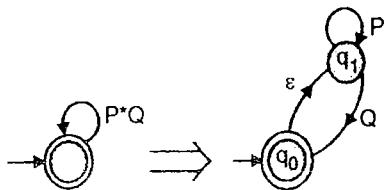
$$= \{\epsilon + PQ + PQPQ + PQPQPQ + \dots\} P$$



$$\begin{aligned} \text{L.H.S.} &= \{P + PQP + PQPQP + \dots\} \\ &= P \{\epsilon + QP + QPQP + \dots\} \\ &= P(QP)^* = \text{R.H.S.} \end{aligned}$$

6. $(P^*Q)^* = \epsilon + (P + Q)^*Q$

Finite automata for $(P^*Q)^*$ is given by :



ϵ -move can be removed by duplicating transitions of q_1 on q_0 .

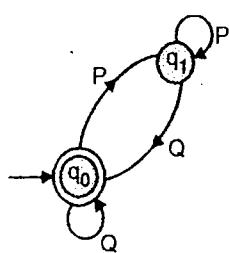
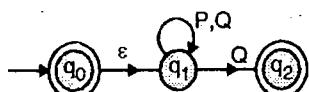
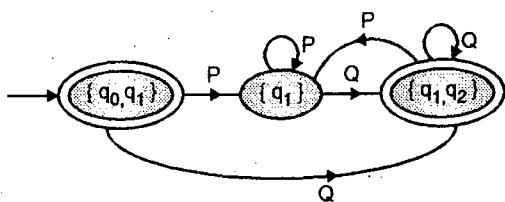


Fig. 3.3.1 : FA for $(P^*Q)^*$

Finite automata for $\epsilon + (P + Q)^*Q$ is given by



An equivalent DFA is being constructed using direct method.



Behaviour of both $\{q_0, q_1\}$ and $\{q_1, q_2\}$ are identical and they can be merged into a single state say q_0 .

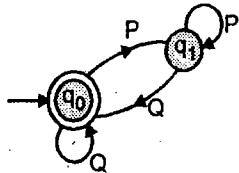


Fig. 3.3.2 : FA for $\epsilon + (P + Q)^*Q$

DFA for $(P + Q)^*$ given in Fig. 3.3.1 and $\epsilon + (P + Q)^*Q$ given in Fig. 3.3.2 are same.

$$\therefore (P^*Q)^* = \epsilon + (P + Q)^*Q.$$

Example 3.3.1

Give the examples of sets that demonstrates the following inequality. Here r_1, r_2, r_3 are regular expression :

(1) $r_1 + \epsilon \neq r_1$ (2) $r_1 \cdot r_2 \neq r_2 \cdot r_1$

(3) $r_1 \cdot r_1 \neq r_1$ (4) $r_1 + (r_2 \cdot r_3) \neq (r_1 + r_2) \cdot (r_1 \cdot r_3)$

Solution : Let us make the following assumptions

$$r_1 = 0 \quad r_2 = 1 \quad r_3 = 1$$

| Sr. No. | R.E. | L.H.S. of R.E. | R.H.S. of R.E. | Remark |
|---------|--|----------------|--------------------------------|----------------------|
| 1. | $r_1 + \epsilon \neq r_1$ | $0 + \epsilon$ | 0 | L.H.S. \neq R.H.S. |
| 2. | $r_1 \cdot r_1 \neq r_1$ | 0.0 | 0 | L.H.S. \neq R.H.S. |
| 3. | $r_1 \cdot r_2 \neq r_2 \cdot r_1$ | 0.1 | 1.0 | L.H.S. \neq R.H.S. |
| 4. | $r_1(r_2 \cdot r_3) \neq (r_1 + r_2)(r_1 \cdot r_3)$ | $0 + 10$ | $(0 + 1) \cdot 00 = 000 + 100$ | L.H.S. \neq R.H.S. |

Example 3.3.2

$$\text{Prove that } \epsilon + 1^*(011)^*(1^*(011))^* = (1 + 011)^*$$

Solution :

$$\begin{aligned} \text{L.H.S.} &= \epsilon + 1^*(011)^*(1^*(011))^* \\ &= \epsilon + RR^*, \text{ where } R \text{ is taken as } 1^*(011)^* \\ &= R^* \quad \dots [\text{By Equation (3.12)}] \\ &= (1^*(011))^* \\ &= (1 + 011)^* \quad \dots [\text{By Equation (3.14)}] \\ &= \text{R.H.S.} \end{aligned}$$

Example 3.3.3

$$\text{Prove that } (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$$

Solution :

$$\begin{aligned} \text{L.H.S.} &= (1 + 00^*1) \\ &\quad + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) \\ &= (1 + 00^*1) \\ &\quad [\epsilon + (0 + 10^*1)^*(0 + 10^*1)] \quad \dots [\text{By Equation (3.7)}] \\ &= (1 + 00^*1)(0 + 10^*1)^* \\ &\quad \dots [\text{By Equation (3.12)}] \\ &= [(\epsilon + 00^*)1](0 + 10^*1)^* \\ &\quad \dots [\text{By Equation (3.8)}] \\ &= 0^*1(0 + 10^*1)^* \\ &\quad \dots [\text{By Equation (3.12)}] \\ &= \text{R.H.S.} \end{aligned}$$

**Example 3.3.4**

Prove or disprove the given regular expression
 $(rs + r)^* = r(sr + r)^*$

Solution :

ϵ and rs are in $(rs + r)^*$

but ϵ and $rs \notin r(sr + r)^*$

Hence, $(rs + r)^* \neq r(sr + r)^*$

Example 3.3.5

If $s = \{aa, b\}$ write all the strings in s^* which are having length 4 or less, also say the following is true or false.

(i) $(s^+)^* = (s^*)^*$ (ii) $(s^+)^+ = s^+$ (iii) $(s^*)^+ = (s^+)^*$

Solution : We have to find all strings of length 4 or less in $(aa + b)^*$

(i) String of length 0 = $\{\epsilon\}$

(ii) String of length 1 = $\{b\}$

(iii) String of length 2 = $\{aa, bb\}$, taking either 'aa' one time or b two times.

(iv) String of length 3 = $\{bbb, bba, aab\}$, taking either b 3 times or b and aa one time each with permutations.

(v) String of length 4 = $\{aaaa, bbbb, bbaa, aabb, baab\}$,

Taking aa two times, taking b four times, taking aa one time & b two times with permutations.

\therefore Required strings are $\{\epsilon, b, aa, bb, bbb, baa, aab, aaaa, bbbb, bbaa, aabb, baab\}$

(i) $(s^+)^* = (s^*)^*$ – True, both $(s^+)^*$ and $(s^*)^*$ contain ϵ

(ii) $(s^+)^+ = s^+$ – True, both $(s^+)^+$ and s^+ do not contain ϵ

(iii) $(s^*)^+ = (s^+)^*$ – True, both $(s^*)^+$ and $(s^+)^*$ contain ϵ .

Example 3.3.6

Find all possible regular expression $L \subseteq \{a, b\}^*$

- the set of all strings ending in b.
- the set of all strings ending in ba.
- the set of all strings ending neither in b nor in ba.
- the set of all strings ending in ab
- the set of all strings ending neither in ab nor ba.

Solution :

(a) The set of all strings ending in b.

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

(b) The set of all strings ending in ba.

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

(c) The set of all strings ending neither in b nor in ba.

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

(d) The set of all strings ending in ab

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

(e) The set of all strings ending neither in ab nor ba.

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

Example 3.3.7

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

- The language of all strings containing exactly two 0's.
- The language of all strings containing at least two 0's.
- The language of all strings that do not end with 01.
- The language of all strings containing every 0 followed by 11.
- The language of all strings not containing the substring 00.

Solution :

(1) Strings containing exactly two 0's

$$R.E. = 1^*01^*01^*$$

(2) Strings containing at least two 0's

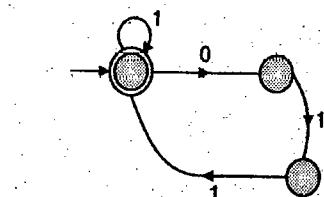
$$R.E. = 1^*01^*0(1 + 0)^*$$

(3) Strings not ending in 01.

$$R.E. = (1 + 0)^*(00 + 11 + 10)$$

(4) The language of all strings containing every 0 followed by 11.

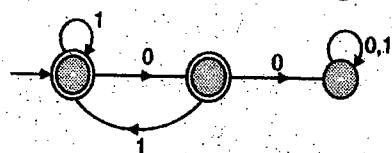
The DFA for the above language is given below



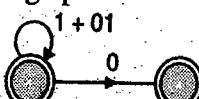
$$\therefore R.E. = (1 + 011)^*$$

(5) The language of all strings not containing the substring 00.

The DFA for the above language is given below.



The above DFA can be represented using an equivalent transition graph.



$$\begin{aligned} \therefore R.E. &= (1 + 01)^* [\epsilon + 0] \\ &= (1 + 01)^* + (1 + 01)^* 0 \end{aligned}$$

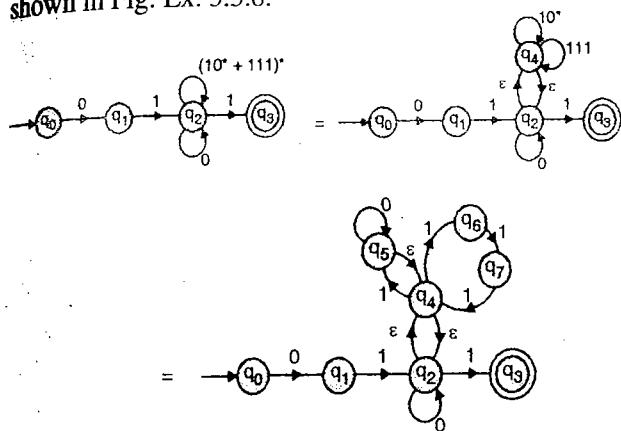
Example 3.3.8

Construct NFA with ϵ -moves for

R.E. = $0 \cdot 1 [(10^* + 111)^* + 0]^* 1$. Convert it to DFA using direct method of conversion from NFA with ϵ -moves to DFA.

Solution :

The ϵ -NFA is being constructed recursively as shown in Fig. Ex. 3.3.8.

Fig. Ex. 3.3.8 : ϵ -NFA

DFA is being constructed from ϵ -NFA. It is shown in Fig. Ex. 3.3.8(a).

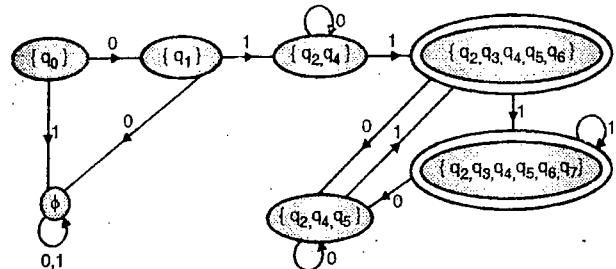


Fig. Ex. 3.3.8(a) : DFA

Example 3.3.9

Construct an NFA to accept the language represented by $a^*b^*c^*$. Convert the NFA to its equivalent DFA.

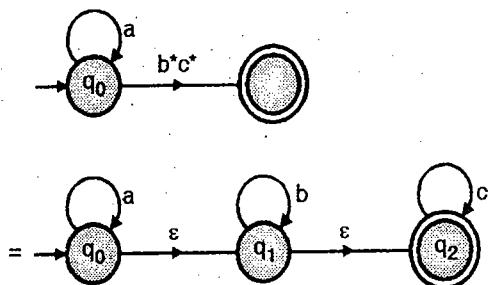
Solution :**Step 1 : RE to NFA**

Fig. Ex. 3.3.9(a)

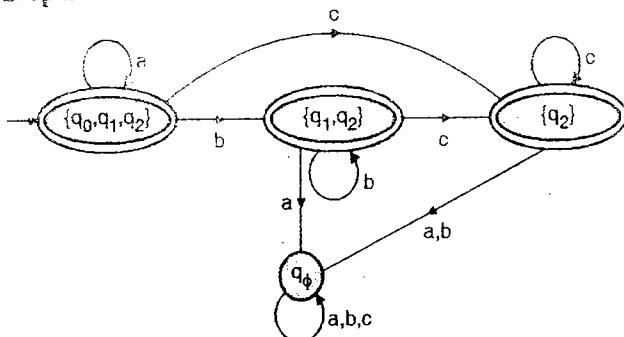
Step 2 : NFA to DFA

Fig. Ex. 3.3.9(b)

Example 3.3.10 SPPU - May 12, 8 Marks

Construct an NFA with ϵ -moves for the regular expression

- (i) $(b(aa)^*b + ab^*a)^*$
- (ii) $(ab + ba)^+$
- (iii) $(a + b)^*ab$.

Convert this NFA to its equivalent DFA.

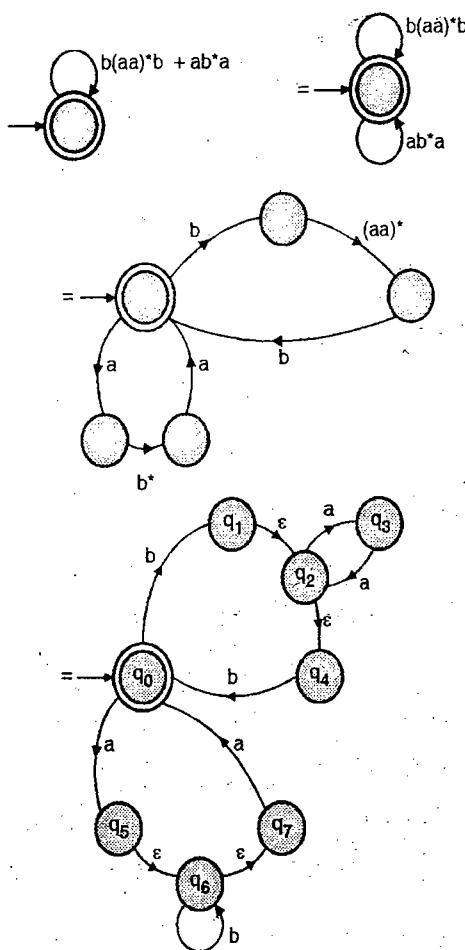
Solution :**(i) $(b(aa)^*b + ab^*a)^*$** **Step 1 : RE to NFA**

Fig. Ex. 3.3.10(a)

**Step 2 : NFA to DFA** ϵ -closure of states :

| State | ϵ -closure |
|-------|---------------------|
| q_0 | { q_0 } |
| q_1 | { q_1, q_2, q_4 } |
| q_2 | { q_2, q_4 } |
| q_3 | { q_3 } |
| q_4 | { q_4 } |
| q_5 | { q_5, q_6, q_7 } |
| q_6 | { q_6, q_7 } |
| q_7 | { q_7 } |

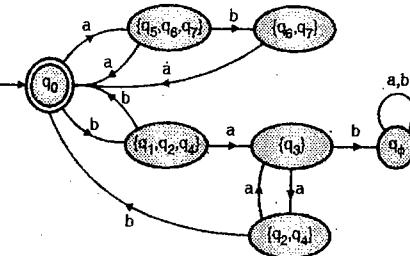


Fig. Ex. 3.3.10(b)

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

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

The required NFA is

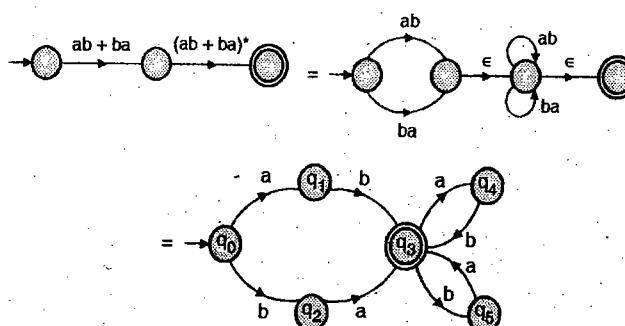


Fig. Ex. 3.3.10(c)

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

The required NFA is

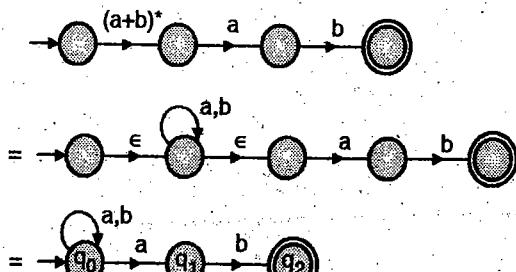


Fig. Ex. 3.3.10(d)

Example 3.3.11

Which of the following are true ? Explain.

- (i) $baa \in a^*b^*a^*b^*$ (iii) $a^*b^* \cap b^*c^* = \emptyset$
 (ii) $b^*a^* \cap a^*b^* = a^* \cup b^*$ (iv) $abcd \in (a(cd)^*b)^*$

Solution :

(i) $baa \in a^*b^*a^*b^*$

$$a^*b^*a^*b^* = (\epsilon + a + aa + \dots)(\epsilon + b + \dots) \\ (\epsilon + a + aa + \dots)(\epsilon + b + bb + \dots)$$

By taking ϵ -from the first set,
 b-from the second set
 aa from the third set
 and ϵ from the fourth set

We can say that, $baa \in a^*b^*a^*b^*$. Answer is True.

(ii) $b^*a^* \cap a^*b^* = a^* \cup b^*$

$b^*a^* = (\epsilon + bb^*)(\epsilon + aa^*)$

...[From Equation (3.12)]

$= \epsilon + aa^* + bb^* + bb^*aa^*$

$= (\epsilon + aa^*) + (\epsilon + bb^*) + bb^*aa^*$

$= a^* + b^* + bb^*aa^* \quad \dots(1)$

Similarly,

$a^*b^* = a^* + b^* + aa^*bb^* \quad \dots(2)$

Taking intersection of equations (1) and (2)

$b^*a^* \cap a^*b^* = (a^* + b^* + bb^*aa^*) \cap (a^* + b^* + aa^*bb^*)$

$= a^* + b^* = a^* \cup b^*$

Answer is True.

(iii) $a^*b^* \cap b^*c^* = \emptyset$

Both a^*b^* and b^*c^* contain the common string b^* .

Answer is False.

(iv) $abcd \in (a(cd)^*b)^*$

A string belonging to $(a(cd)^*b)^*$ can not end in cd.

Answer is False.

Example 3.3.12

For each of the following draw DFA of following regular expressions :

(a) $(11 + 00)^*$ (b) $(111 + 100)^*$

Solution :

(a) DFA for $(11 + 00)^*$

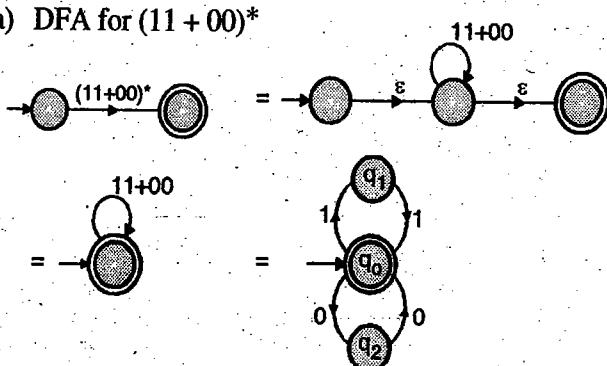


Fig. Ex. 3.3.12(a)

(b) DFA for $(111 + 100)^*0$

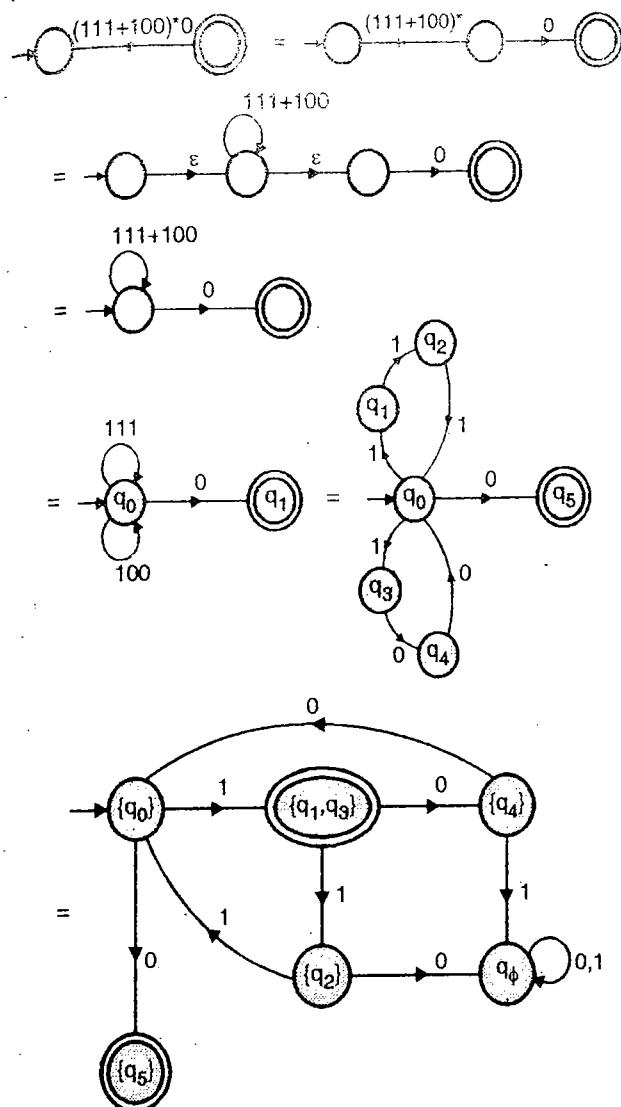


Fig. Ex. 3.3.12(b)

Example 3.3.13

Construct finite automata equivalent to the following regular sets.

(a) $10 + (0 + 11)0^*1$

(b) $01[((10)^* + 111)^* + 0]^*1$

(c) $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

Solution :

(a) FA for $10 + (0 + 11)0^*1$

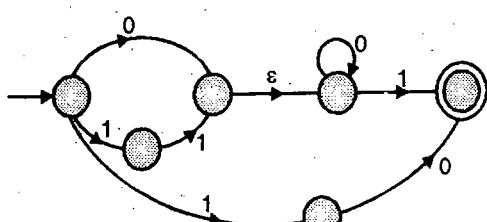


Fig. Ex. 3.3.13

(b) FA for $01[((10)^* + 111)^* + 0]^*1$

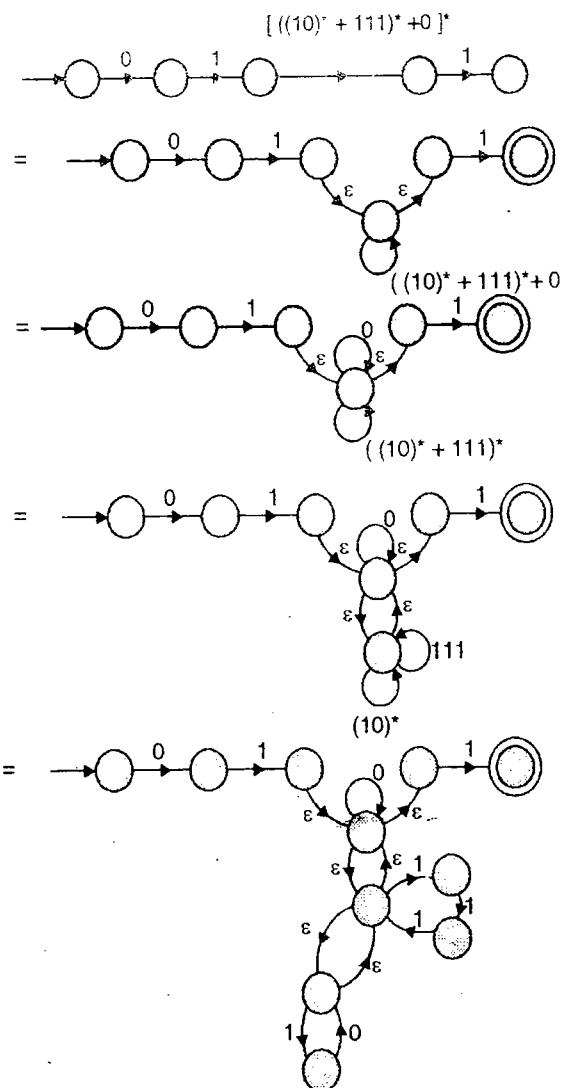


Fig. Ex. 3.3.13(a)

(c) FA for $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

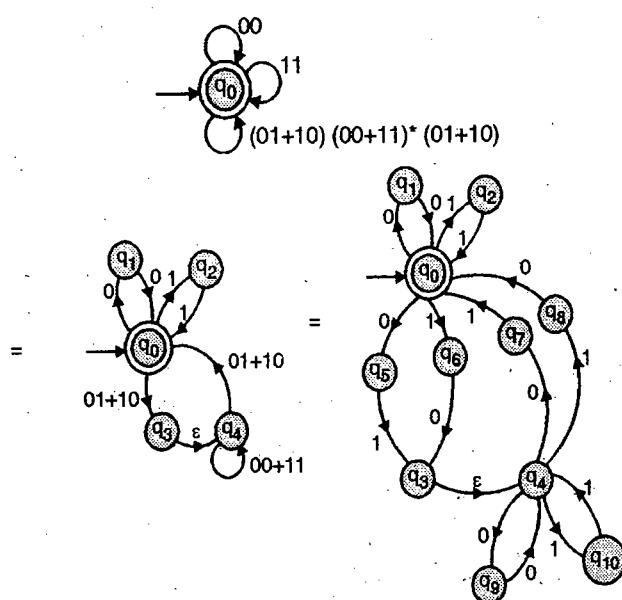


Fig. Ex. 3.3.13(b)

Example 3.3.14 SPPU - Dec. 15, 5 Marks

For each of the following regular expression, draw a finite automata recognizing the corresponding language.

1. $(1 + 10 + 110)^*0$
2. $1(01 + 10)^* + 0(11 + 10)^*$
3. $(010 + 00)^*(10)$
4. $1(1 + 10)^* + 10(0 + 01)^*$

Solution :

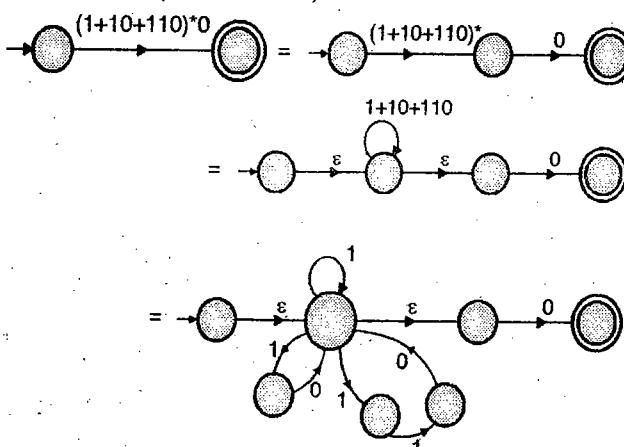
1. F.A. for $(1 + 10 + 110)^*0$ 

Fig. Ex. 3.3.14(a)

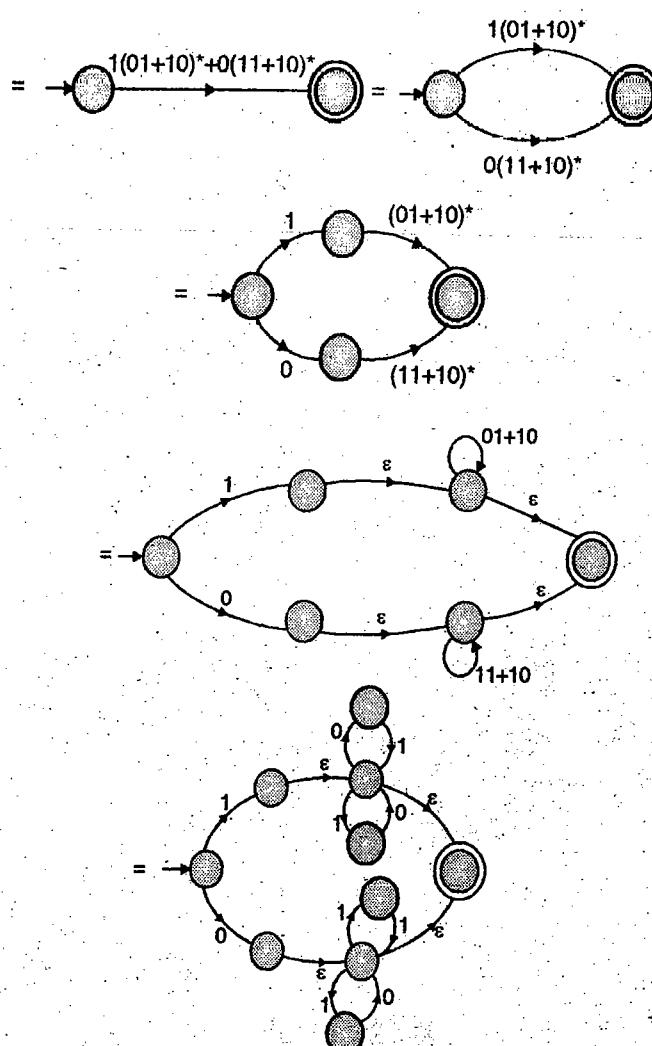
2. F.A. for $1(01 + 10)^* + 0(11 + 10)^*$ 

Fig. Ex. 3.3.14(b)

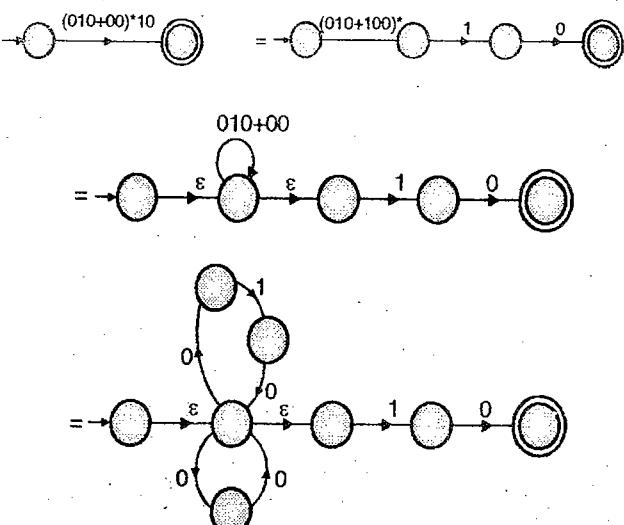
3. F.A. for $(010 + 00)^*(10)$ 

Fig. Ex. 3.3.14(c)

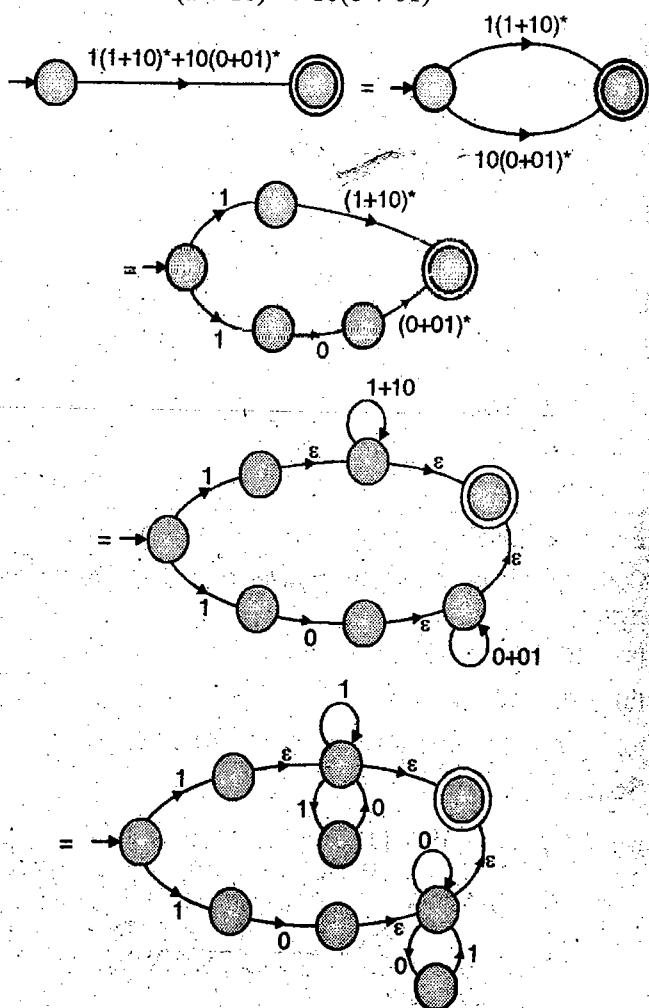
4. F.A. for $1(1 + 10)^* + 10(0 + 01)^*$ 

Fig. Ex. 3.3.14(d)

Example 3.3.15

Convert the following R.E. to DFA. (R.E. to NFA with ϵ -moves to DFA)

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

Solution :

The ϵ -NFA for $(ab + ba)^*$ is given by

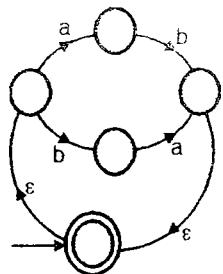
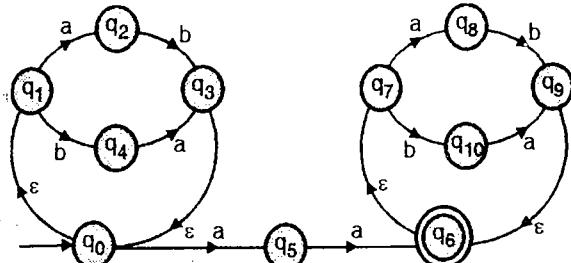


Fig. Ex. 3.3.15

Thus the ϵ -NFA for $(ab + ba)^*aa(ab + ba)^*$ can be drawn as shown in Fig. Ex. 3.3.15(a).

Fig. Ex. 3.3.15(a) : ϵ -NFA for the given example

DFA using the direct approach is shown in Fig. Ex. 3.3.15(b).

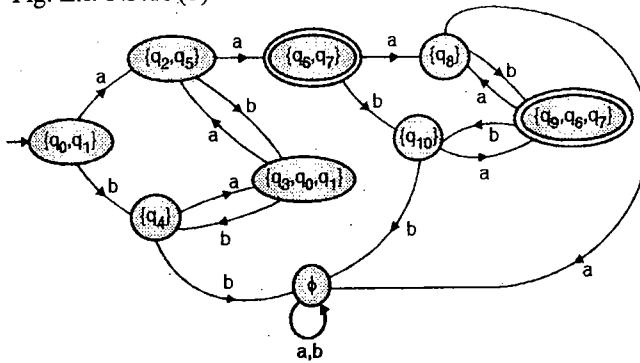


Fig. Ex. 3.3.15(b) : An equivalent DFA

Example 3.3.16

What language is represented by Regular expression : $((a^*a)b) \cup b$?

Solution : The regular set $((a^*a)b) \cup b$ can be written as

$$\begin{aligned} \text{R.E.} &= (a^*a)b + b = (a^*a + \epsilon)b \\ &= a^*b \quad [\text{From Equation (3.12)}] \end{aligned}$$

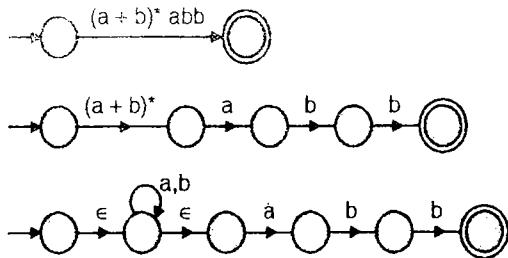
Thus the language represented by the regular expression is "zero or more a's followed by a 'b'".

Example 3.3.17

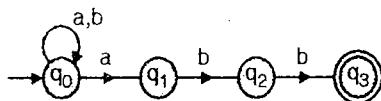
Construct DFA for following regular expression : RE = $(a + b)^* a b b$.

Solution :

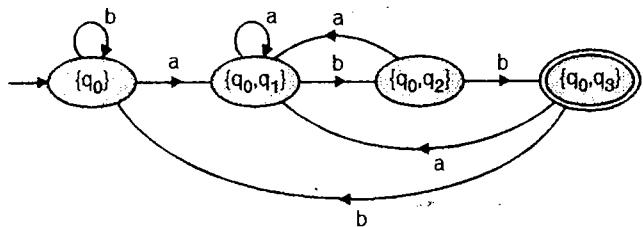
Step 1 : RE to ϵ -NFA



ϵ -transitions can be removed as the adjacent nodes do not contain a loop.



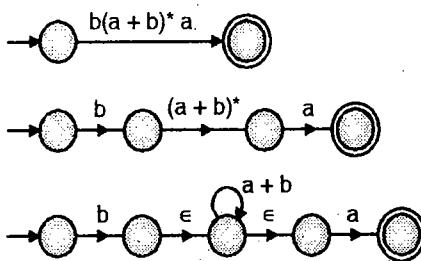
Step 2 : NFA to DFA using the direct method.

**Example 3.3.18**

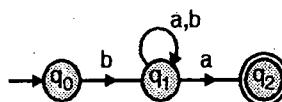
Construct DFA for following regular expression (RE) : RE = $b(a + b)^* a$.

Solution :

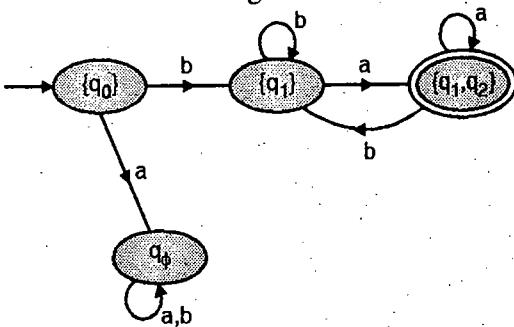
Step 1 : RE to ϵ -NFA



ϵ -transitions can be removed as the adjacent nodes do not contain loops.



Step 2 : NFA to DFA using the direct method.



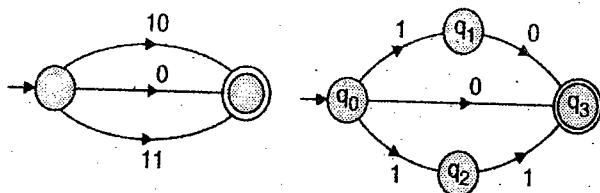
Example 3.3.19 SPPU - Dec. 16, 6 MarksConstruct DFA for the R.E : $10 + (0 + 11)^*$ **Solution :**Given RE = $10 + (0 + 11)^*$ **Step 1 :** Transition diagram

Fig. Ex. 3.3.19(a)

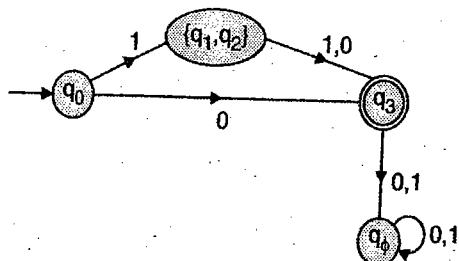
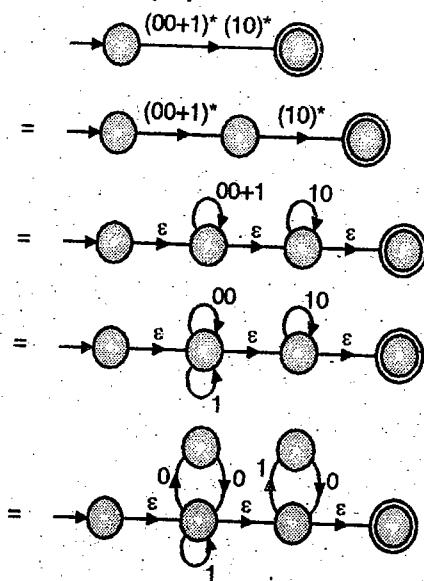
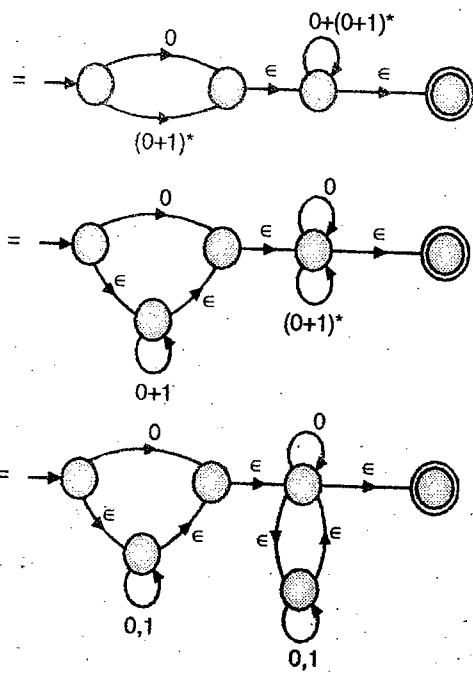
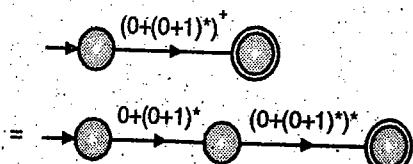
Step 2 : NFA to DFA using direct method.

Fig. Ex. 3.3.19(b)

Example 3.3.20Construct NFA from the following regular expressions : $(00 + 1)^* (10)^*$ **Solution :** $(00 + 1)^* (10)^*$ **Example 3.3.21**Construct NFA for the following regular expression : $(0 + (0 + 1)^*)^+$ **Solution :****Example 3.3.22**

Consider the two regular expression

$$r = 0^* + 1^*, s = 01^* + 10^* + 1^*0 + (01^*)^*$$

- Find the string corresponding to r but not to s.
- Find the string corresponding to s but not to r.
- Find the string corresponding to both r and s.

Solution :

- The string corresponding to r but not to s.
00 is in r but not in s
- The string corresponding to s but not to r.
01 is in s but not in r
- The string corresponding to both r and s.
11 is both in r and s.

Example 3.3.23 SPPU - Dec. 12, 6 Marks

Find all strings of length 5 or less in the regular set represented by the following –

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

Solution :

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

1. Strings of length 1String of length 0 from $(ab + a)^*$ followed by string of length 1 from $(aa + b)$

[b]

2. Strings of length 2Strings of length 0 from $(ab + a)^*$ and 2 from $(aa + b)$, string of length 1 from $(ab + a)^*$ and string of length 1 from $(aa + b)$

[aa, ab]



3. Strings of length 3

Strings of length 1 from $(ab + a)^*$ and 2 from $(aa + b)$, strings of length 2 from $(ab + a)^*$ and 1 from $(aa + b)$

[aaa, abb, aab]

4. Similarly strings of length 4 and 5

abaa, aaaa, abab, aabb, aaab,

abaaa, aabaa, aaaa,

ababb, abaab, aabab, aaabb

\therefore Strings of length 5 or less =

{ b, aa, ab, aaa, abb, aab, abaa, aaaa, abab, aabb, aaab, abaaa, aabaa, aaaaa, ababb, abaab, aabab, aaabb }

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

Required strings are strings of lengths 4 or less from $(a^* b + b^* a)^*$ followed by 'a'. It may be noted that $(a^* b + b^* a)^*$ is equal to $(a + b)^*$.

\therefore Required strings are

{ a, aa, ba, aaa, aba, baa, bba, aaaa, aaba, abaa, abba, baaa, baba, bbaa, bbba, aaaaa, aaaba, aabaa, aabba, abaaa, ababa, abbaa, baaaa, baaba, babaa, babba, bbaaa, bbaba, bbbba, bbbbba }

(iii) $a^* + (ab + a)^*$

Required strings are strings of length 5 or less in a^* + strings of length 5 or less in $(ab + a)^*$

= { ϵ , a, aa, aaa, aaaa, aaaaa, ab, aab, aba, abab, abaa, aaba, aaab, aabab, abaab, ababa, aaaab, aabaa, aaaba, aaaab }

Example 3.3.24 SPPU - Dec. 13, 4 Marks

Show that :

$$(1) (a + b)^* = (a + b)^* + (a + b)^*$$

$$(2) (a \cdot b)^* \neq a^* \cdot b^*$$

Solution :

$$(1) (a + b)^* = (a + b)^* + (a + b)^*$$

Let $L = (a + b)^*$

$$\text{Now, } (a + b)^* + (a + b)^* = L + L = L$$

[union of two identical language]

$$\therefore (a + b)^* = (a + b)^* + (a + b)^*$$

$$(2) (a \cdot b)^* \neq a^* \cdot b^*$$

aabb $\in a^* \cdot b^*$ but aabb $\notin (a \cdot b)^*$

$$\therefore (a \cdot b)^* \neq a^* \cdot b^*$$

Example 3.3.25 SPPU - May 16, 6 Marks

Describe in simple English the language defined by the following RE :

- (i) $(a + b)^* a (a + b)^*$
- (ii) $(01^* 0)^* 1$
- (iii) $a (a + b)^* bb$

Solution :

- (i) Strings containing 'a'.
- (ii) Strings ending with 1 and containing even number of 0's
- (iii) Strings starting with 'a' and ending with 'bb'.

Example 3.3.26

Describe in English the language represented by following expressions :

- (i) $(a + ab)^*$
- (ii) $(a + b)^* a (a + b)^*$
- (iii) $(a^* ab^* ab^*) + b^*$
- (iv) $a^* b^* c^*$

Solution :

- (i) $(a + ab)^*$: \rightarrow The language is "Every b is preceded by an a".
- (ii) $(a + b)^* a (a + b)^*$: \rightarrow The language is "The string contains at least one a".
- (iii) $(a^* ab^* ab^*) + b^*$: The language is "The string contains at least two a's or a string of only b's including null string".
- (iv) $a^* b^* c^*$: The language is "One or more a's followed by any number of b's followed by one or more number of c's".

3.4 DFA to Regular Expression

There are two popular approaches for constructing a regular expression from FA.

(1) Through state/loop elimination

(2) Arden's theorem.

Syllabus Topic : Construction of FA from the given RE using Direct Method

3.4.1 State/Loop Elimination Process

Every state $q_i \notin F$ (set of final states) can be eliminated if q_i is not a start state. Elimination of a state involves writing of regular expression as label on arcs.

For example,

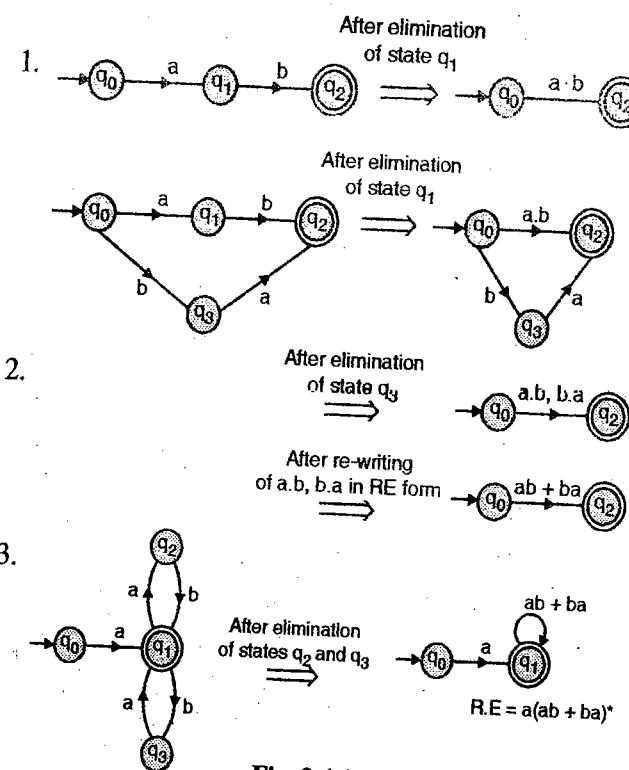


Fig. 3.4.1

3.4.1.1 A Generic One State Machine

The regular expression for strings accepted by a one state generic machine is R^* .

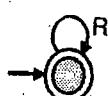


Fig. 3.4.2 : A generic one-state machine

3.4.1.2 A Generic Two State Machine

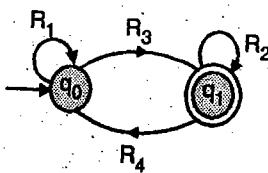


Fig. 3.4.3 : A generic two-state machine

- A regular expression for a two state generic machine can be written easily after elimination of loop between q_0 and q_1 .
- The effect of the loop can be moved either to state q_0 or q_1 .
- Even after elimination of loop between q_0 and q_1 , a path from start state (q_0) to final state (q_1) should be maintained.

Machine after moving the effect of loop to state q_0

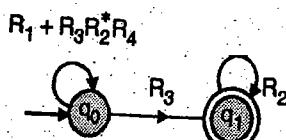


Fig. 3.4.4 : Effect of loop is moved to state q_0

Starting from the start state q_0 , one can come back to q_0 either on R_1 , or $R_3 R_2^* R_4$

The equivalent R.E. for machine shown in Fig. 3.4.4 can be written easily. It is given by

$$R.E. = (R_1 + R_3 R_2^* R_4)^* R_3 R_2^*$$

Machine after moving the effect of loop to state q_1

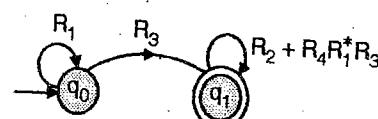


Fig. 3.4.5 : Effect of loop is moved to state q_1

Starting from the state q_1 , one can come back to q_1 either on R_2 ,

$$\text{or } R_4 R_1^* R_3$$

The equivalent R.E. for machine shown in Fig. 3.4.5 can be written easily. It is given by :

$$R.E. = R_1^* R_3 (R_2 + R_4 R_1^* R_3)^*$$

Example 3.4.1

Find a regular expression for the given two state generic machine.

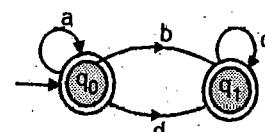


Fig. Ex. 3.4.1(a) : A two state generic machine

Solution :

Neither state q_0 nor state q_1 can be eliminated as both are final states. Effect of the loop between q_0 and q_1 can be moved either to state q_0 or state q_1 . Moving the effect of loop between q_0 and q_1 on state q_0 , we get a machine as shown in Fig. Ex. 3.4.1(b). The above machine has two final states. The machine can be divided into two machines, first with q_0 as final state and second with q_1 as final state. Machine after division is shown in Fig. Ex. 3.4.1(c).

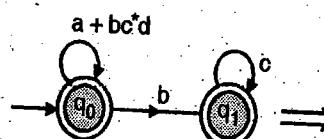


Fig. Ex. 3.4.1(b)

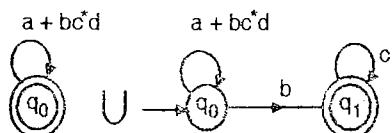


Fig. Ex. 3.4.1(c) : Union of two machines

R.E. corresponding to final state $q_0 = (a + bc^*d)^*$

R.E. corresponding to final state $q_1 = (a + bc^*d)^*bc^*$

The final, regular expression will be obtained by taking union of R.E. for state q_0 and q_1 .

$$\begin{aligned} \therefore \text{Final R.E.} &= (a + bc^*d)^* + (a + bc^*d)^*bc^* \\ &= (a + bc^*d)^*[\epsilon + bc^*] \end{aligned}$$

Example 3.4.2

Convert the automata shown in Fig. Ex. 3.4.2 to RE

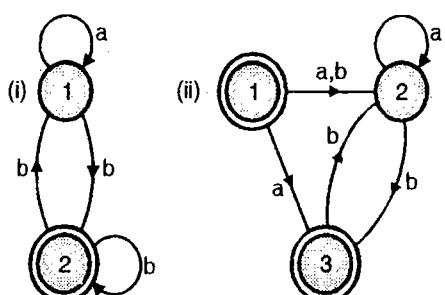


Fig. Ex. 3.4.2

Solution :

- (i) Machine is redrawn after the effect of loop between state 1 and 2 is transferred to state 1.

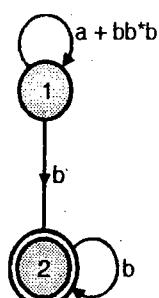


Fig. Ex. 3.4.2(a)

$$\therefore \text{R.E.} = (a + bb^*b)^*b^*b^*$$

- (ii) Machine is redrawn after the effect of loop between state 2 and 3 is transferred to state 2.

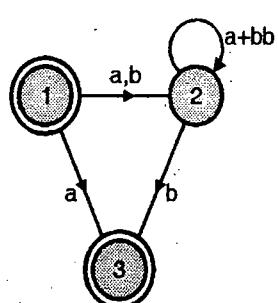


Fig. Ex. 3.4.2(b)

$$\text{R.E.} = \epsilon + (a + b)(a + bb)^*b + a$$

for state 1

for state 3

Example 3.4.3

Find R.E. corresponding to T.G. shown in Fig. Ex. 3.4.3.

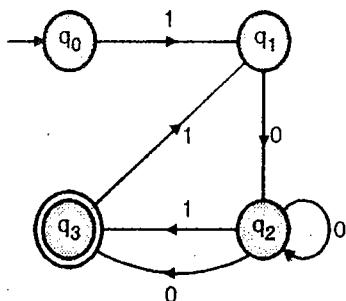


Fig. Ex. 3.4.3

Solution : Reducing the given T.G.

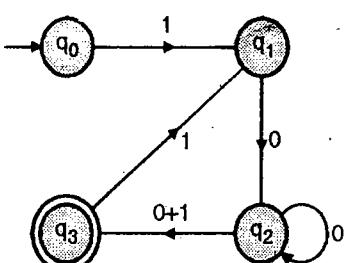


Fig. Ex. 3.4.3(a)

Now, eliminating the state q_2 , we get,

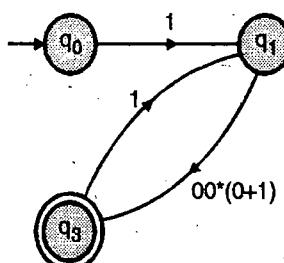


Fig. Ex. 3.4.3(b)

Machine is redrawn after the effect of loop between q_1 and q_3 is transferred to q_1 .

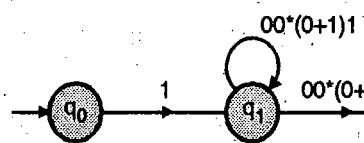
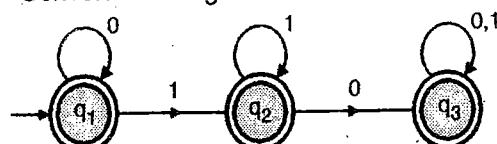


Fig. Ex. 3.4.3(c)

$$\therefore \text{Final R.E.} = 1[00^*(0+1)]^* 00^*(0+1)$$

**Example 3.4.4 SPPU - Dec. 13, 6 Marks**

- (a) Find R.E. for shown in Fig. Ex. 3.4.4 FA
 (b) Convert following DFA to RE.



(a)

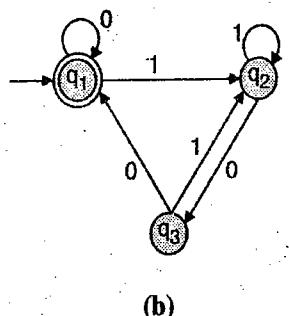


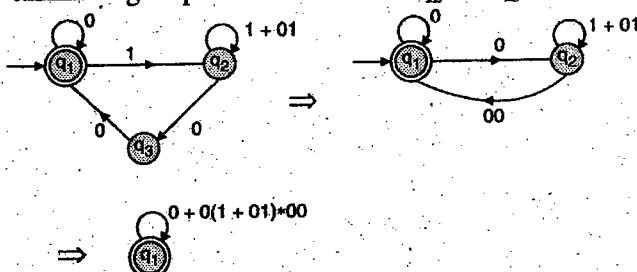
Fig. Ex. 3.4.4

Solution :**(a)**R.E. for state $q_1 = 0^*$ R.E. for state $q_2 = 0^*11^*$ R.E. for state $q_3 = 0^*11^*0(0+1)^*$

$$\therefore \text{Final RE} = 0^* + 0^*11^* + 0^*11^*0(0+1)^*$$

(b)

We can draw an equivalent transition graph by eliminating loop between the states q_2 and q_3 .



$$\therefore \text{R.E.} = [0 + 0(1 + 01)^*00]^*$$

Example 3.4.5

Consider the transition diagram shown in Fig. Ex. 3.4.5. Convert it to equivalent RE.

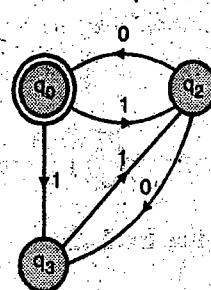


Fig. Ex. 3.4.5

Solution :

Machine is redrawn after the effect of loop between q_0, q_3, q_2 is transferred to q_0 .

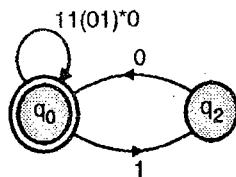


Fig. Ex. 3.4.5(a)

Now, removing the loop between q_0 and q_2 ,

$$\therefore \text{Final RE} = [11(01)^*0 + 1(01)^*0]^*$$

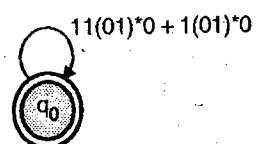


Fig. Ex. 3.4.5(b)

Example 3.4.6

Find a regular expression for the given three state machine shown in Fig. Ex. 3.4.6.

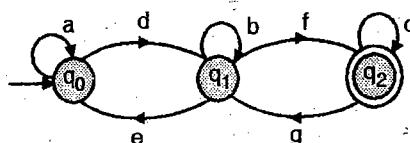
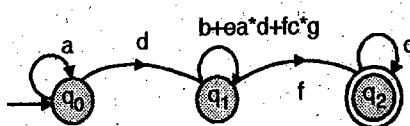
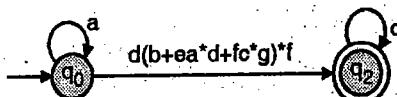


Fig. Ex. 3.4.6

Solution : The state q_1 is at the crossing point of three loops:

1. q_1 to q_1 on input b.
2. Loop between q_1 and q_0 on input ea*d
3. Loop between q_1 and q_2 on input fc*g.

The effect of these three loops can be transferred to state q_1 ; the machine after this effect is shown in Fig. Ex. 3.4.6(a).

Fig. Ex. 3.4.6(a) : Moving the effect of loops on state q_1 Fig. Ex. 3.4.6(b) : After elimination of state q_1

The state q_1 can be eliminated.

Now, the regular expression can be written as,

$$\text{R.E.} = a^*d(b+ea^*d+fc^*g)^*fc^*$$

Example 3.4.7

Find the regular expression for the DFA's, shown in Fig. Ex. 3.4.7.

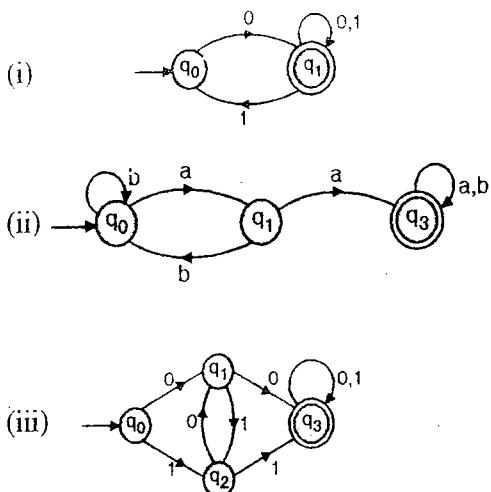


Fig. Ex. 3.4.7

Solution :

- (i) Machine is redrawn after the effect of loop between q_0 and q_1 is transferred to q_0 .

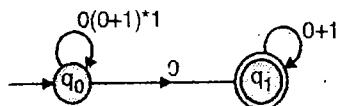


Fig. Ex. 3.4.7(a)

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

- (ii) Machine is re-drawn after the effect of loop between q_0 and q_1 is transferred to q_0

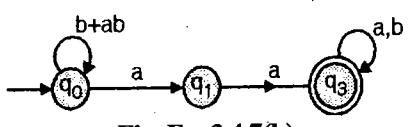


Fig. Ex. 3.4.7(b)

$$\text{R.E.} = (b+ab)^*aa(a+b)^*$$

- (iii) Machine is redrawn after the effect of loop between q_1 and q_2 is transferred to both q_1 and q_2 .

There are two paths from q_1 and q_3 [0 + 11]. There are two paths from q_2 to q_3 [1 + 00].

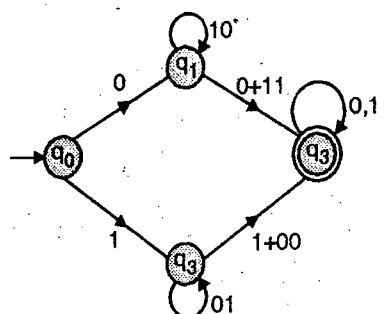


Fig. Ex. 3.4.7(c)

$$\text{R.E.} = [0(10)^*(0+11) + 1(01)^*(1+00)](0+1)^*$$

Example 3.4.8 SPPU - Aug. 15 (In Sem), 4 Marks

Find the regular expression corresponding to each of the following subset of $\{0, 1\}$: Language of all Strings containing an even no. of 0's.

Solution :

Language of all Strings containing an even no. of 0's

An FA for strings containing even number of 0's is given below.

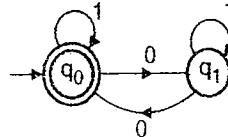


Fig. Ex. 3.4.8

An equivalent transition graph can be drawn by removing loop between the two states q_0 and q_1 .

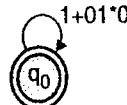


Fig. Ex. 3.4.8(a)

Now, we can write the R.E. which is $(1 + 01^*0)^*$

Example 3.4.9 SPPU - Aug. 15 (In Sem), 6 Marks

Construct a regular expression for given finite automata.

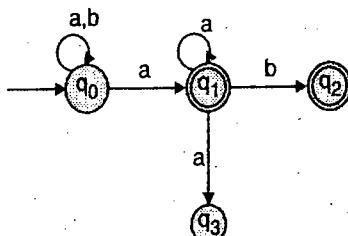


Fig. Ex. 3.4.9

Solution :

Strings accepted by the state q_1 is given by $(a+b)^* aa^*$

Strings accepted by the state q_2 is given by

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

∴ Strings accepted by the above automata is represented by the R.E.

$$(a+b)^* aa^* [\in + b]$$

Example 3.4.10 : Prove the identity given below :

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

Solution :

We can start with construction of NFA for $(a + ab + ba)^*$ and then convert it into a DFA.

Step 1 : $(a + ab + ba)^*$ to NFA. An equivalent NFA is given in Fig. Ex. 3.4.10.

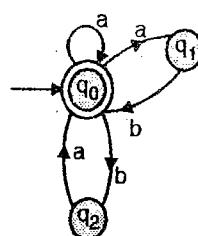


Fig. Ex. 3.4.10 : An equivalent NFA

Step 2 : NFA to DFA : An equivalent DFA is given in Fig. Ex. 3.4.10(a).

| | a | b |
|-------------------------|----------------|----------------|
| $\rightarrow \{q_0\}^*$ | $\{q_0, q_1\}$ | $\{q_2\}$ |
| $\{q_2\}$ | $\{q_0\}$ | \emptyset |
| $\{q_0, q_1\}^*$ | $\{q_0, q_1\}$ | $\{q_0, q_2\}$ |
| $\{q_0, q_2\}^*$ | $\{q_0, q_1\}$ | $\{q_2\}$ |

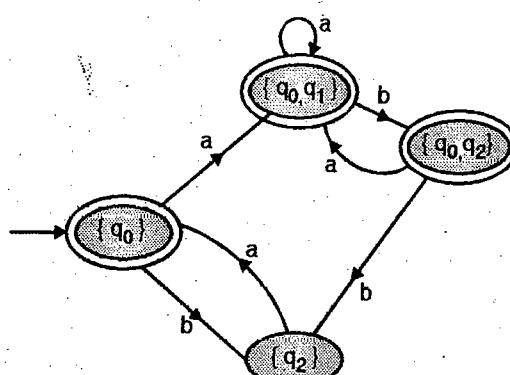


Fig. Ex. 3.4.10(a) : An equivalent DFA

Step 3 : Minimization of DFA obtained from Fig. Ex. 3.4.10(a).

Behaviour of $\{q_0\}$ and $\{q_0, q_2\}$ are identical as their a-successors and b-successors are same. Merging the two states $\{q_0\}$ and $\{q_0, q_2\}$, we get, the minimum state DFA as shown in Fig. Ex. 3.4.10(b).

| | a | b |
|----------------------------------|---|-------------|
| $\{q_0, q_2\} = \rightarrow A^*$ | B | C |
| $\{q_0, q_1\} = B^*$ | B | A |
| $\{q_2\} = C$ | A | \emptyset |

Fig. Ex. 3.4.10(b) : An equivalent minimal DFA

Step 4 : Writing of regular expression for DFA derived from Fig. Ex. 3.4.10(b).

- There are two loops on A.

 1. Loop through $A \rightarrow C \rightarrow A$ on input ba
 2. Loop through $A \rightarrow B \rightarrow B \rightarrow \dots \rightarrow B \rightarrow A$ on input aa^*b

3. There are two final states.

An equivalent machine without a loop is shown in Fig. Ex. 3.4.10(c).

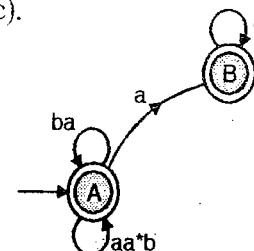


Fig. Ex. 3.4.10(c) : An equivalent machine without a loop

Regular expression for the machine shown in Fig. Ex. 3.4.10(c) is given by :

$$\text{R.E.} = \underbrace{(aa^*b + ba)^*}_{\text{Corresponding to state A}} + \underbrace{(aa^*b + ba)^*aa^*}_{\text{Corresponding to state B}}$$

$$= (aa^*b + ba)^* [\epsilon + aa^*]$$

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

... [By Equation (3.12)]

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

... [By Equation (3.10)]

= L.H.S.

Example 3.4.11

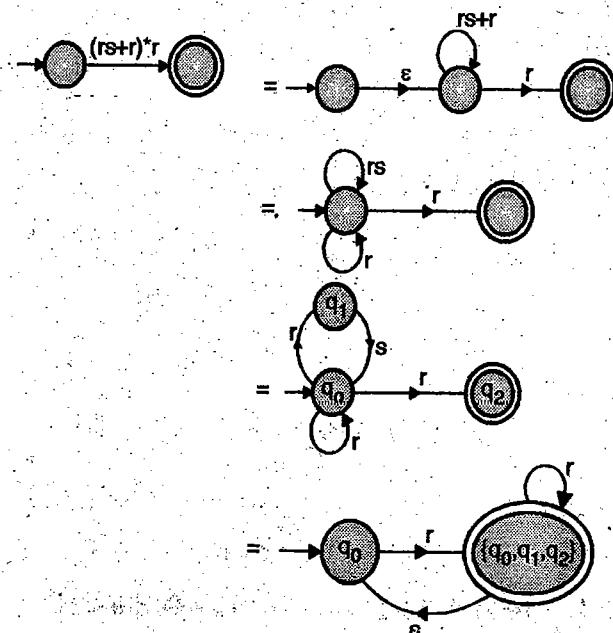
Prove or disprove the following for regular expression r, s and t.

- $(rs + r)^*r = r(sr + r)^*$
- $s(rs + s)^*r = rr^*s(r^*s)^*r$
- $(r + s)^* = r^* + s^*$

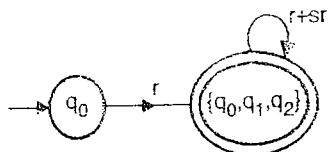
Solution :

(a) $(rs + r)^*r = r(sr + r)^*$

This can be proved by drawing a F.A. for $(rs + r)^*r$



Moving the effect of the loop on $\{q_0, q_1, q_2\}$, we get,



The regular expression for the above FA is given by $r(sr + r)^*$ = R.H.S.

Hence, it is proved.

(b) $s(rs + s)^*r = rr^*s(rr^*s)^*r$

L.H.S. is starting with s, whereas R.H.S. starts with r. Hence, disproved.

(c) $(r + s)^* = r^* + s^*$

The expression rs belongs to $(r + s)^*$ but it does not belong to $r^* + s^*$, hence disproved.

Example 3.4.12 SPPU - Aug. 15 (In Sem), 2 Marks

Find the regular expression corresponding to each of the following subset of $\{0, 1\}^*$

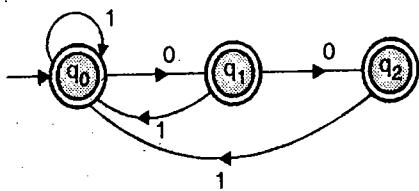
- The language of all strings not containing the substring 000.
- The language of all strings that do not contain the substring 110.
- The language of all strings containing both 101 and 010 as substring.

Solution :

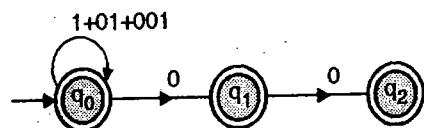
- The language of all strings not containing the substring 000.

R.E. can be written by drawing an equivalent FA and then writing a regular expression for it.

DFA for strings not containing 000 is given by,



Moving the effect of loops on state q_0 we get,

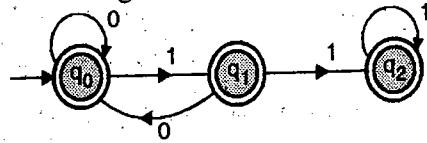


$$\therefore \text{R.E.} = (1 + 01 + 001)^*(\epsilon + 0 + 00)$$

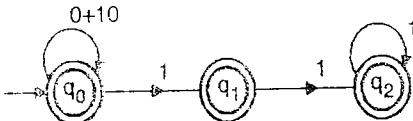
- The language of all strings that do not contain the substring 110.

R.E. can be written by drawing an equivalent FA and then writing a regular expression for it.

DFA for strings not containing 110 is given by,



Moving the effect of loop on state q_0 , we get,



$$\therefore \text{R.E.} = (0 + 10)^* + (0 + 10)^*1 \\ + (0 + 10)^*111^*$$

- The language of all strings containing both 101 and 010 as substring.

$$\therefore \text{R.E.} = (0 + 1)^*101(0 + 1)^*010(0 + 1)^* \\ + (0 + 1)^*010(0 + 1)^*101(0 + 1)^*$$

Example 3.4.13 SPPU - May 15, 4 Marks

Let $\Sigma = \{a, b\}$. Write RE to define language consisting of strings such that.

- Strings without substring bb
- Strings that have exactly one double letter in them.

Solution :

- Strings without substring bb

Equivalent FA is shown below.

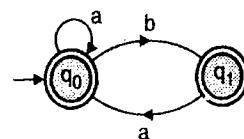


Fig. Ex. 3.4.13(a)

The effect of loop between states q_0 and q_1 can be moved to the state q_0 .

$$\text{R.E. for state } q_0 = (a + ba)^*$$

$$\text{R.E. for state } q_1 = (a + ba)^* b$$

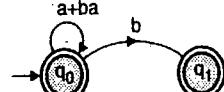


Fig. Ex. 3.4.13 (b)

$$\therefore \text{Equivalent R.E.} = (a + ba)^* + (a + ba)^* b \\ = (a + ba)^* + (a + ba)^* b \\ = (a + ba)^* (\epsilon + b)$$

- Strings that have exactly one double letter in them.

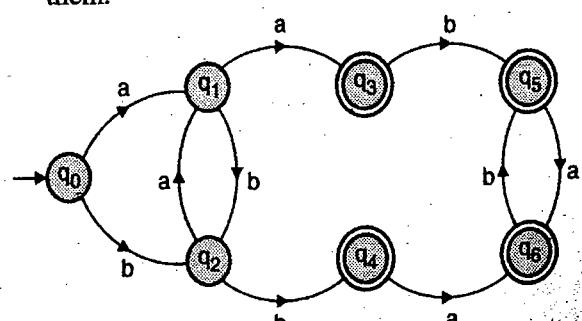


Fig. Ex. 3.4.13(c)



Strings associated with state q_3 are $a(ba)^*$, $a + b(ab)^*$. Strings associated with state q_4 are $b(ab)^*$, $b + a(ba)^*$. Hence, the equivalent transition graph may be drawn as given Fig. Ex. 3.4.13(d)

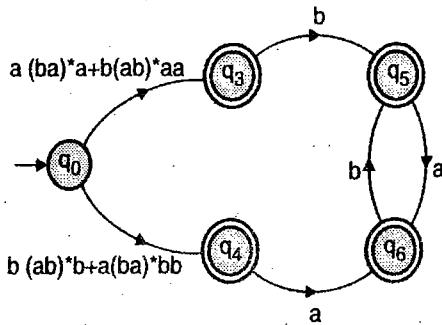


Fig. Ex. 3.4.13(d)

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

Example 3.4.14

Write RE for the following $\Sigma = \{0, 1\}$

- (i) At most 1 pair of consecutive '0' or 1 pair of consecutive '1'.
- (ii) Without any double letter.

Solution :

- (i) At most 1 pair of consecutive '0' or 1 pair of consecutive '1'.

A finite automata with at most 1 pair of consecutive 0's is shown in Fig. Ex. 3.4.14.

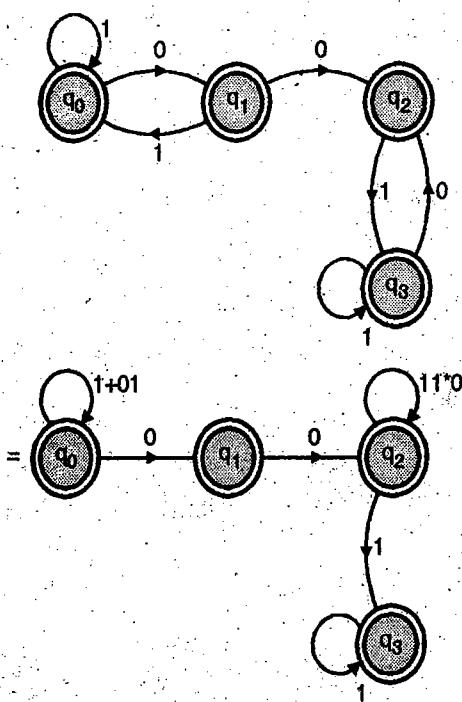


Fig. Ex. 3.4.14

\therefore R.E. for strings with at most 1 pair of consecutive 0's is

$$(1 + 01)^* [\epsilon + 0 + 00 (11^*0)^* + 00 (11^*0)^* 11^*]$$

\therefore R.E. for strings with at most 1 pair 1's is

$$(0 + 10)^* [\epsilon + 1 + 11 (00^*1)^* + 11 (00^*1)^* 00^*]$$

\therefore Required R.E. is

$$(1 + 01)^* [\epsilon + 0 + 00 (11^*0)^* + 00 (11^*0)^* 11^*] + (0 + 10)^* [\epsilon + 1 + 11 (00^*1)^* + 11 (00^*1)^* 00^*]$$

- (ii) Without any double letter

A Finite Automata without any double letter is shown in Fig. Ex. 3.4.14(a).

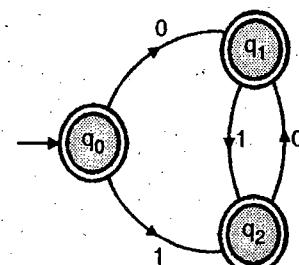


Fig. Ex. 3.4.14(a)

$$\therefore \text{R.E.} = \epsilon + 0(10)^* [\epsilon + 1] + 1(01)^* [\epsilon + 0]$$

Example 3.4.15

Write RE for the following languages :

- (i) The language of all strings that contains at least one occurrence of each symbol in $\Sigma = \{0, 1\}$
- (ii) $\Sigma = \{a, b\}$ such that ab is not a substring of any strings.

Solution :

1. The FA for language of all strings containing at least one occurrence of each symbol in $\Sigma = \{0, 1\}$ is shown in Fig. Ex. 3.4.15(a) :

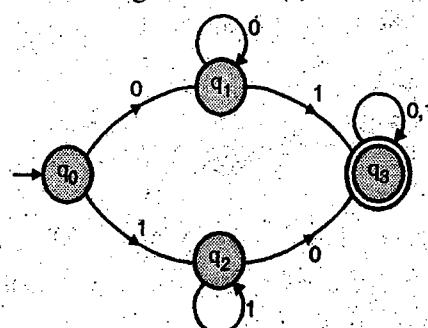


Fig. Ex. 3.4.15(a)

The RE can be written from the DFA.

$$\text{R.E.} = 00^*1(0+1)^* + 11^*0(0+1)^*$$

2. The DFA for language of all strings not containing 'ab' as a substring is shown in Fig. Ex. 3.4.15(b) :

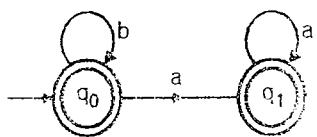


Fig. Ex. 3.4.15(b)

$$\therefore \text{R.E.} = b^* + b^*aa^*$$

Example 3.4.16

Write a R.E. for the following

- (i) $\Sigma = \{0, 1\}$ odd number of 1's in strings
- (ii) $\Sigma = \{0, 1\}$ Triple 0 must never appear in strings.
- (iii) Identifiers in C language.

Solution :

- (i) $\Sigma = \{0, 1\}$ odd number of 1's in strings.

In some cases it is more convenient to draw a DFA for the given language and then an equivalent regular expression can be written for it.

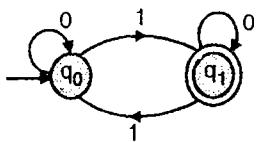
Step 1 : DFA for the given language

Fig. Ex. 3.4.16(a) : DFA for strings having odd number of 1's

An equivalent FA with effect of loop between q_0 and q_1 moved to q_0 .

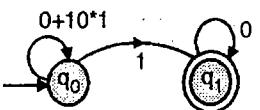


Fig. Ex. 3.4.16(b) : An equivalent machine of DFA given in Fig. Ex. 3.4.16(a)

Step 2 : Writing of R.E. from Fig. Ex. 3.4.16(b).

$$\text{R.E.} = (0 + 10^*1)^*10^*$$

- (ii) $\Sigma = \{0, 1\}$ Triple 0 must never appear in string.

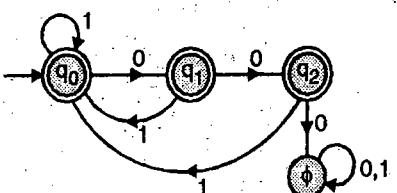
Step 1 : Drawing an equivalent DFA.

Fig. Ex. 3.4.16(c) : DFA for strings not having 000

An equivalent FA with effect of loops moved to q_0 is shown in Fig. Ex. 3.4.16(d).

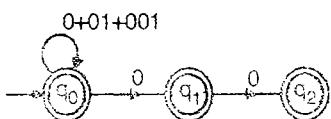


Fig. Ex. 3.4.16(d) : An equivalent FA for DFA of Fig. Ex. 3.4.16(c)

Step 2 : Writing of R.E.

$$\text{R.E.} = (1 + 01 + 001)^*(\epsilon + 0 + 00)$$

- (iii) Identifiers in C language.

$$\text{R.E.} = (A + B + \dots + Z + a + b + z) \\ (A + B + \dots + Z + a + \dots + z + 0 + \dots + 9)^*$$

Example 3.4.17 [SPPU - Dec. 12, 6 Marks]

- (a) Represent the following using regular expressions.

- (i) $\Sigma = \{a, b, c\}$ the language such that "any number of a's followed by any member of c's"
- (ii) If $L(r) = \{0, 2, 201, 21, 011, 211, 0111, \dots\}$ then what is r ?
- (iii) If $L(r) = \{00, 010, 0110, 01110, \dots\}$ then what is r ?
- (iv) Languages defined over $\Sigma = \{a, b\}$ has to have the strings beginning with 'a' and not have two consecutive a's. Write the regular expression for the same.

- (b) Give RE for the following languages over $\Sigma = \{0, 1\}$.

- (i) Strings containing even number of 1's following by odd number of 0's.
- (ii) Strings that do not contain three consecutive 0's.
- (iii) Strings that contain at most three 0's.

Solution :

- (a) (i) $\text{R.E.} = a^*c^*$

- (ii) $L(r) = \{0, 2, 01, 21, 011, 211, 0111, \dots\}$

$$r = 0 + 2 + 01 + 21 + 011 + 211 + 0111 + \dots$$

$$= 0(\epsilon + 1 + 11 + 111 + \dots)$$

$$+ 2(\epsilon + 1 + 11 + \dots) = 01^* + 21^*$$

$$r = (0 + 2)1^*$$

- (iii) $L(r) = \{00, 010, 0110, 01110, \dots\}$

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

- (iv) DFA for the given language can be drawn as shown in Fig. Ex. 3.4.17(a).

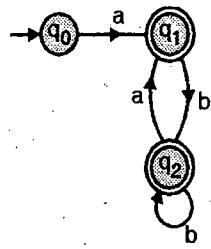


Fig. Ex. 3.4.17(a)

Moving the loop between q_1 and q_2 and q_1 we get :

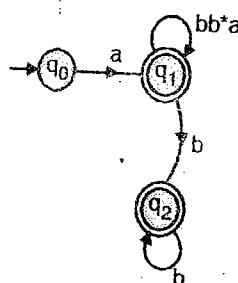


Fig. Ex. 3.4.17(b)

R.E. corresponding to $q_1 = a(bb^*a)^*$

R.E. corresponding to $q_2 = a(bb^*a)^*bb^*$

$$\therefore \text{Combined R.E.} = a(bb^*a)^* + a(bb^*a)^*bb^* \\ = a(bb^*a^*) + [\epsilon + bb^*]$$

- (b) (i) $(11)^*0(00)^*$
- (ii) $(1+01+001)^*(\epsilon+0+00)$
- (iii) $1^* + 1^*01^* + 1^*01*01^* + 1^*01*01*01^*$

Example 3.4.18.

Let $\Sigma = \{0, 1\}$, construct NFA and hence regular expression for each of the following :

- (a) $L_1 = \{\omega \in \Sigma^* \mid \omega \text{ has at least one pair of consecutive } 0\text{'s}\}$
- (b) $L_2 = \{\omega \in \Sigma^* \mid \omega \text{ has no pair of consecutive zero}\}$
- (c) $L_3 = \{\omega \in \Sigma^* \mid \omega \text{ is a string with either } 1 \text{ preceding a } 0 \text{ or } 0 \text{ preceding } 1\}$
- (d) $L_4 = \{\omega \in \Sigma^* \mid \omega \text{ consists of an even number of } 0\text{'s followed by odd number of } 1\text{'s}\}$.

Solution :

- (a) $L_1 = \{\omega \in \Sigma^* \mid \omega \text{ has at least one pair of consecutive } 0\text{'s}\}$

The equivalent FA is shown in Fig. Ex. 3.4.18(a).

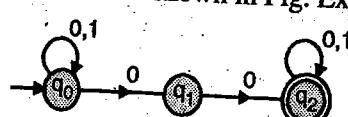


Fig. Ex. 3.4.18(a) : FA for Example 3.4.18(a)

R.E. = $(0+1)^*00(0+1)^*$

- (b) $L_2 = \{\omega \in \Sigma^* \mid \omega \text{ has no pair of consecutive zero}\}$

The equivalent FA is shown in Fig. Ex. 3.4.18(b)

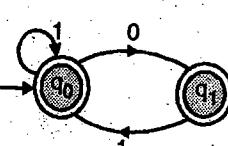
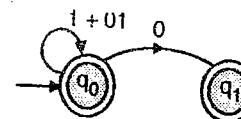


Fig. Ex. 3.4.18(b) : FA for Example 3.4.18(b)

The effect of loop between q_0 and q_1 can be moved to q_0 . A new machine is obtained and shown in Fig. Ex. 3.4.18(c).

Fig. Ex. 3.4.18(c) : FA after the effect of loop moved to q_0

R.E. for state $q_0 = (1+01)^*$

R.E. for state $q_1 = (1+01)^*0$

$$\therefore \text{The final R.E.} = (1+01)^* + (1+01)^*0 \\ = (1+01)^*[\epsilon + 0]$$

- (c) $L_3 = \{\omega \in \Sigma^* \mid \omega \text{ is a string with either } 1 \text{ preceding a } 0 \text{ or } 0 \text{ preceding } 1\}$

The equivalent FA is shown in Fig. Ex. 3.4.18(d).

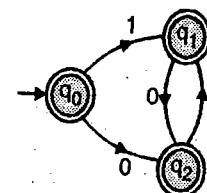


Fig. Ex. 3.4.18(d) : FA for example 3.4.18(c)

The effect of loop between q_1 and q_2 can be removed by moving its effect both on q_1 and q_2 . A new machine is shown in Fig. Ex. 3.4.18(e).

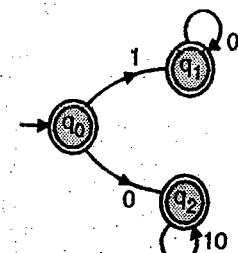


Fig. Ex. 3.4.18(e) : FA after elimination of loop

R.E. = $\epsilon + 1(01)^* + 0(10)^*$

- (d) $L_4 = \{\omega \in \Sigma^* \mid \omega \text{ consists of an even number of } 0\text{'s followed by an odd number of } 1\text{'s}\}$

The equivalent FA is shown in Fig. Ex. 3.4.18(f).

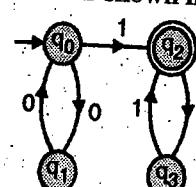


Fig. Ex. 3.4.18(f) : FA for Example 3.4.18(d)

The effect of loop between q_0 and q_1 , between q_2 and q_3 are removed and FA is redrawn as shown in Fig. Ex. 3.4.18(g).

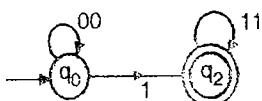


Fig. Ex. 3.4.18(g)

$$\therefore \text{R.E.} = (00)^* 1 (11)^*$$

Example 3.4.19

Construct regular expression for following transition diagram:

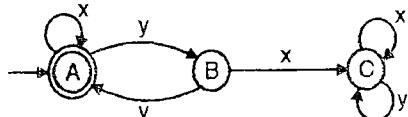
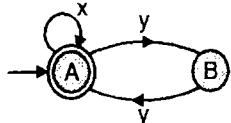


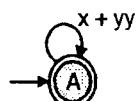
Fig. Ex. 3.4.19

Solution :

Step 1 : The state C is a dead state; It can be eliminated



Step 2 : The effect of loop between the states A and B can be transferred on the state A.



Step 3 : The regular expression for the given transition diagram $= (x + yy)^*$

Example 3.4.20

Construct regular expression for following transition diagram.

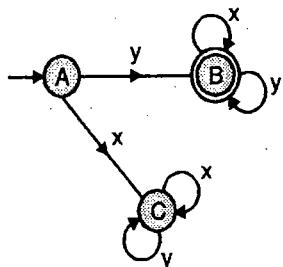
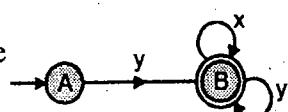


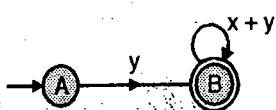
Fig. Ex. 3.4.20

Solution :

The state C is a dead state



The above machine can be equivalently redrawn as



The RE for the above FSM is given by $y(x + y)^*$

Example 3.4.21

Write regular expressions for

- Set of strings of 0's and 1's whose tenth symbol from the right end is 1.
- Set of string of 0's and 1's not containing 101 as substring.
- Set of strings with even number of a's followed by odd number of b's that is for language.
 $L = \{a^{2n} b^{2m+1} : n \geq 0, m \geq 0\}$
- Set of strings of an equal number of 0's and 1's such that in every prefix, the number of 0's differs from the number of 1's by at most 1.

Solution :

- $(0 + 1)^* 1 (0 + 1) (0 + 1) (0 + 1) (0 + 1) (0 + 1) (0 + 1) (0 + 1) (0 + 1) (0 + 1)$
- We can write the RE after giving FA for the given language.

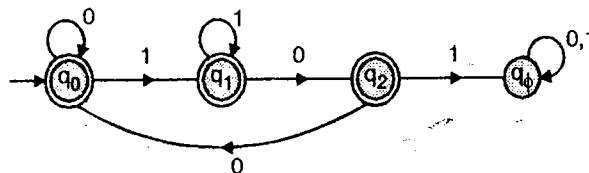
Finite automata

Fig. Ex. 3.4.21

Finite automata to RE

- The state q_3 can be discarded.
- The effect of loop among $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$ can be transferred on the state q_0 .

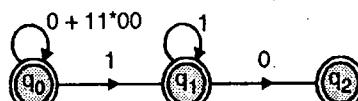


Fig. Ex. 3.4.21(a)

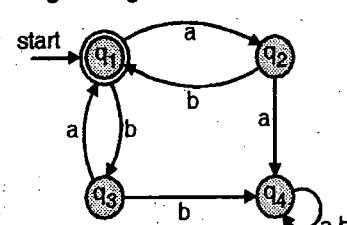
We can write the RE from the above diagram.

$$\text{RE} = (0 + 11^*00)^* [\epsilon + 11^* + 11^*0]$$

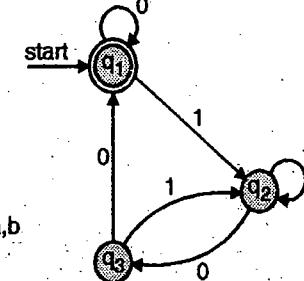
- $\text{RE} = (aa)^* (bb)^* b$
- $(01 + 10)^*$

Example 3.4.22

Construct the regular expressions for the transition diagrams given.



(i)

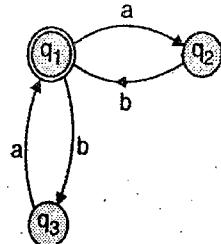


(ii)

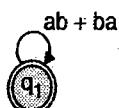
Fig. Ex. 3.4.22

Solution :

Step 1 : Removing the dead state q_4 , we get



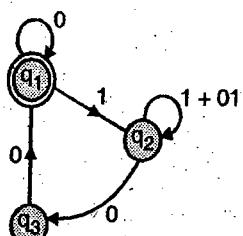
Step 2 : Removing the effect of two state q_2 and q_3 , we get



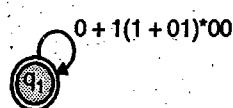
$$\therefore \text{Required RE} = (ab + ba)^*$$

(ii)

1. The effect of loop between the states q_2 and q_3 can be transferred on the state q_2 .



2. The effect of loop can be moved on the state q_1 .



$$3. \text{ The RE} = (0 + 1(1 + 01)*00)^*$$

Example 3.4.23

Construct a regular expression corresponding to the state diagram given below :

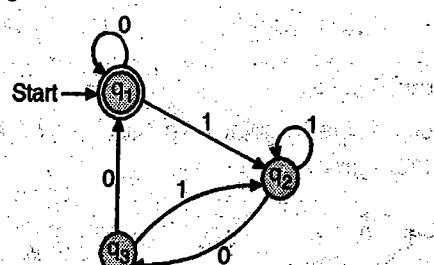


Fig. Ex. 3.4.23

Solution :

Step 1 : Removing loop between the states q_2 and q_3 , we get

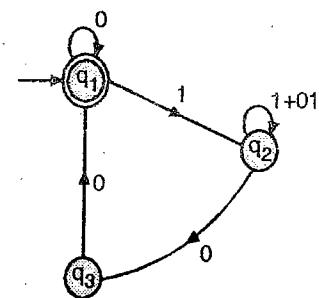
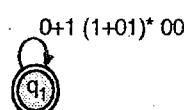


Fig. Ex. 3.4.23(a)

Step 2 : Eliminating states q_2 and q_3 , we get



$$\therefore \text{Required R.E.} = (0 + 1(1 + 01)*00)^*$$

Example 3.4.24

Prove the formula $(00^*1)^*1 = 1 + 0(0 + 10)^*11$

Solution :

The formula can be proved by drawing an equivalent DFA.

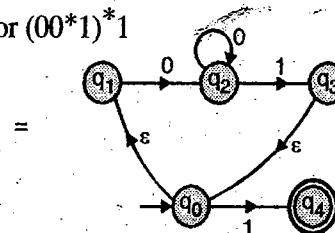
DFA for $(00^*1)^*1$ 

Fig. Ex. 3.4.24(a) : An equivalent ε-NFA

DFA corresponding to ϵ -NFA of Fig. Ex. 3.4.24(b) is given in Fig. Ex. 3.4.24(b).

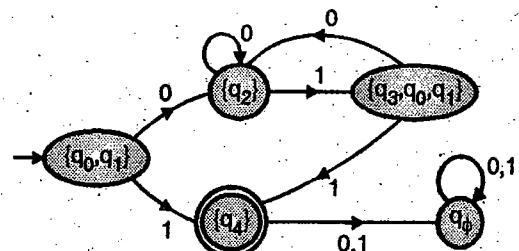
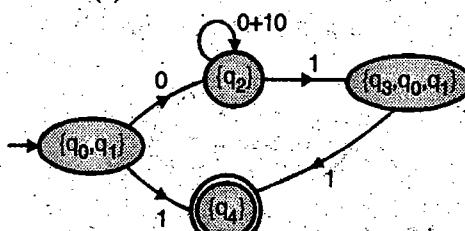


Fig. Ex. 3.4.24(b) : An equivalent DFA

FA after moving the effect of loop between $\{q_2\}$ and $\{q_3, q_0, q_1\}$ is redrawn as shown in Fig. Ex. 3.4.24(c).

Fig. Ex. 3.4.24(c) : Effect of loop is moved to $\{q_2\}$



There are two paths between the start $\{q_0, q_1\}$ and the final state $\{q_4\}$

- First path is represented by

$$R.E. = 1$$

- Second path is represented by

$$R.E. = 0(0 + 10)^*11$$

\therefore The equivalent regular expression for FA shown in Fig. Ex. 3.4.24(c) is,

$$R.E. = 0(0 + 10)^*11 + 1$$

Hence the formula is proved.

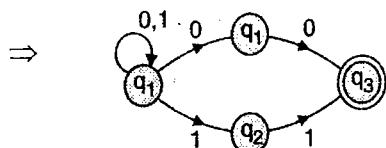
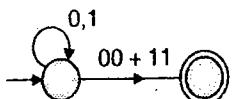
Example 3.4.25 SPPU - Dec. 13, 6 Marks

Construct DFA for Regular Expression

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

Solution :

$$(0 + 1)^* (00 + 11) \Rightarrow$$



NFA to DFA

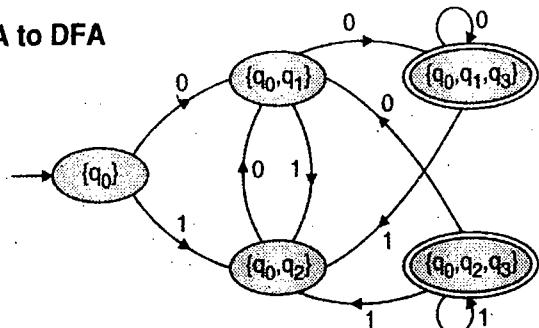


Fig. Ex. 3.4.25

Syllabus Topic : Conversion of FA to RE using Arden's Theorem

3.4.2 Arden's Theorem

Let P , Q and R be regular expressions on Σ . Then, if P does not contain ϵ , the equation

$$R = Q + RP \quad \dots(3.16)$$

has a unique solution given by,

$$R = QP^* \quad \dots(3.17)$$

Proof : Let us verify whether QP^* is a solution to the equation.

$$R = Q + RP$$

On substitution of QP^* for R in Equation (3.16),

$$\begin{aligned} R = Q + RP &= Q + QP^*P = Q^*(\epsilon + P^*P) \\ &= QP^* \end{aligned}$$

Thus the Equation (3.16) is satisfied by $R = QP^*$. We still do not know whether QP^* is a unique solution or not.

To establish uniqueness, we expand

$$\begin{aligned} R &= Q + RP \\ &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \\ &= Q + QP + (Q + RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &\vdots \\ &= Q + QP + QP^2 + \dots + QP^i + RP^{i+1} \\ &= Q(\epsilon + P + P^2 + \dots + P^i) + RP^{i+1} \dots(3.18) \end{aligned}$$

here, i is any arbitrary integer.

Let us take a string $\omega \in R$ | length of $\omega = k$.

Substituting k for i in Equation (3.18)

$$R = Q(\epsilon + P + P^2 + \dots + P^k) + RP^{k+1}$$

Since P does not contain ϵ , w is not contained in RP^{k+1} as the shortest string generated by RP^{k+1} will have a length of $k + 1$.

Since ω is contained in R , it must be contained in $Q(\epsilon + P + P^2 + \dots + P^k)$.

Conversely, if ω is contained in QP^* then for some integer k it must be in $Q\{\epsilon + P + P^2 + \dots + P^k\}$, and hence in $R = Q + RP$.

3.4.2.1 Application of Arden's Theorem

A finite automata can be represented using a system of equations :

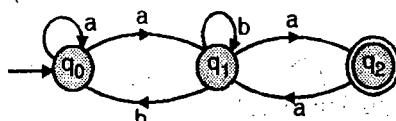


Fig. 3.4.6 : A transition diagram to be converted into a set of equations

Let us consider the FA of Fig. 3.4.6. The set of strings accepted by this FA will be described by a path.

Starting from q_0 and ending q_2 :

- A path from q_0 to q_2 will be reached through q_1 .
- These strings will end in 'a' with a prefix generated during coming to q_1 .



- If the set of strings generated by coming to q_1 from q_0 is q_1 and the set of strings generated by coming to q_2 from q_0 is q_2 .

The set q_2 can be written as

$$q_2 = q_1 a \quad \dots(3.19)$$

Similarly, the state q_1 can be reached from

- (1) q_2 on a, (2) q_1 on b, (3) q_0 on a.

Thus q_1 can be expressed in terms of q_0 and q_2 .

$$q_1 = q_0 a + q_1 b + q_2 a \quad \dots(3.20)$$

Similarly, the state q_0 can be reached from :

- (1) q_0 on a,

- (2) q_1 on b,

- (3) Without any input as q_0 is a start state.

Thus q_0 can be expressed in terms of q_0 , q_1 and ϵ .

$$q_0 = q_0 a + q_1 b + \epsilon \quad \dots(3.21)$$

- These three Equations (3.19, 3.20, 3.21) can be solved with the help of Arden's theorem.
- The set represented by q_2 will be the solution for the FA of Fig. 3.4.6. q_2 is a final state.

Solving of equations using Arden's theorem

1. Substituting for q_2 in Equation (3.20). q_2 is given by Equation (3.19)

$$\begin{aligned} q_1 &= q_0 a + q_1 b + q_1 aa \\ &= q_0 a + q_1 (b + aa) \end{aligned} \quad \dots(3.22)$$

Equation (3.21) is of the form

$$R = Q + RP \text{ with } Q = q_0 a,$$

$$R = q_1 \text{ and } P = (b + aa)$$

and its solution is given by,

$$R = QP^*$$

$$\text{Thus, } q_1 = q_0 a (b + aa)^* \quad \dots(3.23)$$

2. Substituting the value of q_1 from Equation (3.23) into Equation (3.21)

$$\begin{aligned} q_0 &= q_0 a + q_0 a (b + aa)^* b + \epsilon \\ &= \epsilon + q_0 (a + a (b + aa)^* b) \end{aligned} \quad \dots(3.24)$$

The Equation (3.24) is of the form

$$R = Q + RP \text{ from } Q = \epsilon, R = q_0$$

$$\text{and } P = (a + a (b + aa)^* b)$$

and its solution is given by,

$$R = QP^*$$

$$\text{Thus, } q_0 = \epsilon \cdot (a + a (b + aa)^* b)^*$$

$$= (a + a (b + aa)^* b)^* \quad \dots(3.25)$$

3. Substituting the value of q_0 from Equation (3.25) into the Equation (3.23).

$$q_1 = (a + a (b + aa)^*)^* a (b + aa)^* \quad \dots(3.26)$$

4. Substituting the value of q_1 from Equation (3.26), into the Equation (3.19).

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

Thus the R.E. representing the FA of Fig. 3.4.6 is given by :

$$\text{R.E.} = (a + a (b + aa)^*)^* a (b + aa)^* a$$

To use Arden's theorem, the FA should not contain ϵ -moves.

Example 3.4.26

Prove that the following F.A. accepts strings with equal number of 0's and 1's; such that each prefix has at most one more 0 than 1's or at most one more 1 than 0's.

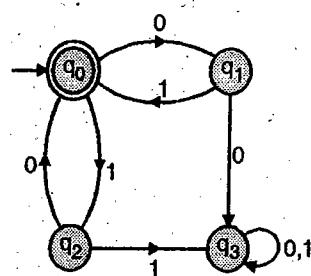


Fig. Ex. 3.4.26

Solution :

The state q_3 is a dead state. It is not required to equate for state q_3 .

Set of equations for regular sets represented by the states q_0 , q_1 and q_2 are given below.

$$q_0 = \epsilon + q_1 \cdot 1 + q_2 \cdot 0 \quad \dots(1)$$

$$q_1 = q_0 \cdot 0 \quad \dots(2)$$

$$q_2 = q_0 \cdot 1 \quad \dots(3)$$

Substituting values of q_1 and q_2 from Equation (2) and (3) in Equation (1)

$$\begin{aligned} q_0 &= \epsilon + q_0 \cdot 1 + q_0 \cdot 0 \\ &= \epsilon + q_0 (01 + 10) \end{aligned} \quad \dots(4)$$

The Equation (4) is of the form $R = Q + RP$ with $Q = \epsilon$, $R = q_0$ and $P = (01 + 10)$ and its solution is given by $R = QP^*$

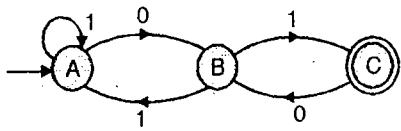
$$\therefore q_0 = \epsilon \cdot (01 + 10)^* = (01 + 10)^*$$

Observations

1. The regular set accepted by the given FA will have equal number of 0's and 1's as the R.E. $= (01 + 10)^*$ implies 01 or 10 coming zero or more times.
2. The prefix of regular set accepted by the given FA is
 - (a) The set represented by q_1 which is $(01 + 10)^*0$. Number of 0's is one more than 1's.
 - (b) The set represented by q_2 which is $(01 + 10)^*1$. Number of 1's is one more than 0's.

Example 3.4.27

Consider the following transition diagram, convert it to an equivalent regular expression using Arden's theorem.


Fig. Ex. 3.4.27

Solution : The set of equations for regular sets represented by the states A, B and C are given below :

$$A = \epsilon + A1 + B1 \quad \dots (1)$$

$$B = A0 + C0 \quad \dots (2)$$

$$C = B1 \quad \dots (3)$$

Rewriting Equation (1),

$A = (\epsilon + B1) + A1$, it is of the form $R = Q + RP$ with $R = A$; $Q = (\epsilon + B1)$ and $P = 1$ and its solution is given by $R = QP^*$

$$\therefore A = (\epsilon + B1)1^* \quad \dots (4)$$

Substituting, the value of A from Equation(4) and the value of C from Equation(3) in Equation(2).

$$\begin{aligned} B &= (\epsilon + B1)1^*0 + B10 \\ &= 1^*0 + B11^*0 + B10 \\ &= 1^*0 + B(11^*0 + 10) \end{aligned}$$

From Arden's theorem,

$$B = 1^*0(11^*0 + 10)^* \quad \dots (5)$$

Substituting the value of B from Equation(5) in Equation(3).

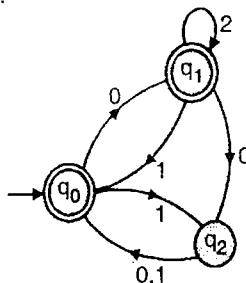
$$C = 1^*0(11^*0 + 10)^*1$$

Since state C is a final state, the regular expression represented by C describes the FA given in the Example 3.4.27.

$$\therefore \text{R.E.} = 1^*0(11^*0 + 10)^*1$$

Example 3.4.28

Find the regular expression for the set strings recognized by the FA shown in Fig. Ex. 3.4.28. Use Arden's theorem.


Fig. Ex. 3.4.28

Solution : The set of equations for regular sets represented by the state q_0 , q_1 and q_2 are given below.

$$q_0 = \epsilon + q_1 1 + q_2 (0 + 1) \quad \dots (1)$$

$$q_1 = q_1 2 + q_0 0 \quad \dots (2)$$

$$q_2 = q_1 0 + q_0 1 \quad \dots (3)$$

Substituting q_2 from Equation (3) in Equation (1)

$$\begin{aligned} q_0 &= \epsilon + q_1 1 + (q_1 0 + q_0 1)(0 + 1) \\ &= \epsilon + q_1 1 + q_1 00 + q_1 01 + q_0 1(0 + 1) \\ &= \epsilon + q_1 (1 + 00 + 01) + q_0 1 (0 + 1) \end{aligned}$$

From Arden's theorem,

$$q_0 = [\epsilon + q_1 (1 + 00 + 01)] (1(0 + 1))^* \quad \dots (4)$$

Putting the value of q_0 from Equation (4) in Equation(2),

$$\begin{aligned} q_1 &= q_1 [2 + (\epsilon + q_1 (1 + 00 + 01))] \\ &\quad (1(0 + 1))^* 0 \\ &= (1(0 + 1))^* 0 + q_1 [2 + (1 + 00 + 01)] \\ &\quad (1(0 + 1))^* 0 \end{aligned}$$

From Arden's theorem,

$$q_1 = (1 + (0 + 1))^* 0 [2 + (1 + 00 + 01)] \\ (1(0 + 1))^* 0]^* \quad \dots (5)$$

Putting the value of q_1 from Equation (5) in Equation(4)

$$\begin{aligned} q_0 &= [\epsilon + (1(0 + 1))^* 0 [2 + (1 + 00 + 01)] \\ &\quad (1(0 + 1))^* 0]^* (1 + 00 + 01)] \\ &\quad (1(0 + 1))^* \end{aligned} \quad \dots (6)$$



The R.E. describing the machine is given by,

$$\begin{aligned} \text{R.E.} &= q_0 + q_1 \quad [\text{both } q_0 \text{ and } q_1 \text{ are final states}] \\ &= [\epsilon + (1(0+1))^*0[2 + (1+00+01) \\ &\quad (1(0+1))^*0]^* \\ &\quad (1+00+01)](1(0+1))^* \\ &\quad +(1(0+1))^* \\ &0[2 + (1+00+01)(1(0+1))^*0] \end{aligned}$$

Example 3.4.29

Obtain regular expression for the following FA.

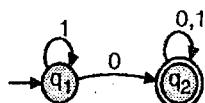


Fig. Ex. 3.4.29

Solution :

The set of equations for regular sets represented by the states q_1 and q_2 are given below.

$$q_1 = \epsilon + q_1 \cdot 1 \quad \dots(1)$$

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

Applying Arden's theorem on Equation (1), we get,

$$q_1 = 1^* \quad \dots(3)$$

Putting the value of q_1 from (3) in Equation (1), we get

$$q_2 = 1^* \cdot 0 + q_2(0+1)$$

$$\therefore q_2 = 1^* \cdot 0 \cdot (0+1)^*$$

∴ The regular expression for the given F.A. = $1^* \cdot 0 \cdot (0+1)^*$

Example 3.4.30 [SPPU - May 15, 4 Marks]

Find RE for the following DFA using Arden's theorem.

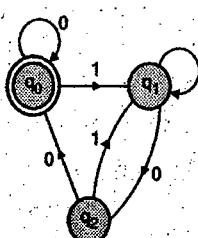


Fig. Ex. 3.4.30

Solution :

The set of equations for regular sets represented by the states q_0 , q_1 and q_2 are given below.

$$q_0 = \epsilon + q_0 \cdot 0 + q_2 \cdot 0 \quad \dots(1)$$

$$q_1 = q_0 \cdot 1 + q_1 \cdot 1 + q_2 \cdot 1 \quad \dots(2)$$

$$q_2 = q_1 \cdot 0 \quad \dots(3)$$

Substituting the value of q_2 from Equation (3) in Equation (1) and (2), we get

$$q_0 = \epsilon + q_0 \cdot 0 + q_1 \cdot 0 \cdot 0 \quad \dots(4)$$

$$\begin{aligned} q_1 &= q_0 \cdot 1 + q_1 \cdot 1 + q_1 \cdot 0 \cdot 1 \\ &= q_0 \cdot 1 + q_1 \cdot (1+0) \end{aligned} \quad \dots(5)$$

Applying Arden's theorem on Equation (5), we get:

$$q_1 = q_0 \cdot 1 \cdot (1+0)^* \quad \dots(6)$$

Substituting the value of q_1 in Equation (4) we get

$$\begin{aligned} q_0 &= \epsilon + q_0 \cdot 0 + q_0 \cdot 1 \cdot (1+0)^* \cdot 0 \cdot 0 \\ &= \epsilon + q_0 [0+1(1+0)^*00] \end{aligned}$$

Applying Arden's theorem, we get

$$\begin{aligned} q_0 &= \epsilon \cdot (0+1(1+0)^*00)^* \\ &= (0+1(1+0)^*00)^* \end{aligned}$$

$$\therefore \text{Required R.E.} = (0+1(1+0)^*00)^*$$

3.5 FA Limitations

1. FA cannot handle the following types of languages.
 - a) Context Free Language
 - b) Context Sensitive Language
 - c) Recursively enumerable language
2. FA cannot be used for computation.
3. FA is not allowed to modify its own input. A turing machine can modify its own input.
4. FA does not have memory to store variables. Therefore, it cannot be used for solving of general problems.
5. FA can be used for a subset of language, which is regular.

Syllabus Topic : Pumping Lemma for RL

3.6 Pumping Lemma for Regular Language

Some languages are regular. There are other languages which are not regular. One can neither express a non-regular language using regular expression nor design finite automata for it.

- Pumping lemma gives a necessary condition for an input string to belong to a regular set.



- Pumping lemma does not give sufficient condition for a language to be regular.
- Pumping lemma should not be used to establish that a given language is regular.
- **Pumping lemma should be used to establish that a given language is not regular.**
- The pumping lemma uses the pigeonhole principle which states that if n pigeons are placed into less than n holes, some holes have to have more than one pigeon in it. Similarly, a string of length $\geq n$ when recognized by a FA with n states will see some states repeating.

3.6.1 Definition of Pumping Lemma

Let L be a regular language and

- $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata with n -states. Language L is accepted by m . Let $\omega \in L$ and $|\omega| \geq n$, then ω can be written as xyz , where
- $|y| > 0$
 - $|xy| \leq n$
 - $xy^i z \in L$ for all $i \geq 0$ here y^i denotes that y is repeated or pumped i times.

3.6.2 Interpretation of Pumping Lemma

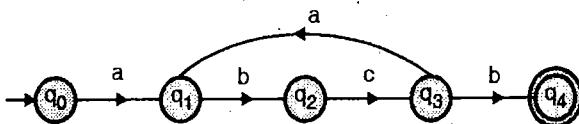


Fig. 3.6.1 : FA considered for interpretation of pumping lemma

Let us consider the FA of Fig. 3.6.1

No. of states = 5 (q_0 to q_4)

Let us take a string ω with $|\omega| \geq 5$, recognized by the FA.

$$\omega = abcabcb$$

To recognize the string $\omega = abcabcb$, the machine will transit through various states as shown in Fig. 3.6.2.

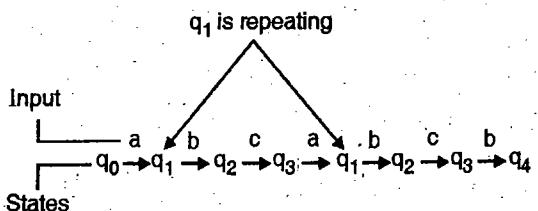


Fig. 3.6.2 : Transitions of FA on input abcabcb

As the input abcabcb takes the machine through the loop $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_1$, this loop can repeat any

number of times. In terms of abcabcb, we can say that if abcabcb is accepted by FA then every string in $a(bca)^*bcb$ will be accepted by the FA of Fig. 3.6.1. The portion bca is input during the loop.

$$q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$$

Thus, if abcabcb is accepted by the FA then abcabcb can be written as xyz, with

$$x = a$$

$$y = bca$$

$$z = bcb$$

- Length of abcabcb is $\geq n$
- $xy^i z$ for every $i \geq 0$ or $a(bca)^i bcb$ for every $i \geq 0$ will be accepted by the FA of Fig. 3.6.1.

3.6.3 Proof of Pumping Lemma

1. Suppose L is a regular language.
2. Suppose M be a DFA with n states, such that DFA accepts the given regular language L .
 - i.e. $L = L(M)$, Language of M is same as the given regular language..
3. Let us consider a string $\omega \in L$ | $|\omega| \geq n$.
- String ω can be written as $a_1 a_2 a_3 \dots a_m$ with $m \geq n$.
4. Let us assume that states of M are given by $q_0, q_1, q_2, \dots, q_{n-1}$ with q_0 as a starting state and q_{n-1} as a final state.
5. Let us assume that after feeding the first i characters of the word $\omega = a_1 a_2 \dots a_m$, machine will be in a state r_i .

$$\delta^*(q_0, a_1 a_2, \dots a_i) = r_i$$
6. As the word ω is fed through the machine M , the machine will go through the various states as :

| | | | | | |
|-------|-------|---------|-----------|-----------|-----------------------------------|
| q_0 | r_1 | r_2 | \dots | r_{m-1} | $q_{n-1} \leftarrow \text{state}$ |
| a_1 | a_2 | \dots | a_{m-1} | a_m | $\leftarrow \text{input}$ |
- Since the length of ω , $|\omega| \geq n$ it is not possible for the machine to move through distinct states.
7. Let us assume that r_i and r_j are same. There is a loop from r_i to r_j .
8. The string ω can be divided into three parts.
 1. $x = \text{Portion before loop} = a_1 a_2 \dots a_i$
 2. $y = \text{Portion of loop} = a_{i+1} a_{i+2} \dots a_j$
 3. $z = \text{Portion of loop} = a_{j+1} a_{j+2} \dots a_m$



9. Since y is a portion relating to loop, it can repeat any number of times. The same is shown in Fig. 3.6.3.

$$x = a_1 a_2 \dots a_i$$

$$y = a_{i+1} \dots a_j$$

$$z = a_{j+1} \dots a_m$$

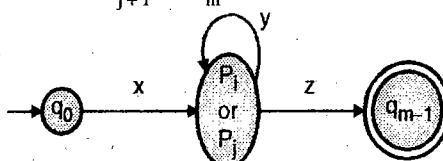


Fig. 3.6.3 : Behaviours of machine M on input $xy^i z$ for $i \geq 0$

10. From the Fig. 3.6.3 it is clear that if $xyz \in L$ then $xy^i z$ will also be accepted by the machine for every $i \geq 0$

3.6.4 Application of Pumping Lemma

- Pumping lemma should be used to prove that given set is not regular.
- Pumping lemma should be applied to a problem in steps as given below.

Step 1 : To prove that L is not regular, we proceed with assumption that L is regular and it is accepted by a FA with n states.

Step 2 : Now, we choose a string $\omega \in L$ such that $|\omega| \geq n$.

where ω to select is very important and it is the key to solution using pumping lemma.

Step 3 : ω is written as xyz ,

With $|xy| \leq n$

and $|y| > 0$

Now we must find an i such that $xy^i z \notin L$.

This will contradict the assumption that L is regular and hence we can conclude that L is not regular.

where i to select such that $xy^i z \notin L$ is very important.

Example 3.6.1

Show that the language $L = \{a^n b^n\}$ is not regular.

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string

$$\omega = a^n b^n$$

$$|\omega| = 2n \geq n$$

Let us write w as xyz , with

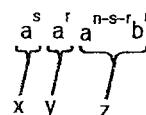
$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since, $|xy| \leq n$, y must be of the form a^r $r > 0$

Since, $|xy| \leq n$, x must be the form a^s .

Now, $a^n b^n$ can be written as



Step 3 : Let us check whether $xy^i z$ for $i = 2$ belongs to L .

$$\begin{aligned} xy^2 z &= a^s (a^r)^2 a^{n-s-r} b^n = a^s a^{2r} a^{n-s-r} b^n \\ &= a^{s+2r+n-s-r} b^n = a^{n+r} b^n \end{aligned}$$

Since $r > 0$, number of a's in $a^{n+r} b^n$ is greater than number of b's. Therefore, $xy^2 z \notin L$. Hence by contradiction we can say that the given language is not regular.

Example 3.6.2

Using pumping lemma for regular sets. Prove that the language $L = \{a^m b^n \mid m > n\}$ is not regular.

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string $w = a^p b^q$ such that

$$p + q > n \text{ and } p = q + 1$$

Let us write w as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

y could take any of the given forms :

1. $a^i \dots$ case I
2. $a^i b^j \dots$ case II
3. $b^j \dots$ case III

Step 3 : We want to find i so that $xy^i z \notin L$ and we must consider all the three cases.

Case I : We can take $i = 0$ with $xyz = xz = a^{p-i} b^q$

Since $p = q + 1$ and $i > 0$ therefore $p - i$ is not greater than q .

$$xy^0 z \notin L$$

Case II : $xy^2 z$ for $i = 2$ is given by

$$a^{p-i} (a^i b^j)^2 b^{q-j}$$

$$= a^{p-i} a^i b^j a^i b^j b^{q-j} = a^p b^j a^i b^q \notin L$$

**Case III**

$$xy^nz = a^p(b^j)^n b^{q-j} = a^p b^{q-j+nj} = a^p b^{q+(n-1)j}$$

Since $p = q + 1$, $n > 1$ and $j \geq 1$, it is clear that p is not greater than $q + (n-1)j$ and hence $xy^nz \notin L$.

Thus in all the three cases we get a contradiction. Therefore, L is not regular.

Example 3.6.3 SPPU - Aug. 15 (In Sem), 6 Marks

Using pumping lemma for regular sets. Prove that the language $L = \{0^i 1^{2i} \mid i > 0\}$ is not regular.

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string $\omega = a^n b^{2n}$

$$|\omega| = 3^n \geq n$$

Let us write w as xyz , with

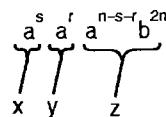
$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since $|xy| \leq n$, y must be of the form $a^r \mid r > 0$.

Since $|xy| \leq n$, x must be of the form a^s .

Now, $\omega = a^n b^{2n}$ can be written as



Step 3 : Let us check whether xy^iz for $i = 2$ belongs to L .

$$\begin{aligned} xy^2z &= a^s(a^r)^2 a^{n-s-r} b^{2n} \\ &= a^s a^{2r} a^{n-s-r} b^{2n} \\ &= a^{s+2r+n-s-r} b^{2n} \\ &= a^{n+r} b^{2n}. \end{aligned}$$

Since $r > 0$, $a^{n+r} b^{2n}$ is not of the form $a^i b^{2i}$. Therefore, $xy^2z \notin L$. Hence by contradiction, we can say that the given language is not regular.

Example 3.6.4

Using pumping lemma for regular sets prove that the language.

$L = \{0^m 1^n 0^{m+n} \mid m \geq 1 \text{ and } n \geq 1\}$ is not regular.

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string.

$$\omega = 0^n 1^m 0^{m+n}$$

$$|\omega| = 2(m+n) \geq n$$

Let us write ω as xyz with

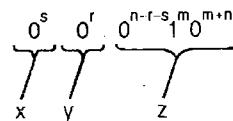
$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since $|xy| \leq n$, y must be of the form $0^r \mid r > 0$

Since $|xy| \leq n$, x must be of the form 0^s .

Now, $\omega = 0^n 1^m 0^{m+n}$ can be written as



Step 3 : Let us check whether xy^iz for $i = 2$ belongs to L .

$$\begin{aligned} xy^2z &= 0^s 0^{2r} 0^{n-r-s} 1^m 0^{m+n} \\ &= 0^{n+r} 1^m 0^{m+n} \end{aligned}$$

Since $r > 0$, $0^{n+r} 1^m 0^{m+n}$ is not of the form

$$0^i 1^j 0^{i+j}$$

Therefore, $xy^2z \notin L$. Hence by contradiction, we can say that given language is not regular.

Example 3.6.5

Using pumping lemma for regular sets, prove that the language $L = \{\omega\omega^R \mid \omega \in \{0, 1\}^*\}$ is not regular.

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string

$$\omega = \underbrace{a^n b}_\omega \underbrace{ba^n}_\omega \leftarrow \text{from } \omega^R$$

$$|\omega| = 2n + 2 \geq n$$

Let us write w as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since $|xy| \leq n$, x must be of the form a^s .

Since $|xy| \leq n$, y must be of the form $a^r \mid r > 0$.

Now,

$$\omega = a^n bba^n = \underbrace{a^s}_x \underbrace{a^r}_y \underbrace{a^{n-s-r} bba^n}_z$$

Step 3 : Let us check whether xy^iz for $i = 2$ belongs to L .

$$xy^2z = a^s a^{2r} a^{n-s-r} bba^n = a^{n+r} bba^n$$

Since $r > 0$, $a^{n+r} bba^n$ is not of the form $\omega\omega^R$ as the strings starts with $(n+r)$ a's but ends



in (n) a's. Therefore, $xy^2z \notin L$. Hence by contradiction, we can say that the given language is not regular.

Example 3.6.6

Using pumping lemma for regular sets. Prove that the language $L = \{ \omega\omega \mid \omega \in \{0, 1\}^* \}$ is not regular.

Solution :

Step 1 : Let us assume that the given language is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string

$$\omega = \underbrace{a^n b}_w \underbrace{a^n b}_w \leftarrow \text{from } \omega\omega$$

$$|\omega| = 2n + 2 \geq n$$

Let us write w as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since $|xy| \leq n$, x must be of the form a^s .

Since $|xy| \leq n$, y must be of the form $a^r \mid r > 0$.

Now,

$$\omega = a^n b a^n b = \underbrace{a^s}_x \underbrace{a^r}_y \underbrace{a^{n-s-r}}_z \underbrace{ba^n b}$$

Step 3 : Let us check whether $xy^i z$ for $i = 2$ belongs to L.

$$\begin{aligned} xy^2z &= a^s a^{n-s-r} ba^n b \\ &= a^{n+r} ba^n b \end{aligned}$$

Since $r > 0$, $a^{n+r} ba^n b$ is not of the form $\omega\omega^R$ as the number of a's in the first half is $n+r$ and in the second half is n.

Therefore, $xy^2z \notin L$. Hence by contradiction, the given language is not regular.

Example 3.6.7

Use pumping Lemma to show whether or not the following language is regular : $L = \{\omega\omega\omega \mid \omega \in \{a,b\}^*\}$

Solution :

Step 1 : Let us assume that L is regular and L is accepted by a Finite Automata with n states.

Step 2 : Let us chose a string

$$\omega = a^n b a^n b$$

$$|\omega| = 3n + 3 \geq n$$

Let us write ω as xyz, with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Since, $|xy| \leq n$, y must be of the form $a^r \mid r > 0$

Since, $|xy| \leq n$, x must be of the form a^s .

Now, $a^n b a^n b$ can be written as

$$\underbrace{a^s}_x \underbrace{a^r}_y \underbrace{a^{n-s-r}}_{\text{ba}^n b} \underbrace{ba^n b}_z$$

Step 3 : Let us check whether $xy^i z$ for $i = 0$ belongs to L.

$$xy^0 z = a^s a^{n-s-r} ba^n b = a^{n-r} ba^n b$$

Since $r > 0$, $a^{n-r} ba^n b$ is not of the form $\omega\omega\omega$. Hence by contradiction, we can say that the given language is not regular.

Example 3.6.8 SPPU - Dec. 16, 4 Marks

Using pumping lemma for regular sets, prove that the language, $L = \{0^n \mid n \text{ is a prime}\}$ is not regular.

Solution :

Step 1 : Let us assume that the given language is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string $\omega = a^p$, where p is a prime and $p > n$.

$$|\omega| = |a^p| = p \geq n$$

Let us write w as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

We can assume that $y = a^m$ for $m > 0$.

Step 3 : Length of $xy^i z$ can be written as given below:

$$\begin{aligned} |xy^i z| &= |xyz| + |y^{i-1}| = p + (i-1)m \\ \text{as } |y| &= |a^m| = m \end{aligned}$$

Let us check whether $p + (i-1)m$ is a prime for every i.

For $i = p+1$, $p + (i-1)m = p + P_m = P(1+m)$.

$P(1+m)$ is not a prime as it has two factors p and $(1+m)$ and.

$$|p| > 1$$

$$|1+m| > 1$$

So $xy^{p+1} z \notin L$. Hence by contradiction the given language is not regular.

**Example 3.6.9 SPPU - Dec 14, 6 Marks**

Using pumping lemma for regular sets, prove that the language, $L = \{a^i \mid i \geq 1\}$ is not regular.

Solution :

Step 1 : Let us assume that the given language L is regular and L is accepted by a FA with n states.

Step 2 : Let us choose a string $\omega = a^{n^2}$.

$$|\omega| = |a^{n^2}| = n^2 \geq n$$

Let us write ω as xyz with

$$|y| > 0$$

$$\text{and } |xy| \leq n$$

Step 3 : We will try to prove that xy^2z is not of the form a^{i^2} by showing that $|xy^2z|$ lies between the square of two consecutive natural numbers.

$$\text{i.e. } n^2 < |xy^2z| < (n+1)^2$$

A number lying between the square of two consecutive numbers can never be of the form i^2 .

Let us find the length xy^2z

$$\begin{aligned} |xy^2z| &= |xyz| + |y| \\ &= n^2 + (> 0) \text{ as the length of } y > 0 \text{ and length of } xyz \text{ is } n^2. \\ \therefore |xy^2z| &> n^2 \end{aligned} \quad \dots(1)$$

$$\text{Again, } |xy^2z| = |xyz| + |y|$$

Since the length of $|y| \leq n$ as $|xy| \leq n$ we can say that

$$\begin{aligned} |xy^2z| &\leq n^2 + n \\ \text{or, } |xy^2z| &< n^2 + n + n + 1 [n+1 \text{ is added on the right of the inequality}] \\ \text{or, } |xy^2z| &< (n+1)^2 \end{aligned} \quad \dots(2)$$

From the two inequalities (1) and (2)

$$n^2 < |xy^2z| < (n+1)^2.$$

Thus $xy^2z \notin L$. Hence by contradiction, the given language is not regular.

Example 3.6.10 : Is $L = \{a^n \mid n \geq 1\}$ regular ?

Solution :

The given language is regular as we can draw an equivalent FA.

[From Kleene's theorem]

The FA is given in Fig. Ex. 3.6.10.

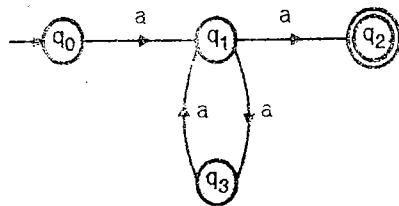


Fig. Ex. 3.6.10 : FA of Example 3.6.10

Example 3.6.11

Is the following set regular ? (Using pumping lemma)
Prove the same : $a^m b^{2m} \mid m, n > 0$

Solution :

$a^m b^{2m}$ with $m, n > 0$ is regular. The corresponding FA is shown in Fig. Ex. 3.6.11 :

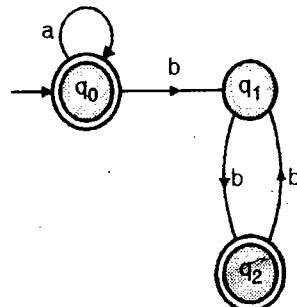


Fig. Ex. 3.6.11 : FA for $a^m b^{2m}$

Example 3.6.12 SPPU - May 15, 4 Marks

Justify why palindrome strings cannot be checked for by FSM.

Solution :

An FSM cannot check whether a string is a palindrome. A finite state machine has no way of fixing the centre of a string without seeing the entire string. Since, there is no control on the length of the string to be checked, there is no way of designing a finite state machine for palindromes.

Syllabus Topic : Closure Properties of RLs**3.7 Closure Properties of Regular Language**

If an operation on regular languages generates a regular language then we say that the class of regular languages is closed under the above operation. Some of the important closure properties for regular languages are given below.

- | | |
|---------------------------|-----------------|
| 1. Union | 2. Difference |
| 3. Concatenation | 4. Intersection |
| 5. Complementation | 6. Kleene star |
| 7. Transpose or reversal. | |

3.7.1 Regular Language is Closed under Union

Let $M_1 = (S, \Sigma, \delta_1, s_0, F)$ and
 $M_2 = (Q, \Sigma, \delta_2, q_0, G)$ be two given automata.

To prove the closure property; we must show that there is another machine M_3 which accepts every string accepted by either M_1 or M_2 and no other string. The construction M_3 is quite simple as shown in Fig. 3.7.1.

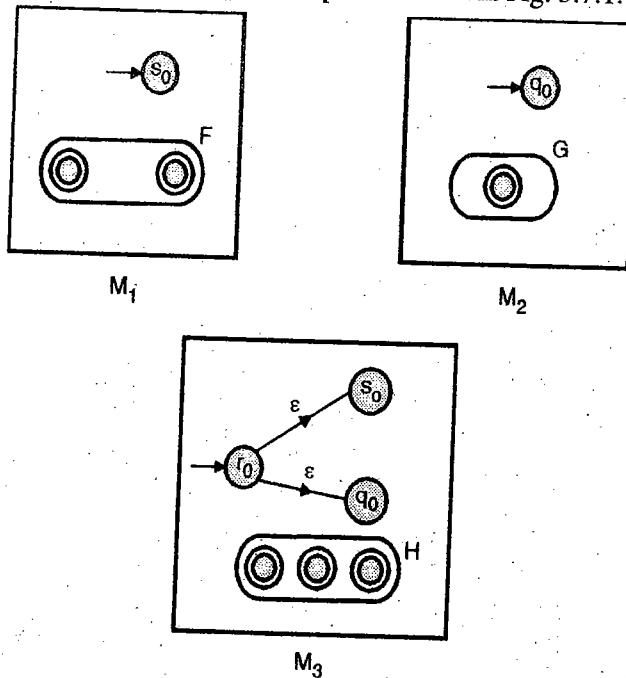


Fig. 3.7.1 : M_3 is constructed such the $L(M_3) = L(M_1) \cup L(M_2)$

Machine M_3 is constructed to accept $L(M_1) \cup L(M_2)$.

$M_3 = (R, \Sigma, \delta_3, r_0, H)$ where r_0 is a new start state. Two ϵ -moves, one from r_0 to s_0 and another from r_0 to q_0 are added.

$$R = S \cup Q \cup \{r_0\}$$

$$H = F \cup G$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(r_0, \epsilon, s_0), (r_0, \epsilon, q_0)\}$$

Machine M_3 can non-deterministically choose either M_1 or M_2 . Therefore,

$$L(M_3) = L(M_1) \cup L(M_2)$$

3.7.2 Regular Language is Closed under Concatenation

Let $M_1 = (S, \Sigma, \delta_1, s_0, F)$
and $M_2 = (Q, \Sigma, \delta_2, q_0, G)$ be two given automata.

To prove that closure properly under concatenation, we must show that there is another machine M_3 such that $L(M_3) = L(M_1) \cdot L(M_2)$. The construction of M_3 is shown in Fig. 3.7.2.

M_3 is constructed by adding ϵ -move from every final state of M_1 to start state of M_2 .

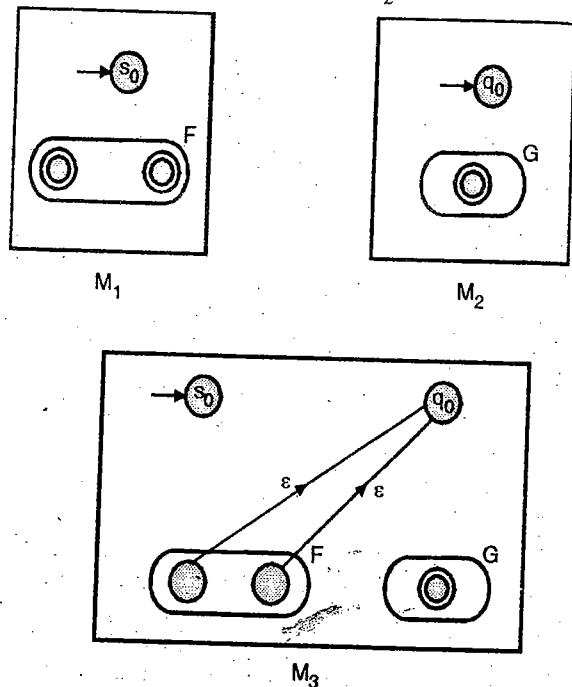


Fig. 3.7.2 : M_3 is constructed such that $L(M_3) = L(M_1) \cdot L(M_2)$

Machine M_3 is given by :

$$M_3 = (R, \Sigma, \delta_3, s_0, G) \text{ where}$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{\epsilon\text{-move from every final state of } M_1 \text{ to start state of } M_2\}$$

Machine M_3 recognizes $L(M_1) \cdot L(M_2)$ by going non-deterministically from the final state of M_1 to start state of M_2 .

3.7.3 Regular Language is Closed under Kleene Star

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be the given automata. We can construct a non-deterministic finite automata M_2 such that $L(M_2) = L(M_1)^*$. The construction of M_2 from M_1 is shown in Fig. 3.7.3.

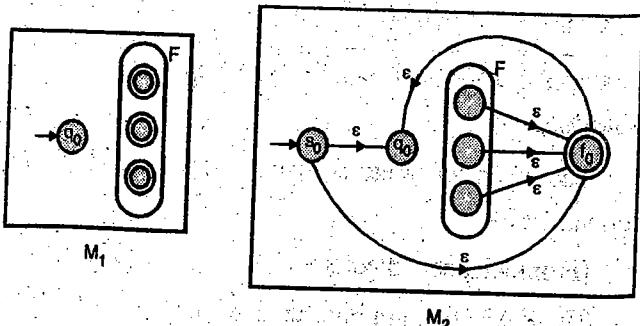


Fig. 3.7.3 : M_2 is constructed such that $L(M_2) = L(M_1)^*$

M_2 is constructed as given below :

- A new start state s_0 is added with an ϵ -move from s_0 to q_0 .
- A new final state f_0 is added with ϵ -moves from every state of F to f_0 . An ϵ -move is added from s_0 to f_0 as ϵ is a member of $L(M_1)^*$.

$$\text{Machine } M_2 = (Q \cup \{s_0, f_0\}, \Sigma, \delta, s_0, \{f_0\})$$

Machine can accept a string $\in L(M_1)$ and resume back from the start state q_0 through the ϵ -move from f_0 to q_0 . Thus accepting $L(M_1)^*$.

3.7.4 Regular Language is Closed under Complementation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the given automata. To prove the closure properly under complementation, we must show that there is another machine \bar{M} which accepts $L(\bar{M})$ where

$$\overline{L(M)} = L(\bar{M}) = \Sigma^* - L(M)$$

Given Machine after
machine complementation

If M is a deterministic finite automata then \bar{M} can be constructed by interchanging final and non final states of M .

$$\therefore \bar{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

3.7.5 Regular Language is Closed under Intersection

If L_1 and L_2 are two regular languages, then

$$\begin{aligned} L_1 \cap L_2 &= ((L_1 \cap L_2)')' = (\bar{L}_1 \cup \bar{L}_2)' \\ &= \Sigma^* - [(\Sigma^* - L_1) \cup (\Sigma^* - L_2)] \end{aligned}$$

Closeness under intersection follows directly from closeness under union and complementation.

3.7.6 Regular Languages are Closed under Difference

Let L_1 and L_2 are two regular languages. The difference $L_1 - L_2$ is the set of strings that are in language L_1 but not in L_2 . Construction of a composite automata for $L(M_1) - L(M_2)$ is already explained in Chapter 2. Thus regular languages are closed under difference.

3.7.7 Regular Languages are Closed under Reversal

Reversal of a language L is obtained by reversing every string in L . Reversal of a language L is represented by L^R .

For example,

$$\begin{aligned} \text{if } L &= \{aab, abb, aaa\}, \text{ then} \\ L^R &= \{baa, bba, aaa\} \end{aligned}$$

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be the given automata. To prove the closure property under reversal, we must show that there is another machine M_2 which accepts $L(M_1)^R$.

$$\text{or } L(M_2) = L(M_1)^R$$

M_2 can be constructed from M_1 by :

1. By reversing every transition in M_1 .
2. Start state of M_1 is made the only final state.
3. A new start state s_0 is added with ϵ -move to every final state of M_1 .

Example 3.7.1

Construct a FA accepting every string over alphabet {0, 1} ending in 01 or 10. Find the transpose of the machine or create a reverse machine.

Solution :

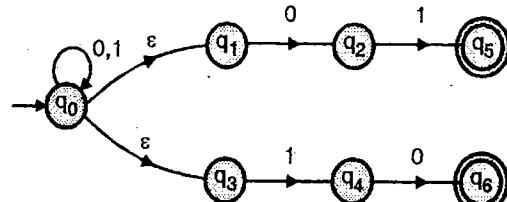


Fig. Ex. 3.7.1(a) : An NFA accepting strings ending in 01 or 10

If L is a language representing strings ending in 01 or 10. A machine for L^R is shown in Fig. Ex. 3.7.1(b).

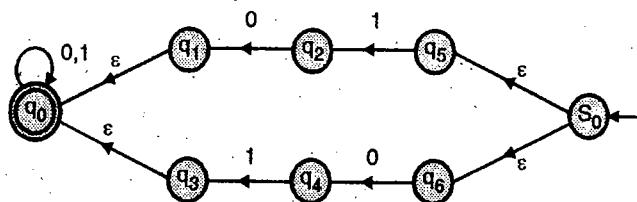


Fig. Ex. 3.7.1(b) : An NFA accepting L^R

Example 3.7.2

Explain your answer in each of the following :

- (i) Every subset of a regular language is regular.
- (ii) Every regular language has a regular proper subset.

**Solution :**

- (i) Every subset of a regular language is regular.

A subset of a regular language need not be regular. Let us consider a language

$$L_1 = \{a^n b^n \mid n \geq 1\}$$

We know from the application of pumping lemma that L_1 is not regular. Language L_1 is subset of language described by $(a+b)^*$.

Thus a subset of a regular language need not be regular.

- (ii) Every regular language has a regular proper subset.

- (1) If the language is an empty language then it will have no proper subset. Hence the above argument is not true for an empty language.

If $L = \emptyset$ then it has no proper subset

- (2) If L is $\{\epsilon\}$ then \emptyset is its proper subset. A language $L = \emptyset$ is a regular language.

- (3) For any other language, let us consider a word $\omega_1 \in L$ where ω_1 is given by $a_1 a_2 \dots a_n$ we can always construct a finite automata recognizing a language L_1 where $L_1 = \{\omega_1\}$.

Thus the language has a proper subset.

Example 3.7.3

For each statement below, decide whether it is true or false, if it is true, prove it, all parts refer to language over the alphabet $\{a, b\}$

- (a) If $L_1 \subseteq L_2$ and L_1 is not regular, then L_2 is not regular.
 (b) If $L_1 \subseteq L_2$ and L_2 is not regular, then L_1 is not regular.
 (c) If L_1 and L_2 are non regular then $L_1 \cup L_2$ is non regular.

Solution : Let us assume that following languages.

$$L_a = \{a^n b^n \mid n \geq 1\}, L_a \text{ is not regular.}$$

$$L_b = \{\omega \mid \omega \in \{a, b\}^*\}, L_b \text{ is regular.}$$

$$L_c = \{ab\}, L_c \text{ is regular.}$$

- (a) If $L_1 \subseteq L_2$ and L_1 is not regular, then L_2 is not regular.

Answer = False.

$L_a \subseteq L_b$, L_a is not regular but L_b is regular.

- (b) If $L_1 \subseteq L_2$ and L_2 is not regular, then L_1 is not regular.

Answer = False.

$L_c \subseteq L_a$, L_c is not regular but L_a is regular.

- (c) If L_1 and L_2 are non-regular then $L_1 \cup L_2$ is non-regular.

Answer = True.

We will not be able construct a FA for $L_1 \cup L_2$ if L_1 and L_2 are non-regular.

Example 3.7.4

Let L be any subset of 0^* . Prove that L^* is regular.

Solution : It is given that L is any subset of 0^* .

DFA of L is given by :



Fig. Ex. 3.7.4 : DFA for L

L^* can be considered to be regular if we can construct an FA for L^* . FA for L^* is given below :

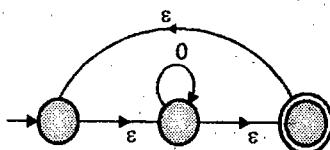


Fig. Ex. 3.7.4(a) : FA for L^*

Thus, L^* is regular

Example 3.7.5

Let L be a language. It is clear from the definition that $L^+ \subseteq L^*$. Under what circumstances are they equal ?

Solution : Let us assume that L is a language with ϵ belonging to it.

$$L^+ = LL^* = (\epsilon + L_1)L^*$$

where L_1 is a language obtained from L after deleting ϵ from it.

$$\begin{aligned} L^+ &= \epsilon L^* + L_1 L^* = L^* + L_1 L^* \\ &= L^*, \text{ as } L_1 L^* \subseteq L^* \end{aligned}$$

Thus, L^+ will be equal to L^* if ϵ is a member of L .

Example 3.7.6

Are the following True or False. Support your answer with proofs or counter examples.

- (i) If $L_1 \cup L_2$ is regular and L_1 is regular then L_2 is regular.
 (ii) If $L_1 \cdot L_2$ is regular and L_1 is regular then L_2 is regular.
 (iii) If L^* is regular, then L is regular.

Solution :

- (i) If $L_1 \cup L_2$ is regular and L_1 is regular then L_2 is regular.

- We can always construct a combined



- FA for $M(L_1 \cup L_2) - M(L_1)$
 $M(L_2)$ is same as $M(L_1 \cup L_2) - M(L_1)$
 $\therefore L_2$ is regular (True)
- (ii) If $L_1 \cdot L_2$ is regular and L_1 is regular then L_2 is regular.
- True [If L_2 is not regular then $L_1 \cdot L_2$ can not be regular]
- (iii) If L^* is regular, then L is regular.
- True
- Existence of an FA for L^* will trivially imply existence of an FA for L .

Example 3.7.7

If L is regular, prove L^T is also regular.

Solution : Let us assume $L = L(M)$ for some FA $= (Q, \Sigma, \delta, q_0, F)$

For simplicity, we can assume that F contains only one state q_n .

$$\therefore F = \{F_n\}$$

The set of states $Q = \{q_0, q_1, \dots, q_n\}$

The Finite Automate M^T accepting L^T can be derived from the given Finite Automata M . Steps for the same are given below :

1. q_n will become the start state in M^T .
 2. q_0 will become the final state in M^T .
 3. For every transition $\delta(q_i, a) = q_j$ (where $a \in \Sigma$), a transition $\delta(q_j, a) = q_i$ is added to M^T .
- It is clear that M^T is FA and the language of M^T must be regular.
- If ω is accepted by M , i.e. $\delta^*(q_0, \omega) = q_n$ then ω^T will be accepted by M^T by tracing the reverse path.
- Therefore, $\delta^*(q_n, \omega) = q_0$.

Syllabus Topic : Applications of Regular Expressions

3.8 Application of RE

SPPU - Dec. 15

University Question

Q. Write a short note on the applications of regular expressions. (Dec. 2015, 5 Marks)

3.8.1 R.E. in Unix

The UNIX regular expression lets us specify a group of characters using a pair of square brackets []. The rules for character classes are :

1. **[ab]** Stand for a + b
2. **[0 – 9]** Stand for a digit from 0 to 9
3. **[A – Z]** Stands for an upper-case letter

4. **[a – z]** Stands for a lower-case letter
 5. **[0 – 9A-Za – z]** Stands for a letter or a digit.
- The grep utility in UNIX, scans a file for the occurrence of a pattern and displays those lines in which the given pattern is found.
- For example :
- \$ grep president emp.txt
- It will list those lines from the file emp.txt which has the pattern "president". The pattern in grep command can be specified using regular expression.
6. ***** matches zero or more occurrences of previous character.
 7. **.** matches a single character.
 8. **[^ pqr]** Matches a single character which is not a p, q or r.
 9. **^ pat** Matches pattern **pat** at the beginning of a line
 10. **pat \$** Matches pattern at end of line.

Example

- a) The regular expression **[aA] g [ar] [ar]** wal stands for either "Agarwal" or 'agrawal'.
- b) **g*** stands for zero or more occurrences of g.
- c) **\$grep "A . * thakur" emp.txt** will look for a pattern starting with A. and ending with **thakur** in the file emp.txt.

3.8.2 Lexical Analysis

Lexical analysis is an important phase of a compiler. The lexical analyser scans the source program and converts it into a stream of tokens. A token is a string of consecutive symbol defining an entity.

For example a C statement **x = y + z** has the following tokens :

- x** – An identifier
- =** – Assignment operator
- y** – An identifier
- +** – Arithmetic operator +
- z** – An identifier

Keywords, identifiers and operators are common examples of tokens.

The UNIX utility **lex** can be used for writing of a lexical analysis program. Input to **lex** is a set of regular expressions for each type of token and output of **lex** is a C program for lexical analysis.

Context Free Grammar (CFG) and Languages

Syllabus

Introduction, Formal Definition of Grammar, Notations, Derivation Process: Leftmost Derivation, Rightmost Derivation, derivation trees, Context Free Languages, Ambiguous CFG, Removal of ambiguity, Simplification of CFG, Normal Forms, Chomsky Hierarchy.

Syllabus Topic : Introduction

4.1 An Example to Explain Grammar

SPPU - May 12

University Question

Q. Define and explain Grammar. (May 2012, 2 Marks)

- We have seen that every finite automata M accepts a language L, which is represented by L (M).
- We have seen that a regular language can be described by a regular expression.
- We have seen that there are several languages which are not regular.
 1. $L_1 = \{a^p \mid p \text{ is a prime}\}$ is not regular.
 2. $L_2 = \{a^n b^n \mid n \geq 0\}$ is not regular.
 3. $L_3 = \{a^{i^2} \mid i \geq 1\}$ is not regular.
 4. $L_4 = \{ww \mid w \in \{a, b\}^*\}$ is not regular.
 5. $L_5 = \{ww^R \mid w \in \{a, b\}^*\}$ is not regular.
- We have seen two ways for representing a language :
 1. Using finite automata.
 2. Using a regular expression.
- If a language is non-regular, it can not be represented either using a FA or using a regular expression. Hence there was a need for representing such languages.
- Grammar is another approach for representing a language.
 1. In this approach, a language is represented using a set of equations.
 2. Equations are recursive in nature.
 3. A finite automata has a set of states. A context free grammar has a set of variables.

4. Finite automata is defined over an alphabet. Similarly, a grammar is defined over a set of terminals.
5. A finite automata has a set of transitions; a grammar has a set of equations (productions).

Table 4.1.1 : A set of DFA represented using an equivalent set of productions

| Sr. No. | DFA | Equivalent grammar |
|---------|-----|--|
| 1. | | $X \rightarrow a$ |
| 2. | | $X \rightarrow aX$ $X \rightarrow b$ |
| 3. | | $X \rightarrow aX$ $X \rightarrow \epsilon$ |

- The production $X \rightarrow a$ should read as X produces a or X derives a or X gives a.
- The production $X \rightarrow aX$ is a way of defining a string containing one or more a's.
- If X stands for a set of strings containing one or more a's then a can be expressed in terms of X. The concept is shown below in Fig. 4.1.1.

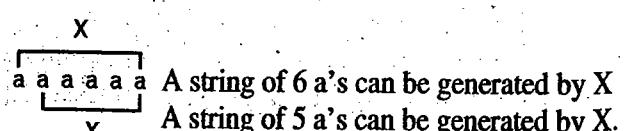


Fig. 4.1.1 : Recursive definition of $X \rightarrow aX$

A string of 6 a's = a followed by a string of 5 a's.

Or $X = aX$



Every recursive definition has a base case or termination case.

Let us consider the DFA given below in Fig. 4.1.2.

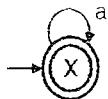


Fig. 4.1.2 : DFA taken as an example

The language of the DFA is given below

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

Now, let us consider the grammar given below and see if it can generate the language generated by the DFA of Fig. 4.1.2.

$$X \rightarrow aX \quad \dots(4.1)$$

$$X \rightarrow \epsilon \quad \dots(4.2)$$

- The empty string ϵ can be generated by the production $X \rightarrow \epsilon$
- A string "a" can be generated as shown in Fig. 4.1.3.

First by writing X as aX and then by replacing X on the right hand side with an ϵ .

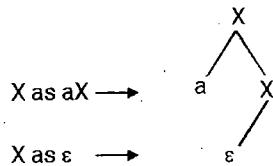


Fig. 4.1.3

- A string "aa" can be generated as shown in Fig. 4.1.4.

First by writing X as aX , secondly by replacing X on RHS with aX and finally replacing X on RHS with ϵ .

- Thus the language generated by the set of productions $\{X \rightarrow aX, X \rightarrow \epsilon\}$ is

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

The production $X \rightarrow \epsilon$, is for termination of recursion.

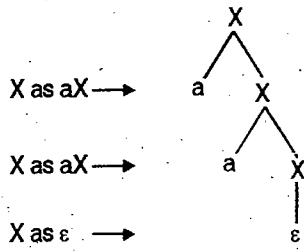


Fig. 4.1.4

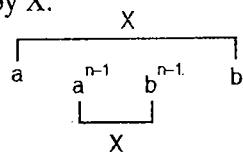
A non regular language can be expressed using a set of productions

Let us consider a language

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

We know from pumping lemma that the above language is not regular; but this language can be represented using a set of productions. It will be worthwhile to understand the recursive nature of the above language.

1. A string $0, 1 \in L$ [Base case or termination case]
2. $a^n b^n$ can be written as $a(a^{n-1} b^{n-1})b$
If $a^n b^n$ is generated by X then $a^{n-1} b^{n-1}$ can also be generated by X .



Now, we can say that,

$$X \rightarrow aXb$$

i.e. If we have a string of the form $a^n b^n$, after removing the first a and the last b , the string remains in the form $a^n b^n$.

Thus the language.

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

can be represented using

1. $X \rightarrow aXb \quad \dots(4.3)$
2. $X \rightarrow ab \quad \dots(4.4)$

The production $X \rightarrow aXb$ can recursively generate a string $a^n b^n$ and the second production $X \rightarrow ab$ is used for termination of recursion.

Let us try to generate a string $a^3 b^3$ using the two productions (4.3) and (4.4).

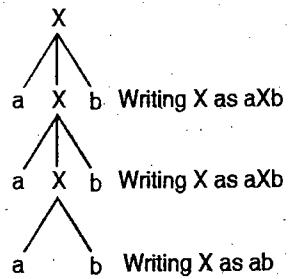


Fig. 4.1.5 : Generation of $a^3 b^3$

Conclusion : Grammar can be used to represent both.

1. Regular languages
2. Non-regular languages.



Syllabus Topic : Formal Definition of Grammar

4.2 Context Free Grammar

SPPU - Dec. 14, Aug. 15, Dec. 15

University Questions

Q. Describe CFG with suitable examples.

(Dec. 2014, 2 Marks)

Q. Define Context free Grammar with example.

(Aug. 2015 (In-Sem.), Dec. 2015, 2 Marks)

A context free grammar G is a quadruple (V, T, P, S) ,

Where,

V is a set of variables.

T is a set of terminals.

P is a set of productions.

S is a special variable called the start symbol $S \in V$.

A production is of the form

$V_i \rightarrow \alpha_i$ where $V_i \in V$ and α_i is a string of terminals and variables.

Syllabus Topic : Notations

4.2.1 Notations

- Terminals are denoted by lower case letters a, b, c ... or digits 0, 1, 2 ... etc.
- Non-terminals (variables) are denoted by capital letters A, B, ..., V, W, X ...
- A string of terminals or a word $w \in L$ is represented using u, v, w, x, y, z.
- A sentential form is a string of terminals and variables and it is denoted by α, β, γ etc.

CFG explained through an example

Let us consider English sentences of the form :

- Mohan eats.
- Soham plays.
- Ram reads.

The first word of in the above sentences is a noun and the second word is a verb. A sentence of the above form can be written as

$\langle \text{sentence} \rangle \rightarrow \langle \text{noun} \rangle \langle \text{verb} \rangle$

Here, noun can be replaced with Mohan, Soham or Ram and $\langle \text{verb} \rangle$ can be replaced with reads, plays or eats. We can write :

$\langle \text{noun} \rangle \rightarrow \text{Mohan}$

$\langle \text{noun} \rangle \rightarrow \text{Soham}$

$\langle \text{noun} \rangle \rightarrow \text{Ram}$

$\langle \text{verb} \rangle \rightarrow \text{eats}$

$\langle \text{verb} \rangle \rightarrow \text{plays}$

$\langle \text{verb} \rangle \rightarrow \text{reads}$

In the above example :

- $\langle \text{sentence} \rangle, \langle \text{noun} \rangle$ and $\langle \text{verb} \rangle$ are variables or non-terminals.
- Mohan, Soham, Ram, eats, plays and reads are terminals.
- The variable $\langle \text{sentence} \rangle$ is the start symbol as a sentence will be formed using the start symbol $\langle \text{sentence} \rangle$.

Formally, the grammar can be written as :

$$G = (V, T, P, S), \text{ where}$$

$$V = \{\langle \text{sentence} \rangle, \langle \text{noun} \rangle, \langle \text{verb} \rangle\}$$

$$T = \{\text{Mohan}, \text{Soham}, \text{Ram}, \text{eats}, \text{plays}, \text{reads}\}$$

$$P = \{\langle \text{sentence} \rangle \rightarrow \langle \text{noun} \rangle \langle \text{verb} \rangle,$$

$$\langle \text{noun} \rangle \rightarrow \text{Mohan},$$

$$\langle \text{noun} \rangle \rightarrow \text{Soham},$$

$$\langle \text{noun} \rangle \rightarrow \text{Ram},$$

$$\langle \text{verb} \rangle \rightarrow \text{eats},$$

$$\langle \text{verb} \rangle \rightarrow \text{plays},$$

$$\langle \text{verb} \rangle \rightarrow \text{reads}$$

}

$$S = \langle \text{sentence} \rangle$$

Several productions of $\langle \text{noun} \rangle$ and $\langle \text{verb} \rangle$ can be merged together and the set of productions can be re-written as :

$$P = \{\langle \text{sentence} \rangle \rightarrow \langle \text{noun} \rangle \langle \text{verb} \rangle,$$

$$\langle \text{noun} \rangle \rightarrow \text{Mohan} \mid \text{Soham} \mid \text{Ram},$$

$$\langle \text{verb} \rangle \rightarrow \text{eats} \mid \text{plays} \mid \text{reads}$$

}

Syllabus Topic : Context Free Languages

4.2.2 The Language of a Grammar

Every grammar generates a language. A word of a language is generated by applying productions a finite number of times. Derivation of a string should start from the start symbol and the final string should consist of terminals.

If G is a grammar with start symbol S and set of terminals T, then the language of G is the set

$$L(G) = \left\{ w \mid w \in T^* \text{ and } S \xrightarrow[G]{*} w \right\}$$



If the production rule is applied once, then we write $\alpha \Rightarrow \beta$. When it is applied a number of times G then we write $\alpha \xrightarrow{G} \beta^*$

Derivations are represented either in the

1. Sentential form OR
2. Parse tree form.

Syllabus Topic : Derivation Process : Leftmost Derivation, Rightmost Derivation

4.2.2.1 Sentential Form

Let us consider a grammar given below :

$$S \rightarrow A1B \quad (\text{Production 4.5})$$

$$A \rightarrow 0A \mid \epsilon \quad (\text{Production 4.6})$$

$$B \rightarrow 0B \mid 1B \mid \epsilon \quad (\text{Production 4.7})$$

where, G is given by (V, T, P, S)

with, V = {S, A, B}

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

P = {Productions 4.5, 4.6 and 4.7}

S = Start symbol

Let us try to generate the string 00101 from the given grammar.

$$\begin{aligned} S &\rightarrow A1B && [\text{Starting production}] \\ &\rightarrow 0A1B && [\text{Using the production } A \rightarrow 0A] \\ &\rightarrow 00A1B && [\text{Using the production } A \rightarrow 0A] \\ &\rightarrow 001B && [\text{Using the production } A \rightarrow \epsilon] \\ &\rightarrow 0010B && [\text{Using the production } B \rightarrow 0B] \\ &\rightarrow 00101B && [\text{Using the production } B \rightarrow 1B] \\ &\rightarrow 00101 && [\text{Using the production } B \rightarrow \epsilon] \end{aligned}$$

Thus the string 00101 $\in L(G)$.

In sentential form, derivation starts from the start symbol through a finite application of productions.

A string α derived so far consists of terminals and non-terminals.

$$S \xrightarrow[G]{*} \alpha \mid \alpha \in (V \cup T)^*$$

- A final string consists of terminals.
- In left sentential form, leftmost symbol is picked up for expansion.
- In right sentential form, rightmost symbol is picked up for expansion.
- A string can be derived in many ways. But we restrict ourselves to :
 1. Leftmost derivation.
 2. Rightmost derivation.

In leftmost derivation the leftmost variable of α (sentential form) is picked for expansion.

In rightmost derivation the rightmost variable of α (sentential form) is picked for expansion.

Example 4.2.1

For the grammar given below

$$S \rightarrow A1B \quad A \rightarrow 0A \mid \epsilon \quad B \rightarrow 0B \mid 1B \mid \epsilon$$

Give leftmost and rightmost derivation of the string 1001.

Solution :

- (i) Leftmost derivation of 1001 (Leftmost variable is picked up for expansion.)

$$\begin{aligned} S &\rightarrow A1B && [\text{Derivation starts from the start symbol}] \\ &\rightarrow 1B && [\text{Using the production } A \rightarrow \epsilon] \\ &\rightarrow 10B && [\text{Using the production } B \rightarrow 0B] \\ &\rightarrow 100B && [\text{Using the production } B \rightarrow 0B] \\ &\rightarrow 1001B && [\text{Using the production } B \rightarrow 1B] \\ &\rightarrow 1001 && [\text{Using the production } B \rightarrow \epsilon] \end{aligned}$$

- (ii) Rightmost derivation of 1001 (Rightmost variable is picked up for expansion)

$$\begin{aligned} S &\rightarrow A1B && [\text{Derivation starts from the start symbol}] \\ &\rightarrow A10B && [\text{Using the production } B \rightarrow 0B] \\ &\rightarrow A100B && [\text{Using the production } B \rightarrow 0B] \\ &\rightarrow A1001B && [\text{Using the production } B \rightarrow 1B] \\ &\rightarrow A1001 && [\text{Using the production } B \rightarrow \epsilon] \\ &\rightarrow 1001 && [\text{Using the production } A \rightarrow \epsilon] \end{aligned}$$

Syllabus Topic : Derivation Trees

4.2.2.2 Parse Tree

SPPU - May 16

University Question

Q. Define derivation tree with suitable example.

(May 2016, 2 Marks)

A set of derivations applied to generate a word can be represented using a tree. Such a tree is known as a parse tree. A parse tree representation gives us a better understanding of :

1. Recursion
2. Grouping of symbols

A parse tree is constructed with the following condition :

1. Root of the tree is represented by start symbol.
 2. Each interior mode is represented by a variable belonging to V.
 3. Each leaf mode is represented by a terminal or ϵ .
- A string generated by a parse tree is seen from left to right.

**Example 4.2.2**

For the grammar given below

$$S \rightarrow A1B \quad A \rightarrow 0A \mid \epsilon \quad B \rightarrow 0B \mid 1B \mid \epsilon$$

Give parse tree for leftmost and rightmost derivation of the string 1001.

Solution :

(i) Parse tree for leftmost derivation of 1001.

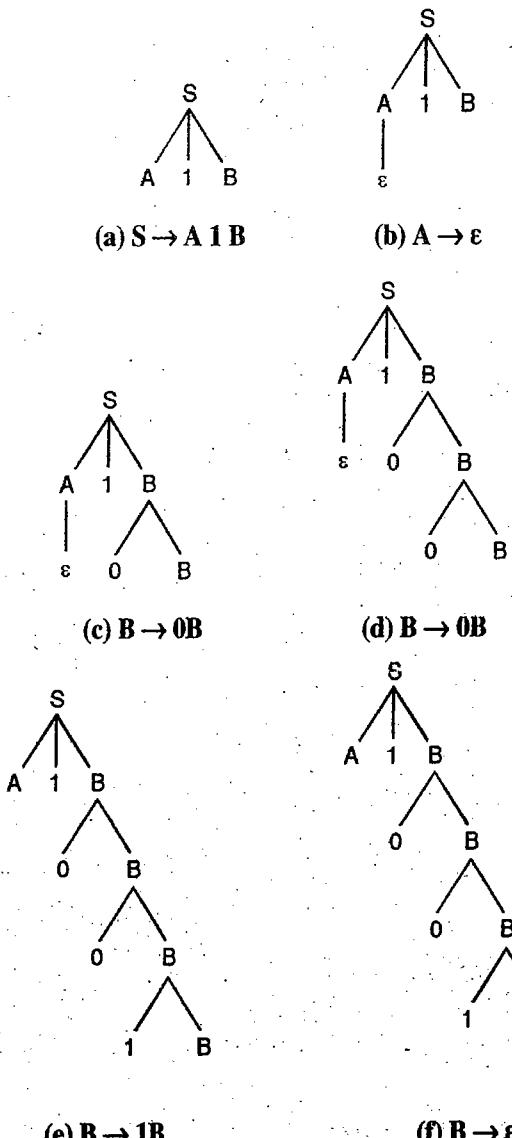


Fig. Ex. 4.2.2

Leftmost derivation of 1001 is shown by series of Fig. Ex. 4.2.2(a) to Fig. Ex. 4.2.2(f).

(ii) Parse tree for rightmost derivation of 1001.

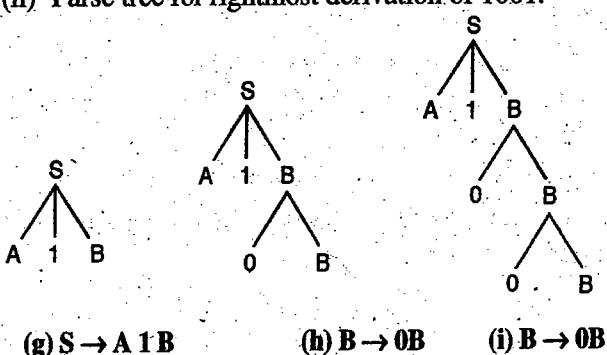


Fig. Ex. 4.2.2

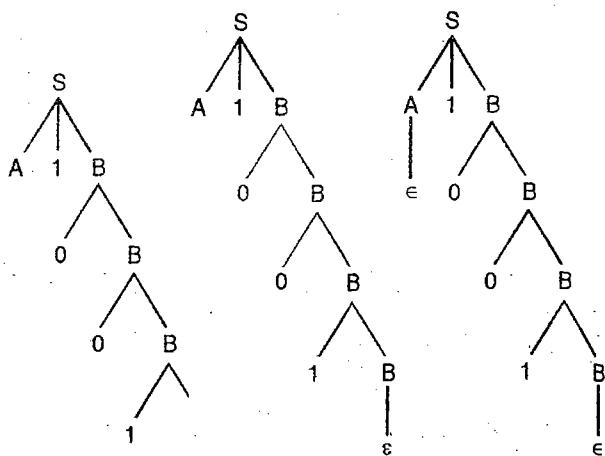


Fig. Ex. 4.2.2

Rightmost derivation of 1001 is shown by a series of Fig. Ex. 4.2.2(g) to Ex. 4.2.2(l).

It may be noted that the final parse trees Fig. Ex. 4.2.2(l) and Ex. 4.2.2(l) are identical.

Example 4.2.3

For the grammar given below $S \rightarrow 0S1 \mid 01$. Give derivation of 000111.

Solution :

1. Derivation in sentential form

$$S \rightarrow 0S1 \quad [\text{Using production } S \rightarrow 0S1]$$

$$S \rightarrow 00S11 \quad [\text{Using production } S \rightarrow 0S1]$$

$$S \rightarrow 000111 \quad [\text{Using production } S \rightarrow 01]$$

2. Derivation using parse tree.

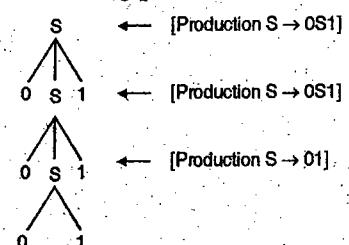


Fig. Ex. 4.2.3 : Derivation of 000111

Example 4.2.4

Find whether the string aabbb is in $L = L(G)$, where G is given by

$$S \rightarrow XY, X \rightarrow YY1a, Y \rightarrow XY \mid b$$

Solution :

Parse tree for the string aabbb is shown in Fig. Ex. 4.2.4.

Hence, $aabbb \in L(G)$.

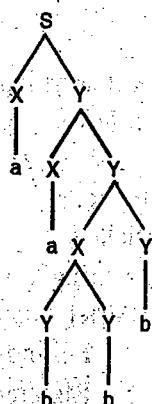


Fig. Ex. 4.2.4 : Derivation of aabbb

**Example 4.2.5**

For the grammar given below

$$E \rightarrow E + T \mid T \quad T \rightarrow T * F \mid F \quad F \rightarrow (E) \mid a \mid b$$

Give the derivation of $(a + b) * a + b$.

Solution : For the grammar given in Example 4.2.5,

$$\text{Set of variables } V = \{E, T, F\}$$

$$\text{Set of terminals } \Sigma = \{+, *, (,), a, b\}$$

$$\text{Set of productions } P = \left\{ \begin{array}{l} E \rightarrow E + T \mid T, \\ T \rightarrow T * F \mid F, \\ F \rightarrow (E) \mid a \mid b \end{array} \right\}$$

Start symbol = E

(i) Derivation in sentential form

$$\begin{aligned} E &\rightarrow E + T & [\text{Using production } E \rightarrow E + T] \\ &\rightarrow T + T & [\text{Using production } E \rightarrow T] \\ &\rightarrow T * F + T & [\text{Using production } T \rightarrow T * F] \\ &\rightarrow F * F + T & [\text{Using production } T \rightarrow F] \\ &\rightarrow (E) * F + T & [\text{Using production } F \rightarrow (E)] \\ &\rightarrow (E + T) * F + T & [\text{Using production } E \rightarrow E + T] \\ &\rightarrow (T + T) * F + T & [\text{Using production } E \rightarrow T] \\ &\rightarrow (F + F) * F + F & [\text{Using production } T \rightarrow F] \\ &\rightarrow (a + b) * a + b & [\text{Using production } F \rightarrow a \mid b] \end{aligned}$$

(ii) Derivation using parse tree :

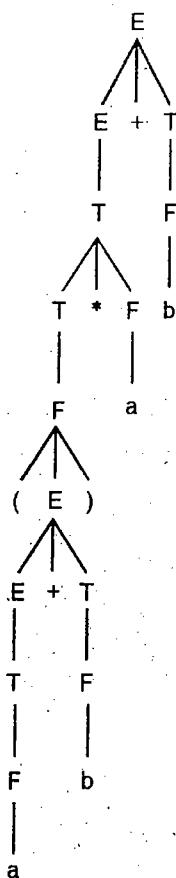


Fig. Ex. 4.2.5 : Parse tree for $(a + b) * a + b$

Example 4.2.6

Construct the parse trees for the strings using specified derivation format for given grammar G.

$$G = (\{S, A, B\}, \{a, b\}, P, \{S\})$$

$$P = \{S \rightarrow aB, S \rightarrow bA,$$

$$A \rightarrow a, A \rightarrow aS, A \rightarrow bAA,$$

$$B \rightarrow b, B \rightarrow bS, B \rightarrow aBB\}$$

Strings :

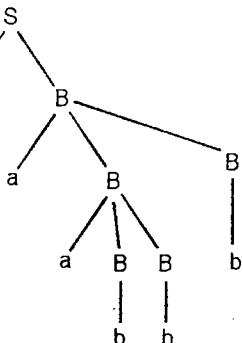
(I) a a a b b b (leftmost derivation)

(II) a b a b a b b a (rightmost derivation).

Solution :

(I) aaabbb

Parse tree

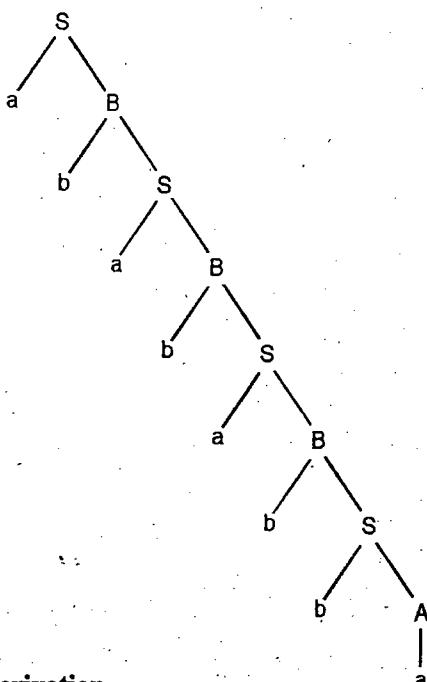


leftmost derivation :

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaaBBB \rightarrow aaabbB \\ &\rightarrow aaabbb \end{aligned}$$

(II) abababba

Parse tree



Rightmost derivation

$$\begin{aligned} S &\rightarrow aB \rightarrow abs \rightarrow abaB \rightarrow ababs \rightarrow ababaB \\ &\rightarrow abababs \rightarrow abababbaA \rightarrow abababba \end{aligned}$$

Example 4.2.7 SPPU - Dec. 14, Dec. 15, 4 Marks

Construct the parse trees for the strings using specified derivation format for given grammar G.

$$G = (\{S, A, B\}, \{a, b\}, P, \{S\})$$

$$P = \{S \rightarrow aB, S \rightarrow bA,$$

$$A \rightarrow a, A \rightarrow aS, A \rightarrow bAA,$$



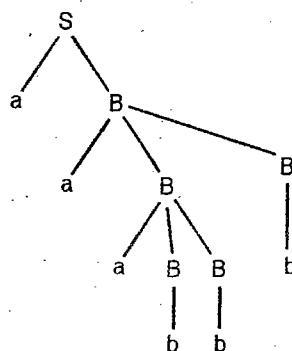
$B \rightarrow b, B \rightarrow bS, B \rightarrow aBB\}$

Strings : (I) a a a b b b (rightmost derivation) (II) a a b a b b (leftmost derivation)

Solution :

(I) aaabbb (rightmost derivation)

Parse Tree

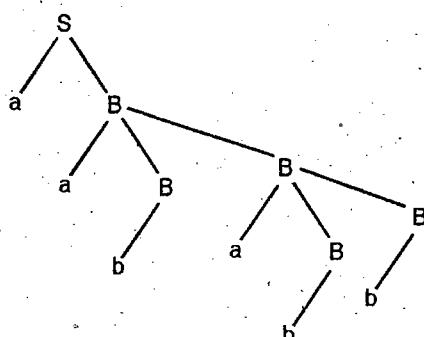


Rightmost derivation

$S \rightarrow aB \rightarrow aaBB \rightarrow aaBb \rightarrow aaaBBb \rightarrow aaaBbb \rightarrow aaabbb$

(II) aababb (leftmost derivation)

Parse Tree



Leftmost derivation

$S \rightarrow aB \rightarrow aaBB \rightarrow aabB \rightarrow aabaBB \rightarrow aababb \rightarrow aababb$

Example 4.2.8

Consider the following grammar

$S \rightarrow aAS \mid a \quad A \rightarrow SbA \mid SS \mid ba$

Derive the string aabbbaa using

- (i) Leftmost derivation (ii) Rightmost derivation.

Solution :

(i). Leftmost derivation :

| | |
|----------------------|-----------------------------------|
| $S \rightarrow aAS$ | [Production $S \rightarrow aAS$] |
| $\rightarrow aSbAS$ | [Production $A \rightarrow SbA$] |
| $\rightarrow aabAS$ | [Production $S \rightarrow a$] |
| $\rightarrow aabbAS$ | [Production $A \rightarrow ba$] |
| $\rightarrow aabbba$ | [Production $S \rightarrow a$] |

(ii). Rightmost derivation :

| | |
|----------------------|-----------------------------------|
| $S \rightarrow aAS$ | [Production $S \rightarrow aAS$] |
| $\rightarrow aAa$ | [Production $S \rightarrow a$] |
| $\rightarrow aSbAa$ | [Production $A \rightarrow SbA$] |
| $\rightarrow aSbbAa$ | [Production $A \rightarrow ba$] |
| $\rightarrow aabbba$ | [Production $S \rightarrow a$] |

4.2.3 Writing Grammar for a Language

SPPU - Dec. 13

University Question

Q. Construct CFG for language with $\Sigma = \{a, b\}$

$L = \{w \mid w \text{ is in } \Sigma^* \text{ and palindrome strings of ODD LENGTH}\}$ (Dec. 2013, 3 Marks)

Productions for an infinite language are written recursively. A production is called recursive if its left side variable occurs on its right side.

For example :

(1) $S \rightarrow aS$ is recursive

(2) $S \rightarrow b \mid aA$

$A \rightarrow c \mid bS$ is indirectly recursive

as, $S \rightarrow aA \rightarrow abS$ [S gives A and A gives S]

$A \rightarrow bS \rightarrow baA$ [A gives S and S gives A]

Thus a recursive grammar can be written either using recursive productions or using indirectly recursive productions. Grammar for some basic languages is given below.

(1) Language $L = \{\epsilon, a, aa, \dots\}$

The productions required to generate the above language are :

$S \rightarrow aS \quad S \rightarrow \epsilon$

The production, $S \rightarrow \epsilon$ is taken as ϵ is a member of L.

The production, $S \rightarrow aS$ can generate one or more a's.

Example : Generation of the string aaaa.

Sentential form

| | |
|---------------------|--|
| $S \rightarrow aS$ | [Using production $S \rightarrow aS$] |
| $\rightarrow aaS$ | [Using production $S \rightarrow aS$] |
| $\rightarrow aaaS$ | [Using production $S \rightarrow aS$] |
| $\rightarrow aaaaS$ | [Using production $S \rightarrow aS$] |
| $\rightarrow aaaa$ | [Using production $S \rightarrow \epsilon$] |

Parse tree

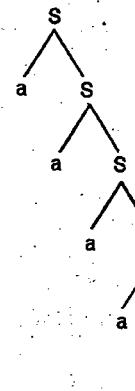


Fig. 4.2.1 : Parse tree for aaaa

(2) Language $L = \{a, aa, aaa, \dots\}$

The productions required to generate the above language are :

$$S \rightarrow aS$$

$$S \rightarrow a$$

The production $S \rightarrow aS$ can generate one or more a's. The production, $S \rightarrow a$ is for termination of recursion.

Example : Generation of the string aaaa

Sentential form

$$S \rightarrow aS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaaS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaaa \quad [\text{Using production } S \rightarrow aS]$$

Parse tree

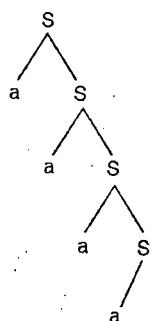


Fig. 4.2.2 : Parse tree for aaaa

(3) Language $L = \{b, ab, aab, aaab, \dots\}$

The productions required to generate the above languages are :

$$S \rightarrow aS$$

$$S \rightarrow b$$

The production $S \rightarrow aS$ can generate one or more a's. The production, $S \rightarrow b$ is for termination of recursion and thus generating zero or more a's followed by ab.

Example : Generation of the string aaab.

Sentential form

$$S \rightarrow aS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaaS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow aaab \quad [\text{Using production } S \rightarrow b]$$

Parse tree

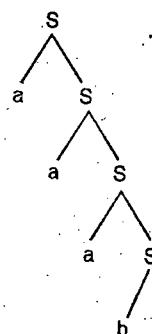


Fig. 4.2.3 : Parse tree for aaab

(4) Language $L = \{w \in \{a, b\}^*\}$

The above language can also be written as

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

Thus any string that can be derived from the alphabet $\{a, b\}$ belongs to L. This language can also be generated by :



The DFA $M =$

or by the regular expression $(a + b)^*$.

The productions required to generate the above language are :

$$S \rightarrow aS$$

$$S \rightarrow bS$$

$$S \rightarrow \epsilon$$

The two productions $S \rightarrow aS$, $S \rightarrow bS$ can generate any number of a's or b's with the next character as either a or b. The production $S \rightarrow \epsilon$ is for termination of recursion. Since ϵ is also a member of L, $S \rightarrow \epsilon$ is taken for termination of recursion.

Example : Generation of the string abba

Sentential form

$$S \rightarrow aS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow abS \quad [\text{Using production } S \rightarrow bS]$$

$$\rightarrow abbS \quad [\text{Using production } S \rightarrow bS]$$

$$\rightarrow abbaS \quad [\text{Using production } S \rightarrow aS]$$

$$\rightarrow abba \quad [\text{Using production } S \rightarrow \epsilon]$$

Parse tree

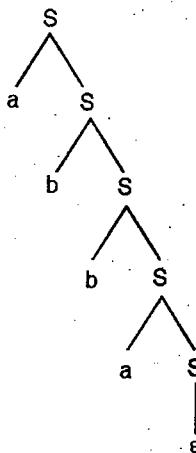


Fig. 4.2.4 : Parse tree for abba

(5) Language $L = \{\epsilon, ab, aabb, \dots, a^n b^n\}$

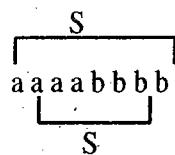
The above language can also be described as a language of words in which every word consists of any number of a's followed by equal number of b's.

The productions required to generate the above language are :

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

The recursion involved in the above language is explained below.



Both the strings $a^4 b^4$ and $a^3 b^3$ are generated by S. $a^4 b^4$ can be written as $aa^3 b^3 b$ and thus we have the recursive relation $S \rightarrow aSb$.

The production, $S \rightarrow aSb$ can generate any number of a's followed by equal number of b's. The production $S \rightarrow \epsilon$ is for termination of recursion. Since ϵ is a member of the given language, $S \rightarrow \epsilon$ is taken for termination of recursion.

Example : Generation of the string aaabbb:

Sentential form

$$\begin{aligned} S &\rightarrow aSb & [\text{Using production } S \rightarrow aSb] \\ &\rightarrow aaSbb & [\text{Using production } S \rightarrow aSb] \\ &\rightarrow aaaSbbb & [\text{Using production } S \rightarrow aSb] \\ &\rightarrow aaabbb & [\text{Using production } S \rightarrow \epsilon] \end{aligned}$$

Parse tree

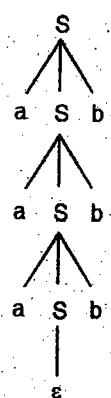


Fig. 4.2.5 : Parse tree for aaabbb or $a^3 b^3$

(6) Language $L = \{ab, aabb, \dots, a^n b^n\}$

The productions required to generate the above language are :

$$S \rightarrow aSb$$

$$S \rightarrow ab$$

The production, $S \rightarrow aSb$ can generate a string of the form $a^n b^n$. The production $S \rightarrow ab$ is for termination of recursion.

Example : Generation of the string aaabbb

Sentential form

$$\begin{aligned} S &\rightarrow aSb & [\text{Using production } S \rightarrow aSb] \\ &\rightarrow aaSbb & [\text{Using production } S \rightarrow aSb] \\ &\rightarrow aaabbb & [\text{Using production } S \rightarrow ab] \end{aligned}$$

Parse tree

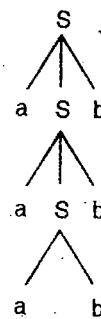


Fig. 4.2.6 : Parse tree for aaabbb or $a^3 b^3$

(7) Language $L = \{w \in \{a, b\}^* | w \text{ is a palindrome of odd length}\}$

SPPU - Dec. 13

- o String 'a' is palindrome of odd length.
- o String 'b' is a palindrome of odd length
- o String abababa is palindrome. The palindrome nature of abababa can be understood as given shown in Fig. 4.2.7.

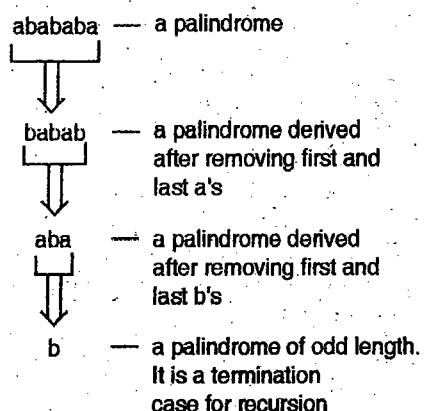


Fig. 4.2.7

Thus a palindrome can be generated recursively using the two productions :

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

Since, we are considering odd length palindromes, S should terminate in either 'a' or 'b' and the productions are :

$$S \rightarrow a, S \rightarrow b.$$

Thus the productions required to generate the above language are :

$$S \rightarrow aSa \quad S \rightarrow bSb$$

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



Or, in short, it can be written as

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

Example : Generation of the string abababa.

Sentential form

- $S \rightarrow aSa$ [Using production $S \rightarrow aSa$]
- $\rightarrow abSba$ [Using production $S \rightarrow bSb$]
- $\rightarrow abaSaba$ [Using production $S \rightarrow aSa$]
- $\rightarrow abababa$ [Using production $S \rightarrow b$]

Parse tree

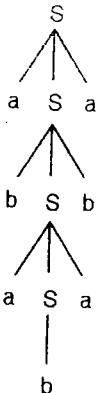


Fig. 4.2.8 : Parse tree for abababa

- (8) Language $L = \{w \in \{a, b\}^* \mid w \text{ is an even length palindrome with } |w| > 0\}$

The productions required to generate the above language are :

- $S \rightarrow aSa$
- $S \rightarrow bSb$
- $S \rightarrow bb$
- $S \rightarrow aa$

The productions $S \rightarrow aa$, $S \rightarrow bb$ are for termination of recursion. It may be noted that smallest strings generated by the above grammar are aa and bb. The two productions $S \rightarrow aSa$ and $S \rightarrow bSb$ can generate a palindrome of arbitrary length.

Example : Generation of the string ababbaba

Sentential form

- $S \rightarrow aSa$ [Using production $S \rightarrow aSa$]
- $\rightarrow abSba$ [Using production $S \rightarrow bSb$]
- $\rightarrow abaSaba$ [Using production $S \rightarrow aSa$]
- $\rightarrow ababbaba$ [Using production $S \rightarrow bb$]

Parse tree or derivation tree

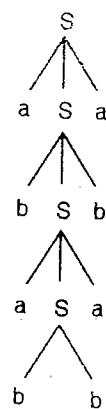


Fig. 4.2.9 : Derivation tree for ababbaba

- (9) Language $L = \{w \in \{a, b\}^* \mid w \text{ is a palindrome of either even length or odd length with } |w| > 0\}$

The productions required to generate the above language are :

- $S \rightarrow aSa \mid bSb \mid aa \mid bb \mid a \mid b$
- $S \rightarrow aa$ or $S \rightarrow bb$ are for termination of recursion for even length palindromes.
- $S \rightarrow a$ and $S \rightarrow b$ are for termination of recursion for odd length palindromes.
- $S \rightarrow aSa$ or $S \rightarrow bSb$ can generate a palindrome of an arbitrary length.

4.2.3.1 Union Rule for Grammar

If a language L_1 is generated by a grammar with start symbol S_1 and L_2 is generated by a grammar with start symbol S_2 then the union of the languages $L_1 \cup L_2$ can be generated with start symbol S , where

$$S \rightarrow S_1 \mid S_2$$

Example : Let the languages L_1 and L_2 are given as below :

$$L_1 = \{a^n \mid n > 0\}$$

$$L_2 = \{b^n \mid n > 0\}$$

Productions for L_1 are :

$$S_1 \rightarrow aS_1 \mid a$$

Productions for L_2 are :

$$S_2 \rightarrow bS_2 \mid b$$

Then the productions for $L = L_1 \cup L_2$ can be written as :

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow aS_1 \mid a$$

$$S_2 \rightarrow bS_2 \mid b$$



4.2.3.2 Concatenation Rule for Grammar

If a language L_1 is generated by a grammar with start symbol S_1 and L_2 is generated by a grammar with start symbol S_2 then the concatenation (product) of the languages $L_1 \cdot L_2$ can be generated with start symbol S , where

$$S \rightarrow S_1 S_2$$

Example : Let the languages L_1 and L_2 are given as below :

$$L_1 = \{a^n \mid n > 0\}$$

$$L_2 = \{b^n \mid n > 0\}$$

Productions for L_1 are :

$$S_1 \rightarrow a S_1 \mid a$$

Productions for L_2 are :

$$S_2 \rightarrow b S_2 \mid b$$

Then the productions for $L = L_1 \cdot L_2$ can be written as :

$$S \rightarrow S_1 S_2$$

$$S_1 \rightarrow a S_1 \mid a$$

$$S_2 \rightarrow b S_2 \mid b$$

Example 4.2.9

Give a context free grammar for the following language.

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

Solution : The $(0+1)^*$ can be generated by the following productions.

$$S_1 \rightarrow 0 S_1 \mid 1 S_1 \mid \epsilon$$

Using the product rule, we can write a set of productions for the language

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

$$P = \{ S \rightarrow 0 S_1 01 S_1 \mid 1 \}$$

$$S_1 \rightarrow 0 S_1 \mid 1 S_1 \mid \epsilon$$

}

Where, Set of variables $V = \{S, S_1\}$

Set of terminals $T = \{0, 1\}$

Start symbol = S

Example 4.2.10

Construct the context free grammar corresponding to the regular expression.

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

Solution : Grammar for $(0+1)$ is given by

$$S_1 \rightarrow 0 \mid 1 \text{ [union rule]}$$

Grammar for 1^* is given by

$$S_2 \rightarrow 1 S_2 \mid \epsilon$$

Grammar for $(1+(01)^*)$ is given by

$$S_3 \rightarrow 1 \mid S_4 \text{ where } S_4 \text{ stands for } (01)^*$$

$$S_4 \rightarrow 01 S_4 \mid \epsilon$$

Using the concatenation rule, we can write a set of productions for

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

$$P = \{ S \rightarrow S_1 S_2 S_3$$

$$S_1 \rightarrow 0 \mid 1$$

$$S_2 \rightarrow 1 S_2 \mid \epsilon$$

$$S_3 \rightarrow 1 \mid S_4$$

$$S_4 \rightarrow 01 S_4 \mid \epsilon$$

}

Where, Set of variables $V = \{S_1, S_2, S_3, S_4\}$

Set of terminals $T = \{0, 1\}$

Start symbol = S

Example 4.2.11

Write a CFG to represent a language defined by the regular expression a^*b^* .

Solution : We can write the set of productions using the rule of concatenation.

$$S \rightarrow S_1 S_2$$

$$S_1 \rightarrow a S_1 \mid \epsilon$$

$$S_2 \rightarrow b S_2 \mid \epsilon$$

Note : S_1 generates a^* and S_2 generates b^*

∴ The required CFG

$$G = (\{S, S_1, S_2\}, \{a, b\}, \{S \rightarrow S_1 S_2, S_1 \rightarrow a S_1 \mid \epsilon, S_2 \rightarrow b S_2 \mid \epsilon\}, S)$$

Example 4.2.12

Construct a grammar for L which consists of strings over $\{0, 1\}^*$ with at least one occurrence of '000'.

Solution : The regular expression for the set of given strings is $(0+1)^*000(0+1)^*$.

We can write the set of productions using the rule of concatenation.

$$S \rightarrow S_1 000 S_1$$

$$S_1 \rightarrow 0 S_1 \mid 1 S_1 \mid \epsilon$$

Note : S_1 generates strings from $(0+1)^*$

∴ The required CFG

$$G = (\{S, S_1\}, \{0, 1\}, \{S \rightarrow S_1, 000 S_1, S_1 \rightarrow 0 S_1 \mid 1 S_1 \mid \epsilon\}, S)$$

**Example 4.2.13**

Give the CFG for $L = \{a^i b^j c^q \mid i \leq j \leq 2i, i \geq 1\}$

Solution :

A string of L has to meet the following requirements

- A finite number of a's followed by a finite number of b's.
- Number of b's should be at least as many as number of a's.
- Number of b's should not be more than twice the number of a's.

The set of productions for the above language are given by :

$$\begin{aligned} P = \{ & \\ S \rightarrow aSb & \quad [\text{Number of b's} = \text{Number of a's}] \\ S \rightarrow aSbb & \quad [\text{Number of b's} = 2 \times \text{Number of a's}] \\ S \rightarrow ab \mid abb & \quad [\text{Termination cases}] \\ \} & \end{aligned}$$

Where, Set of variables $V = \{S\}$

Set of terminals $T = \{a, b\}$

Start symbol = S.

Example 4.2.14 SPPU - Dec. 12, 6 Marks

Construct a CFG to generate following language

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

Solution :

$$S \rightarrow XY \quad [\text{where X generates the string } 0^m 1^n \text{ and Y generates the string } 2^n \mid m \geq 1 \text{ and } n \geq 0]$$

$$X \rightarrow 0X1 \mid 01$$

$$Y \rightarrow 2Y \mid \epsilon$$

Example 4.2.15

Obtain a grammar to generate the language

$$L = \{a^{n+2} b^n \mid n \geq 0 \text{ and } m > n\}$$

Solution : The above language can be written as

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

The set of productions for the above language are given by :

$$\begin{aligned} P = \{ & \\ S \rightarrow aSb \mid aa & \\ \} & \end{aligned}$$

Thus the required grammar

$$G = (\{S\}, \{a, b\}, \{S \rightarrow aSb \mid aa\}, S)$$

Example 4.2.16 SPPU - Aug. 15, 3 Marks

Give CFG for $L = \{a^i b^j c^q \mid i + j = q; i, j \geq 1\}$

Solution :

A string of the form $a^i b^j c^q$ with $i + j = q$ can be written as $a^i b^j c^{i+j}$

A string of the form $a^i c^j$ can be generated with the help of the following productions :

$$S \rightarrow aSc$$

$$S \rightarrow X$$

Recursion is terminated by the production $S \rightarrow X$, where X gives a string of the form $b^j c^j$.

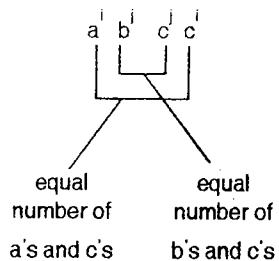


Fig. Ex. 4.2.16

$$X \rightarrow bXc$$

$$X \rightarrow bc$$

Thus, the set of productions for the above language are given by :

$$\begin{aligned} P = \{ & S \rightarrow aSc \mid X \\ & X \rightarrow bXc \mid bc \\ \} & \end{aligned}$$

where, Set of variables $V = \{S, X\}$

Set of terminals $T = \{a, b\}$

Start symbol = S

Working of the above grammar can be understood with the help of the following example.

Example : Generation of the string aabbccccc

Parse tree

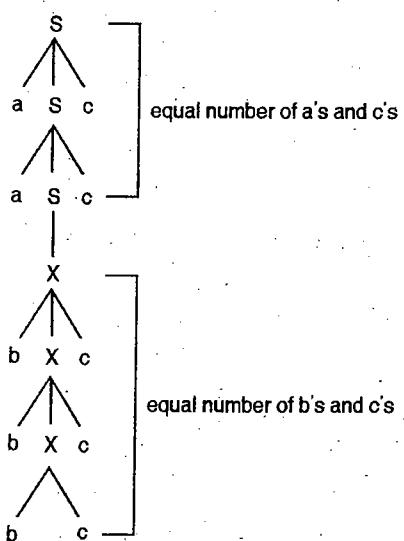


Fig. Ex. 4.2.16(a) : Parse tree for aabbccccc

**Example 4.2.17**

Find context free grammars generating each of these languages

- (1) $L_1 = \{a^i b^j c^k \mid i=j+k\}$
- (2) $L_2 = \{a^i b^j c^k \mid j=i+k\}$
- (3) $L_3 = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$

Solution :

$$(1) L_1 = \{a^i b^j c^k \mid i=j+k\}$$

Outer a's should be matched with c's then the remaining a's should be matched with b's.

$$S \rightarrow aSc \mid aXc$$

$$X \rightarrow aXb \mid ab$$

$$(2) L_2 = \{a^i b^j c^k \mid j=i+k\}$$

The string $a^i b^j c^k$ can be written as

a's and b's are equal
(generated by X)

B's and c's are equal
(generated by Y)

$$S \rightarrow XY$$

$$X \rightarrow aXb \mid ab$$

$$Y \rightarrow bYc \mid bc$$

$$(3) L_3 = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$$

$$S \rightarrow AB \quad [i=j]$$

$$S \rightarrow CD \quad [j=k]$$

A $\rightarrow aAb \mid ab$ [equal number of a's and b's followed

B $\rightarrow CB \mid c$ by some c's]

C $\rightarrow aC \mid a$ [Some a's followed by equal number of

D $\rightarrow bDC \mid bc$ b's and c's]

Example 4.2.18

Give the context free grammar for the following languages.

$$(a) (011 + 1)^* (01)^* \quad (b) 0^i 1^{i+k} 0^k \text{ where } i, k \geq 0$$

Solution :

$$(a) \text{ CFG for } (011 + 1)^* (01)^*$$

Let us assume that $(011 + 1)^*$ is generated by the variable X.

$$\therefore X \rightarrow 011X \mid 1X \mid \epsilon$$

Let us assume that $(01)^*$ is generated by the variable Y.

$$\therefore Y \rightarrow 01Y \mid \epsilon$$

Using the concatenation rule, the set of productions for $(011 + 1)^* (01)^*$ can be written as :

$$\begin{aligned} P = & \{ S \rightarrow XY \\ & X \rightarrow 011X \mid 1X \mid \epsilon \\ & Y \rightarrow 01Y \mid \epsilon \\ & \} \end{aligned}$$

Where, the set of variable V = {S, X, Y}

the set of terminals T = {0, 1}

the start symbol = S

$$(b) \text{ CFG for } 0^i 1^{i+k} 0^k \mid i, k \geq 0$$

The language L can be written as :

$$\begin{array}{c} L = \{0^i 1^i \underbrace{1^k 0^k} \\ \quad \quad \quad | \\ \quad \quad \quad X \quad \quad \quad Y \end{array}$$

Let us assume that $0^i 1^i$ is generated by the variable X.

$$\therefore X = 0X1 \mid \epsilon$$

Let us assume that $1^k 0^k$ is generated by the variable Y.

$$\therefore Y = 1Y0 \mid \epsilon$$

The set of productions P is given by :

$$\begin{aligned} P = & \{ S \rightarrow XY \\ & X \rightarrow 0X1 \mid \epsilon \\ & Y \rightarrow 1Y0 \mid \epsilon \\ & \} \end{aligned}$$

Where, the set of variable V = {S, X, Y}

the set of terminals T = {0, 1}

the start symbol = S

Example 4.2.19

Show that the language,

$$(1) L_1 = \{a^i b^j \mid i, j \geq 1\} \text{ and}$$

$$(2) L_2 = \{a^i b^j \mid i, j \geq 1\} \text{ are context free language.}$$

Solution : These languages can be shown to CFL by writing the required productions in each case.

$$\text{Productions for } L_1 = \{a^i b^j \mid i, j \geq 1\}$$

$$S \rightarrow XY$$

$$X \rightarrow aXb \mid a \quad [X \text{ generates } a^i \text{ and}]$$

$$Y \rightarrow cYb \mid c \quad [Y \text{ generates } b^j]$$



Productions for $L_2 = \{a^i b^j c^j \mid i, j \geq 1\}$

$$S \rightarrow XY$$

$$X \rightarrow aXla \quad [X \text{ generates } a^i \text{ and}]$$

$$Y \rightarrow bYc \mid bc \quad [Y \text{ generates } b^j c^j]$$

Example 4.2.20 : Give CFG for $L = \{0^i 1^j 0^k \mid j > i + k\}$

Solution :

A string of the form $0^i 1^j 0^k$ with $j > i + k$ can be written as

$$= \begin{array}{c} 0^i 1^{i+k+p} 0^k \text{ with } P > 0 \\ \boxed{0^i} \quad \boxed{1^p} \quad \boxed{1^k 0^k} \\ X \quad Y \quad Z \end{array}$$

A string $0^i 1^p 1^k 0^k$ can be represented as concatenation of three strings.

1. $0^i 1^i$ generated by X.
2. 1^p with $P > 0$, generated by Y
3. $1^k 0^k$ generated by Z.

Thus, the set of productions for the above language are given by :

$$\begin{aligned} P = \{ & S \rightarrow XYZ \\ & X \rightarrow 0X1 \mid \epsilon \\ & Y \rightarrow 1Y \mid 1 \\ & Z \rightarrow 1Z0 \mid \epsilon \\ & \} \end{aligned}$$

Where, Set of variables $V = \{X, Y, Z\}$

Set of terminals $T = \{0, 1\}$

Start symbol $= S$

Example 4.2.21

Give CFG for matching parenthesis.

Solution :

- A production of the form

$$S \rightarrow (S)$$

can generate nested parenthesis.

- A production of the form

$$S \rightarrow SS$$

will allow parenthesis to grow side ways.

Nesting : ((0))

Side ways : (()) (())



Thus, the set of productions for the above language are given by :

$$P = \{S \rightarrow (S) \mid SS \mid \epsilon\}$$

Where, Set of variables $V = \{S\}$

Set of terminals $T = \{(,)\}$

Start symbol $= S$

Example 4.2.22 SPPU - Aug. 15, 3 Marks

Write a context free grammar for following Language :

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

Solution :

CFG for $(a+b)b(a+b)^*b(a+b)$:

$$S \rightarrow A_1 b A_2 b A_1$$

$$A_1 \rightarrow a \mid b \quad A_2 \rightarrow a A_2 \mid b A_2 \mid \epsilon$$

Example 4.2.23 SPPU - Dec. 13, 3 Marks

Give CFG for all strings with at least two 0's, $\Sigma = \{0, 1\}$

Solution : The regular expression for the above language is given by :

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

Let us assume that 1^* is generated by the variable X.

$$\therefore X \rightarrow 1X \mid \epsilon$$

Let us assume that $(0+1)^*$ is generated by the variable Y.

$$\therefore Y \rightarrow 0Y \mid 1Y \mid \epsilon$$

Using the concatenation rule, the set of productions for $1^* 0 1^* 0 (0+1)^*$ can be written as :

$$\begin{aligned} P = \{ & S \rightarrow X0X0Y \\ & X \rightarrow 1X \mid \epsilon \\ & Y \rightarrow 0Y \mid 1Y \mid \epsilon \\ & \} \end{aligned}$$

where, the set of variables $V = \{S, X, Y\}$

the set of terminals $T = \{0, 1\}$

the start symbol $= S$.

Example 4.2.24

Give CFG for set of odd length strings in $\{0, 1\}^*$ with middle symbol '1'.

Solution : A string generated by the above language could be shown as given below :

String of length n from $\{0, 1\}^*$

1 String of length n from $\{0, 1\}^*$

The hidden recursion can be understood from the fact that deletion of first and last character will not change the basic nature of the string.



i.e. If S is a string of the given form then S are :
 $0S0, 0S1, 1S0, 1S1$.

Thus, the set of productions for the above language is given by :

$$P = \{$$

$$S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 1 \\ \}$$

The production $S \rightarrow 1$, terminates the recursion with 1 as a middle character.

where, the set of variables $V = \{S\}$

the set of terminals $T = \{0, 1\}$

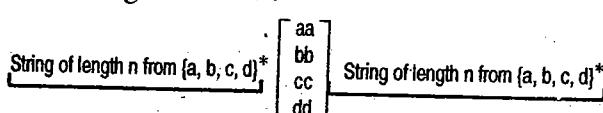
the start symbol = S

Example 4.2.25

Give CFG for set of even length strings in $\{a, b, c, d\}^*$ with two middle symbol equal.

Solution :

A string generated by the above language could be shown as given below :



The hidden recursion can be understood from the fact that deletion of first and last character will not change the basic nature of the string.

i.e., if S is a string of the given form then S are :

$aSa, aSb, aSc, aSd, bSa, bSb, bSc, bSd, cSa, cSb, cSc, cSd, dSa, dSb, dSc, dSd$.

Thus, the set of productions for the above language is given by :

$$P = \{S \rightarrow aSa \mid aSb \mid aSc \mid aSd \mid \\ bSa \mid bSb \mid bSc \mid bSd \mid \\ cSa \mid cSb \mid cSc \mid cSd \mid \\ dSa \mid dSb \mid dSc \mid dSd \mid \\ aa \mid bb \mid cc \mid dd \\ \}$$

where, set of variables $V = \{S\}$

set of terminals $T = \{a, b, c, d\}$

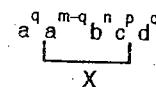
start symbol = S

Example 4.2.26

Give CFG for $L = \{a^m b^n c^p d^q \mid m + n = p + q\}$

Solution : There could be three possibilities on number of a's and number of d's.

Case I : $m > q, a^m b^n c^p d^q$ can be written as



where $a^m - a^q b^n c^p$ is generated by the variable X (say).

$$a^m b^n c^p = a^q b^n c^p$$

The string X is of the form

Productions for X can be written as given below :

$$X \rightarrow aXc \mid Z$$

$$Z \rightarrow bZc \mid \epsilon$$

Case II : $m < q, a^m b^n c^p d^q$ can be written as

$$a^m b^n c^p d^q = a^m b^{n-m} c^p d^q$$

where $b^{n-m} c^p d^{q-m}$ is generated by the variable Y (say).

$$\text{The string } Y \text{ is of the form } b^{n-m} c^p d^q = b^{n-m} c^p d^q$$

Productions for Y can be written as given below :

$$Y \rightarrow bYd \mid Z$$

$$Z \rightarrow bZc \mid \epsilon$$

Case III : $m = q, a^m b^n c^p d^q$ can be written as

$$a^m b^n c^p d^q$$

Z(say)

where $b^n c^p$ is generated by Z (say)

$$Z = bZc \mid \epsilon$$

Thus the set of productions for the above language are given by :

$$P = \{S \rightarrow aSd \mid X \mid Y$$

$$X \rightarrow aXc \mid Z$$

$$Y \rightarrow bYd \mid Z$$

$$Z \rightarrow bZc \mid \epsilon$$

}

where, Set of variables $V = \{S, X, Y, Z\}$

Set of terminals $T = \{a, b, c, d\}$

Start symbol = S

Example 4.2.27

Give CFG for $L = \{a^m b^n c^p \mid m, n >= 1\}$

Solution : The production $S \rightarrow aSc$ can generate equal number of a's and b's.

Thus the set of productions for the given language is :



$$\begin{aligned} P = \{ & S \rightarrow aSc \mid X \\ & X \rightarrow bX \mid b \\ & \} \end{aligned}$$

The production $S \rightarrow X$, terminates the recursion.
The variable X can generate one or more b's.

Where, Set of variables $V = \{S, X\}$

Set of terminals $T = \{a, b\}$
Start symbol = S

Example 4.2.28

Give CFG for $L = \{x \mid x \text{ contains equal number of } a's \text{ and } b's\}$.

Solution : Let us assume that there are three variables S, A and B.

where,

S generates a string with equal number of a's and b's.

A generates a string in which number of a's = 1 + number of b's.

B generates a string in which number of b's = 1 + number of a's.

We can write the set of production using indirect recursion by relating the three variables S, A and B.

- The relation between S and B is given by $S \rightarrow aB$, B represents a string in which number of b's is one more than a's. If we prepend an 'a' to 'B', both a's and b's will become equal.
- Similarly, the relation between S and A is given by

$$S \rightarrow bA$$

- A is related to S by

$A \rightarrow aS$, removing an 'a' from a string represented by A will render both a's and b's equal.

- A is related to A by

$A \rightarrow bAA$, one b and two A's on the right hand side will mean number of a's one more than number of b's.

- Similarly,

$$B \rightarrow bS$$

$$\text{and } B \rightarrow aBB$$

Thus the set of productions for the above language are given by :

$$\begin{aligned} P = \{ & S \rightarrow aB \mid bA \\ & A \rightarrow aS \mid bAA \mid a \\ & B \rightarrow bS \mid aBB \mid b \\ & \} \end{aligned}$$

where, set of variables $V = \{S, A, B\}$

set of terminals $T = \{a, b\}$
start symbol = S

The behaviour of productions can be understood with the help of the following example.

Example : Generation of the string aababb

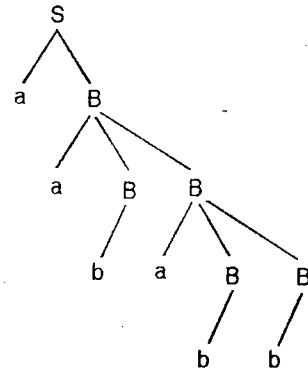


Fig. Ex. 4.2.28 : Parse tree for aababb

Example 4.2.29

Give CFG for strings in ab^*

Solution : Let us say that b^* is generated by variable B.

$$\therefore B \rightarrow bB \mid \epsilon$$

The set of productions for the above language can be written using the rule of concatenations.

$$\begin{aligned} P = \{ & S \rightarrow aB \\ & B \rightarrow bB \mid \epsilon \\ & \} \end{aligned}$$

where, Set of variables $V = \{S, B\}$

Set of terminals $T = \{a, b\}$
Start symbol = S

Example 4.2.30

Give CFG for strings in a^*b^* .

Solution :

Let us assume that a^* and b^* are generated by variable A and B respectively.

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

The set of productions for the above language can be written using the rule of concatenation.

$$\begin{aligned} P = \{ & S \rightarrow AB \\ & A \rightarrow aA \mid \epsilon \\ & B \rightarrow bB \mid \epsilon \\ & \} \end{aligned}$$



Where, Set of variables $V = \{S, A, B\}$

Set of terminals $T = \{a, b\}$

Start symbol = S

Example 4.2.31

Give CFG for $L = \{wcw^T \mid w \in \{a, b\}^*\}$

Solution : L is a language of odd length palindromes with middle character as c .

The set of productions for the above language is given by :

$$S \rightarrow aSa \mid bSb \mid c$$

Where, Set of variables $V = \{S\}$

Set of terminals $T = \{a, b, c\}$

Start symbol = S .

Example 4.2.32 SPPU - May 15, 2 Marks

Give CFG for $L = \{a^n b^m \mid n \neq m\}$

Solution :

A string belonging to L can have either of the two forms given below :

1. $a^j b^j$ where $j > 0$

or 2. $a^j b^j$ where $j > 0$

Let us assume that a^j is generated by A and b^j is generated by B .

The set of productions for the above language is given by :

$$P = \{S \rightarrow aSb \mid A \mid B$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

}

Where, Set of variables $V = \{S, A, B\}$

Set of terminals $T = \{a, b\}$

Start symbol = S

Example 4.2.33

Give CFG for the language

$$L = \{a^n b^m c^k \mid n = m \text{ or } m \leq k\}$$

Solution :

The language L is union of two languages L_1 and L_2 .

$$\text{Where, } L_1 = \{a^n b^m c^k \mid n = m\} = \{a^n b^n c^k\}$$

$$L_2 = \{a^n b^m c^k \mid m \leq k\}$$

$$= \{a^n b^m c^{m+p} \mid p \geq 0\}$$

$$= \{a^n b^n c^m c^p \mid p \geq 0\}$$

The productions for languages $L_1 = \{a^n b^n c^k\}$ are

given as :

$S_1 = AB$, where A generates $a^n b^n$ and B generates c^k .

$$A = aAb \mid \epsilon$$

$$B = cB \mid \epsilon$$

The productions for language

$$\{L_2 = a^n b^m c^m c^p \mid p \geq 0\}$$
 are given as :

$S_2 = XYB$, where X generates a^n and Y generates $b^m c^m$.

$$X = aX \mid \epsilon$$

$$Y = bYc \mid \epsilon$$

Now, we can write productions for the given language L using the rule of union.

$$L = L_1 \cup L_2$$

$$P = \{$$

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow AB$$

$$A \rightarrow aAb \mid \epsilon$$

$$B \rightarrow cB \mid \epsilon$$

$$S_2 \rightarrow XYB$$

$$X \rightarrow aX \mid \epsilon$$

$$Y \rightarrow bYc \mid \epsilon$$

}

where, The set of variables $V = \{S, S_1, S_2, A, B, X, Y\}$

The set of terminals $T = \{a, b\}$

The start symbol = S .

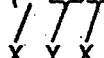
Example 4.2.34

Give CFG for the language $L = \{0^m 1^n 0^{m+n} \mid m, n \geq 0\}$

Solution :

The language L can be written as :

$$L = \{0^m 1^n 0^{m+n}\} = \{0^m 1^n 0^m 0^n\}$$



Let us assume that 0^m is generated by a variable X and $1^n 0^n$ is generated by a variable Y .

The set of productions for the given language is given by :

$$P = \{ \quad S \rightarrow XYX$$

$$X \rightarrow 0X \mid \epsilon$$

$$Y \rightarrow 1Y0 \mid \epsilon$$

}

Where, the set of variables $V = \{S, X, Y\}$

The set of terminals $T = \{0, 1\}$

The start symbol $= S$

Example 4.2.35

Give CFG for the language $L = \{0^a 1^b 2^c \mid a - c = b\}$

Solution :

The string $0^a 1^b 2^c$ with $a - c = b$

Can be written as $0^{c+b} 1^b 2^c$

$$= 0^c 0^b 1^b 2^c$$

[]

The set of productions for the given language is :

$$\begin{aligned} P = & \{ S \rightarrow 0S2 \mid X \text{ [X generates a} \\ & \text{string of the form } 0^b 1^b] \\ & X \rightarrow 0X1 \mid \epsilon \\ & \} \end{aligned}$$

where, The set of variables $V = \{S, X\}$

The set of terminals $T = \{0, 1\}$

Start symbol $= S$

Example 4.2.36

Given the context free grammar for the following languages :

$$(a) L = \{a^n b^{2n} \mid n > 1\}$$

$$(b) L = \{a^m b^n \mid n > m\}$$

Solution :

$$(a) \quad L = \{a^n b^{2n} \mid n > 1\}$$

$$S \rightarrow aSbb \mid aabb$$

$$(b) \quad L = \{a^m b^n \mid n > m\}$$

$$S \rightarrow aSb \mid B$$

$$B \rightarrow bB \mid b$$

Example 4.2.37

Give CFG for the following language : The set of odd length strings in $\{a, b\}^*$ whose first, middle and last symbols are all same.

Solution : A string belonging to the language will have any of the two forms, given below.

Form 1 :

$$a \text{ a string of length } n \text{ over } \{a, b\}^* \cdot a \text{ a string of length } n \text{ over } \{a, b\}^* a$$

[] []

Generated by X

Form 2 :

$$b \text{ a string of length } n \text{ over } \{a, b\}^* \cdot b \text{ a string of length } n \text{ over } \{a, b\}^* b$$

[] []

Generated by Y

The set of productions for the given language is :

$$\begin{aligned} P = & \{ S \rightarrow aXa \mid bYb \\ & X \rightarrow aXa \mid aXb \mid bXa \mid bXb \mid a \\ & Y \rightarrow aYa \mid aYb \mid bYa \mid bYb \mid b \\ & \} \end{aligned}$$

Where, The set of variables $V = \{S, X, Y\}$

The set of terminals $T = \{a, b\}$

The start symbol $= S$

Example 4.2.38

Write a CFG for the following languages

$$L = \{u\omega b \mid u, \omega \text{ are in } \{a, b\}^* \text{ and } |u| = |\omega|\}$$

Solution : A string generated by the above language could be shown as given below :

$$\text{String of length } n \text{ from } \{a, b\}^* \xrightarrow{a} \text{String of length } n \text{ from } \{a, b\}^* \xrightarrow{b}$$

[] []

Generated by X

The hidden recursion in X can be understood from the fact that deletion of first and last character will not change the basic nature of X.

$\therefore X$ is a string of the given form then X are :

$$X \rightarrow aXa \mid aXb \mid bXa \mid bXb$$

Thus, the set of productions for the above language is given by,

$$\begin{aligned} P = & \{ S \rightarrow \dots Xb \\ & X \rightarrow aXa \mid aXb \mid bXa \mid bXb \mid a \\ & \} \end{aligned}$$

The production $X \rightarrow a$, terminates the recursion with a as the middle character of X.

The set of variables $V = \{S, X\}$

The set of terminals $T = \{a, b\}$

The start symbol $= S$

Example 4.2.39

Write a CFG which generates the language L defined by the regular expression : $(a + b)^*bbb(a + b)^*$

Solution : Let us assume that the start symbol is S.

$\therefore S \rightarrow XbbbX$, where X generates
a string in $(a + b)^*$



$$X \rightarrow aX \mid bX \mid \epsilon$$

The CFG, $G = (V, T, P, S)$

Where, $V = (S, X)$

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

$$P = \left\{ \begin{array}{l} S \rightarrow XbbbX \\ X \rightarrow aX \mid bX \mid \epsilon \end{array} \right\}$$

S = Start symbol

Example 4.2.40

Write CFG for the language $\Sigma = \{a, b\}$ number of a's is a multiple of 3.

Solution : We can design a DFA for strings in which number of a's is multiple of 3.

The set of productions can be written from the DFA :

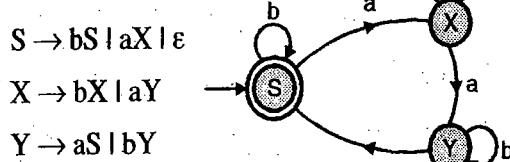


Fig. Ex. 4.2.40

Example 4.2.41 SPPU - May 15, 2 Marks

Give the CFG for $\Sigma = \{a, b\}$.

To generate strings in which no consecutive b's can occur but a's can be consecutive.

Solution : It is a regular language and hence the grammar can be written after drawing DFA for the same.

Step 1 : DFA for the given language :

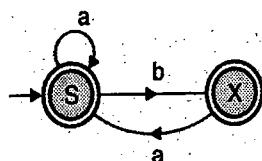


Fig. Ex. 4.2.41

Step 2 : Writing the set of productions from the above finite automata.

$$S \rightarrow as \mid bx \mid \epsilon$$

$$X \rightarrow as$$

Example 4.2.42

Consider the following grammar :

$$S \rightarrow aSa \mid bSb \mid aa \mid bb$$

Show that

1. $L(G)$ has no strings of odd length.
2. Any string in $L(G)$ is of length $2n$, $n > 0$
3. The number of strings of length $2n$ is 2^n .

Solution :

1. Each production from $(S \rightarrow aSa \mid bSb \mid aa \mid bb)$ has two terminals on the right side. Terminals are always generated in pair. Therefore, $L(G)$ has no strings of odd length.

2. Smallest string generated by S is either aa or bb with length 2. The language of the given grammar is a set of even length palindromes. Therefore, any string in $L(G)$ is of length $2n$, $n > 0$.

3. Let us take a string of length $2n$ (say S_{2n})

$$S_{2n} = aS_{2n-2} a \mid bS_{2n-2} b$$

Thus, we can write a recurrence relation for S_{2n}

$$S_{2n} = 2 \times S_{2n-2} = 2^2 S_{2n-4} = 2^{n-1} \times S_2 = 2^n$$

Example 4.2.43

Construct a context free grammar G generating all integers (with sign). Derive an example integer.

Solution : The set of production P is given by

$$P = \{$$

$$S \rightarrow +II -III$$

$$I \rightarrow 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \\ \mid 7X \mid 8X \mid 9X$$

$$X \rightarrow 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \\ \mid 7X \mid 8X \mid 9X \mid \epsilon$$

}

The grammar G is given by

$$G = (\{S, I, X\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, P, S)$$

Example considered for derivation is the string : +95.

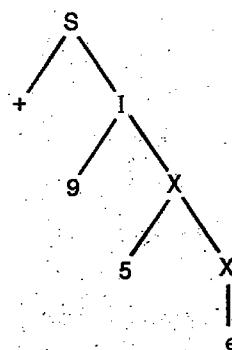


Fig. Ex. 4.2.43 : Parse tree

Syllabus Topic : Ambiguous CFG, Removal of Ambiguity

4.3 Ambiguous Grammar

SPPU - Dec. 14, Dec. 15

University Question

Q. Define ambiguous grammar.

(Dec. 2014, Dec. 2015, 2 Marks)



A grammar is said to be ambiguous if the language generated by the grammar contains some string that has two different parse trees.

Example : Let us consider the grammar given below :

$$E \rightarrow E + E \mid a \mid b \quad \dots (\text{Grammar 4.8})$$

A string $a + b + a$ is generated by the given grammar.

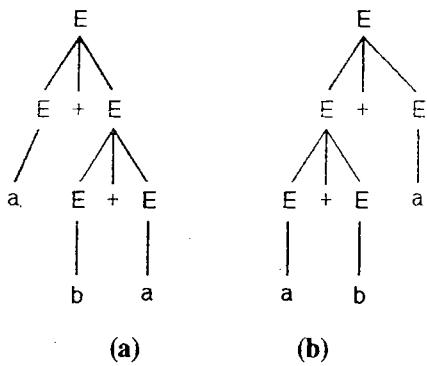


Fig. 4.3.1 : Parse trees considered for ambiguity

- The grammar (4.8), generates $a + b + a$ in two different ways. The two derivations are shown in Fig. 4.3.1(a) and 4.3.1(b).
- The first derivation (Fig. 4.3.1(a)) says that $(b + a)$ is evaluated first and then the evaluated value is added to a . Thus the right side $+$ operator gets a precedence over the left side $+$ operator. The expression $a + b + a$ is treated as $(a + (b + a))$.
- The second derivation (Fig. 4.3.1(b)) says that $a + b$ is evaluated first and then the evaluated value is added to a . Thus the left side $+$ operator will get precedence over right side $+$ operator. The expression $a + b + a$ is treated as $((a + b) + a)$.

Removing ambiguity : There is no general rule for removing ambiguity from CFG. Removing ambiguity from a grammar involves rewriting of grammar so that there is only one derivation tree for every string belonging to $L(G)$ i.e., language generated by grammar G .

Ambiguity from the grammar

$$E \rightarrow E + E \mid a \mid b$$

can be removed by strictly assigning higher precedence to left side $+$ operator over right side $+$ operator. This will mean evaluation of an expression of the form $a + b + a$, from left to right. This property can be incorporated in the grammar itself by suitably modifying the grammar.

- Parse tree of Fig. 4.3.1(b) is based on left to right evaluation.

- Left to right evaluation in grammar(4.8) can be enforced by introducing one more variable T . Variable T can not be broken by $+$ operator.

- An unambiguous grammar for the grammar of (4.8) is given in (4.9)

$$E \rightarrow E + T \mid T$$

$$T \rightarrow a \mid b \quad \dots (\text{Grammar 4.9})$$

- A production of the form $E \rightarrow E + T$ provides a binding. $E + T$, implies that E must be evaluated first before an atomic T can be added to it. E can be broken down in $E + T$ but T cannot be broken further. This ensures higher precedence to left side $+$ operator over right side $+$ operator.
- Parse tree in Fig. 4.3.2 is based on unambiguous grammar (4.9), for the string $a + b + a$.

In general,

1. Sometimes we can remove ambiguity from a grammar by hand.
2. There is no algorithm for either deletion of ambiguity or for removing ambiguity.
3. Some context free languages are inherently ambiguous. They have only ambiguous CFG.

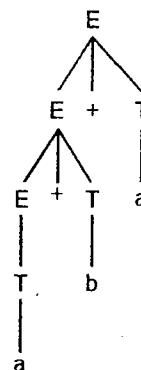


Fig. 4.3.2 : A parse tree for $a + b + a$ using an unambiguous grammar

An example of inherently ambiguous language is

$$L = L_1 \cup L_2$$

$$\text{Where, } L_1 = \{a^n b^m c^m d^n \mid n, m > 0\}$$

$$L_2 = \{a^n b^n c^m d^m \mid n, m > 0\}$$

Example 4.3.1

Consider the grammar

$$E \rightarrow E + E \mid E * E \mid (E) \mid I$$

$$I \rightarrow a \mid b$$

- Show that the grammar is ambiguous.
- Remove ambiguity.

**Solution :**

There are two problems in the given grammar.

1. Both + operator and * operator are at the same precedence level.
2. An expression can be evaluated either left to right or right to left.

These problems can be removed by introducing more variables, each representing expressions that share a level of "binding strength."

- A variable T is an expression that cannot be broken by + operator. Let us call it a **term**. A term can be expressed as product of one or more factors.
- A variable F is a factor that can not be broken by + operator or * operator. A factor is
 - (a) Identifiers
 - (b) Literals
 - (c) Parenthesized expressions

An unambiguous grammar is given below :

$$\begin{array}{l} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid a \mid b \end{array} \quad \text{(Grammar 4.10)}$$

A parse tree for $a + b * a$ using both ambiguous and unambiguous grammar is shown in Fig. Ex. 4.3.1.

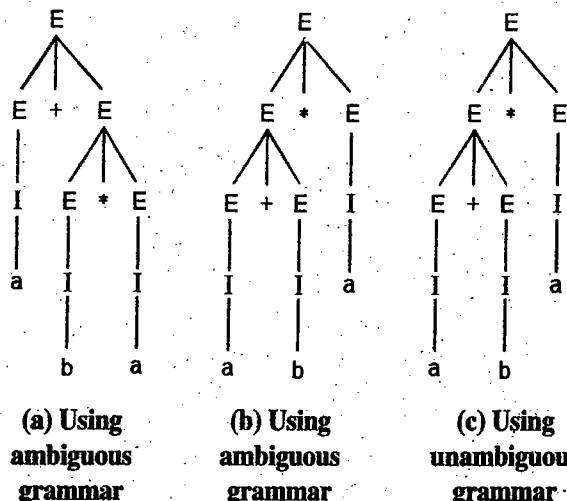


Fig. Ex. 4.3.1 : Parse trees for $a + b * a$

Fig. Ex. 4.3.1(a) and Ex. 4.3.1(b) are using the ambiguous grammar.

Fig. Ex. 4.3.1(c) is using an unambiguous grammar.

The grammar given in the example is an ambiguous grammar as there are two different parse trees for the expression $a + b * a$ (shown in Fig. Ex. 4.3.1(a) and 4.3.1(b)).

Example 4.3.2

Give an ambiguous grammar for if-then-else statement and then re-write an equivalent unambiguous grammar.

Solution :

The dangling else is a well known problem in programming language. If statement in C-language can be written in any of the two forms :

Form A : if ($x > y$)
 $a = a + 1;$
 Form B : if ($x > y$)
 $a = a + 1$
 else
 $a = a - 1;$

We can substitute the condition $x > y$ with C and S for $a = a + 1$. With these substitutions, if statement in two forms can be written as :

if C then S
 if C then S_1 else S_2

For the case of understanding, a reserve word 'then' has been introduced.

A nested if statement can be written with two interpretations. It is shown in Fig. Ex. 4.3.2.

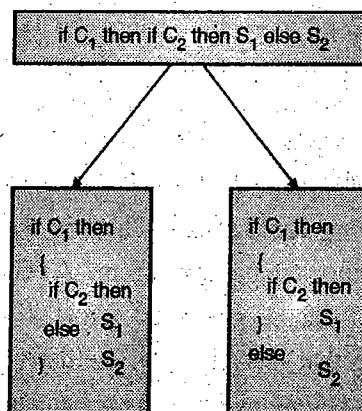


Fig. Ex. 4.3.2 : Two interpretations of a nested if-statement

- A nested if statement can have two interpretations.

Example 4.3.3

An ambiguous grammar for if statement may have dangling else problem.

Solution :

An unambiguous grammar should attach the dangling else with inner if-statement.



An ambiguous grammar for if-statement is given below :

$$\begin{aligned} S &\rightarrow \text{if } C \text{ then } S \\ &\quad | \text{ if } C \text{ then } S \text{ else } S \\ &\quad | a \text{ [termination of a statement by 'a']} \\ C &\rightarrow c_1 \mid c_2 \end{aligned}$$

The above grammar can be written in short as :

$$\begin{array}{l} S \rightarrow iCtS \mid CtSeS \mid a \\ C \rightarrow c_1 \mid c_2 \end{array} \quad \boxed{\quad} \quad (\text{Grammar 4.11})$$

Where, i stands for if,
 t stands for then,
 and e stands for else

Example : Grammar (4.11) can be shown to be ambiguous by drawing two different derivation trees for the string c_1tic_2taea . Two different derivations are shown in the Fig. Ex. 4.3.2(a) and (b).

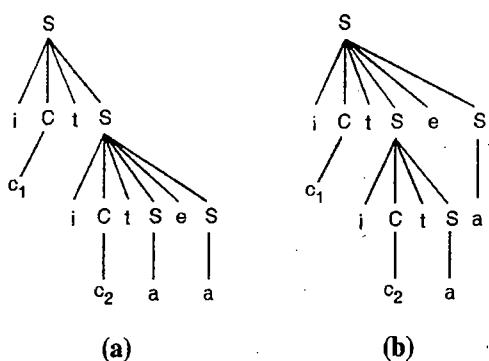


Fig. Ex. 4.3.3 : Two derivations for c_1tic_2taea

Removing ambiguity : The ambiguity from the grammar (4.11) can be removed by binding dangling else with inner if – statement. We will introduce two more variables.

$U \rightarrow$ For if-statement without else part.

$M \rightarrow$ For if-statement with else part.

An unambiguous grammar is given below :

$$\begin{aligned} S &\rightarrow U \mid M \\ M &\rightarrow iCtMeM \mid a \\ U &\rightarrow iCtS \mid iCtMeU \\ C &\rightarrow c_1 \mid c_2 \end{aligned} \quad \boxed{\quad} \quad (\text{Grammar 4.12})$$

The working of grammar (4.12) can be understood with the help of derivation tree for c_1tic_2taea . It is shown in Fig. Ex. 4.3.3(c).

- Outer if-statement is without else part.
- Inner if-statement is with else part.

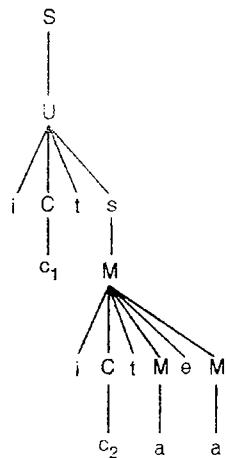


Fig. Ex. 4.3.3(c) : Derivation tree for c_1tic_2taea

Example 4.3.4

Show that the CFG given below. Which generates all strings of balanced parentheses is ambiguous. Give an equivalent unambiguous grammar.

$$S \rightarrow SS \mid (S) \mid \epsilon$$

Solution :

Let us consider derivation of the string $(())()$. Two derivations are shown in Fig. Ex. 4.3.4(a) and Fig. Ex. 4.3.4(b).

Since, there are two different derivation for the string $(())()$ the grammar is ambiguous.

An unambiguous grammar is given by

$$S \rightarrow ST \mid T$$

$$T \rightarrow (S) \mid ()$$

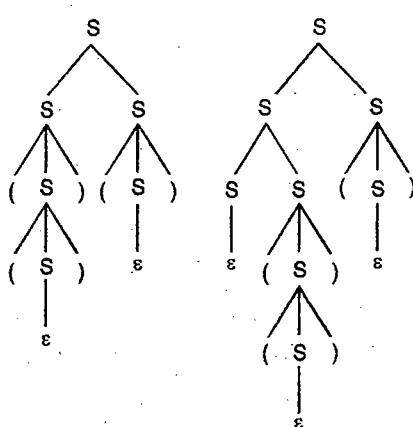


Fig. Ex. 4.3.4 : Two different derivations for $(())()$

- Sideways parentheses can be generated by $S \rightarrow ST$, where T is atomic as far as sideways generation is concerned. It forces left to right generation of balanced parenthesis. Nesting of parenthesis is generated by $T \rightarrow (S)$.

Example 4.3.5

Write an unambiguous CFG for arithmetic expressions with operators : +, *, /, ^, unary minus and operand a, b, c, d, e, f. Also, if should be possible to generate brackets with your grammar.

Derive $(a + b)^d / e + (-f)$ from your grammar.

Solution : An unambiguous grammar is given below.

$$E \rightarrow E + T \mid T$$

[+ has lowest priority with L \rightarrow R associativity]

$$T \rightarrow T * F \mid T / F \mid F$$

[* and / has higher priority over + with L \rightarrow R associativity]

$$F \rightarrow F^G \mid G$$

[^ has higher priority over * and / with L \rightarrow R associativity]

$$G \rightarrow - H \mid H$$

[unary - has the highest priority]

$$H \rightarrow a \mid b \mid c \mid d \mid e \mid f \mid (E)$$

[to handle brackets and identifiers]

Derivation tree for $(a + b)^d / e + (-f)$

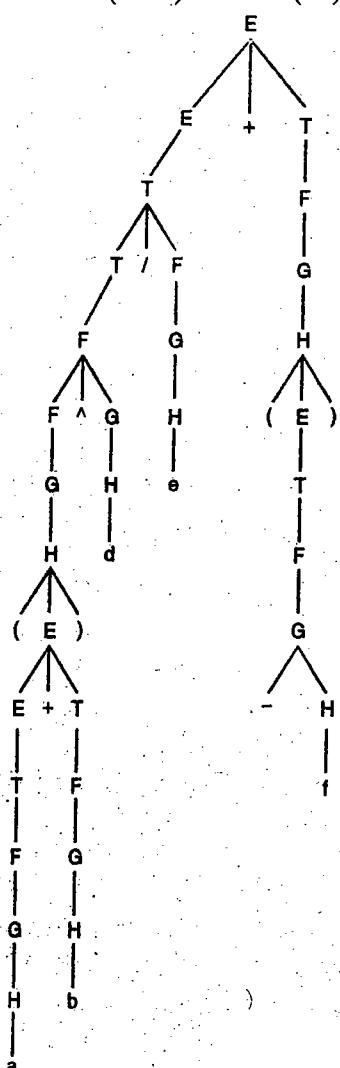


Fig. Ex. 4.3.5 : Derivation tree for $(a + b)^d / e + (-f)$

Example 4.3.6

Is the following CFG ambiguous ?

$$S \rightarrow aB \mid ab \quad A \rightarrow aAB \mid a \quad B \rightarrow ABb \mid b$$

If so, show multiple derivation trees for the same string.

Solution : The given CFG is ambiguous as we can have two different derivation trees for the string ab. Derivation trees are shown in Fig. Ex. 4.3.6.

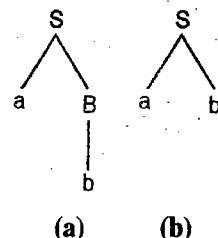


Fig. Ex. 4.3.6 : Two derivations for ab

Example 4.3.7

Is the following CFG ambiguous ?

$$G = (\{S, A\}, \{a, b\}, P, S)$$

where P consists of

$$S \rightarrow aAS \mid a$$

$$A \rightarrow SbA \mid SS \mid ba$$

Solution : The above grammar can be proved to be an ambiguous grammar by deriving a string a^7 in two different ways. Derivation of a^7 is shown in Fig. Ex. 4.3.7.

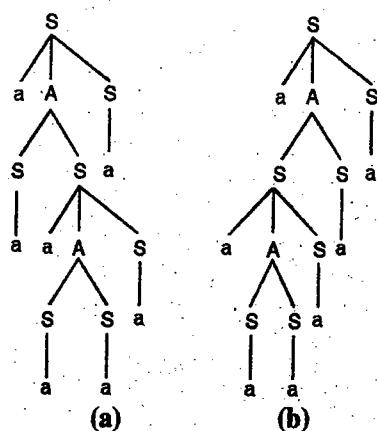


Fig. Ex. 4.3.7 : Two derivations for a^7

Example 4.3.8

Consider the grammar having productions :

$$S \rightarrow aS \mid \epsilon$$

$$S \rightarrow aSbS$$

The grammar is ambiguous.

- Show in particular that the string aab has two parse trees.

- Find an unambiguous grammar for the same.

Solution :

- Two parse trees for aab are given in Fig. Ex. 4.3.8.

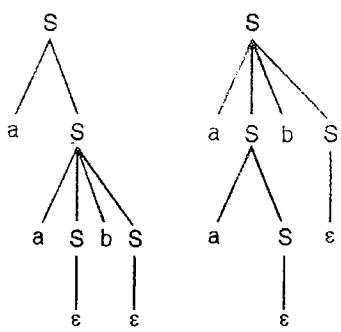


Fig. Ex. 4.3.8 : Two derivations for aab

- (ii) The set of productions resemble productions for if-statement. Thus the ambiguity can be removed in a similar way through introduction of two variables U and M with resulting productions is :

$$\begin{aligned} S &\rightarrow U \mid M \\ U &\rightarrow aS \mid aMbU \\ M &\rightarrow aMbM \mid \epsilon \end{aligned}$$

Example 4.3.9 SPPU - May 12, 8 Marks

Let G be the grammar

$$S \rightarrow aB \mid bA \quad A \rightarrow a \mid aS \mid bAA \quad B \rightarrow b \mid bS \mid aBB$$

For string aaabbabbba find

- (i) Left most derivation (ii) Right most derivation
 (iii) Parse Tree (iv) Is the grammar unambiguous ?

Solution : It will be worth while to draw the parse tree and from the parse tree, one can easily write left most and right most derivation.

(iii) Parse tree

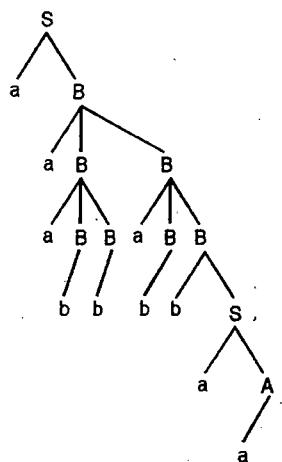


Fig. Ex. 4.3.9

(i) Left most derivation

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaaBBB \rightarrow aaabBB \\ &\rightarrow aaabbB \rightarrow aaabbaBB \rightarrow aaabbabB \rightarrow aaabbabbS \\ &\rightarrow aaabbabbA \rightarrow aaabbabbba \end{aligned}$$

(ii) Right most derivation

$$\begin{aligned} S &\rightarrow aB \rightarrow aaBB \rightarrow aaBaBB \rightarrow aaBaBbS \rightarrow aaBaBbbA \\ &\rightarrow aabA \\ &\rightarrow aabbbA \rightarrow aaaBBabbba \rightarrow aaaBbabbba \\ &\rightarrow aaabbabbba \end{aligned}$$

- (iv) The grammar is ambiguous as we can draw two parse trees for aababb

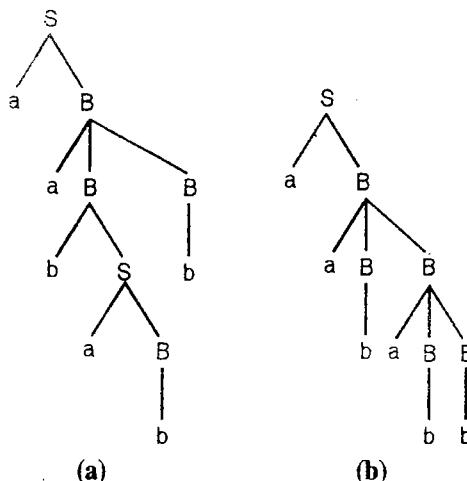


Fig. Ex. 4.3.9

Example 4.3.10

In each case, show that the grammar is ambiguous, and find the equivalent unambiguous grammar

- (a) $S \rightarrow SS \mid a \mid b$
 (b) $S \rightarrow ABA, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid \epsilon$
 (c) $S \rightarrow aS \mid baa \mid Sb \mid \epsilon$

Solution :

- (a) $S \rightarrow SS \mid a \mid b$ is ambiguous

Two parse trees can be generated for the string aab.

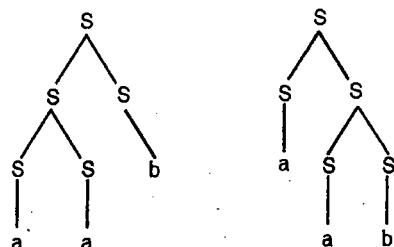


Fig. Ex. 4.3.10(a)

Unambiguous grammar : $S \rightarrow aS \mid bS \mid aalab \mid balbb$

- (b) $S \rightarrow ABA, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid \epsilon$

Two parse trees can be generated for the string aa

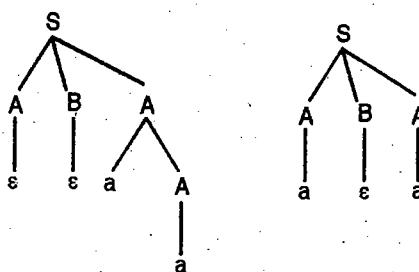


Fig. Ex. 4.3.10(b)

Unambiguous grammar :

$$S \rightarrow aslbX\epsilon$$

[String of only a's will be generated by $S \rightarrow aS \mid \epsilon$]



$$X \rightarrow bXly$$

$$Y \rightarrow aYl\epsilon$$

(c) $S \rightarrow aSblaSb\epsilon$ is ambiguous

We can draw two production trees for aaabb

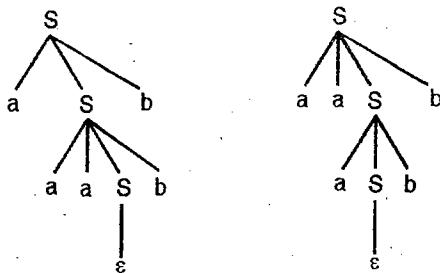


Fig. Ex. 4.3.10(c)

Unambiguous grammar : Given language is of the form

$$L = \{a^i b^j \mid i \geq j \text{ and } i, j \geq 0\}$$

$$S \rightarrow aSb \mid X$$

$$X \rightarrow aX \mid \epsilon$$

Example 4.3.11.

Consider the grammar

$$G = \{V = \{E, F\}, T = \{a, b, -\}, E, P\}$$

where P consists of rules :

$$E \rightarrow F - E, F \rightarrow a, E \rightarrow E - F, F \rightarrow b, E \rightarrow F$$

(a) Show that G is ambiguous

(b) Remove the ambiguity.

Solution : The production rules can be re-written as given below :

$$E \rightarrow F - E \mid E - F \mid F$$

$$F \rightarrow a \mid b$$

(a) The grammar can be shown to be an ambiguous grammar by drawing two different derivation trees for the string a - b.

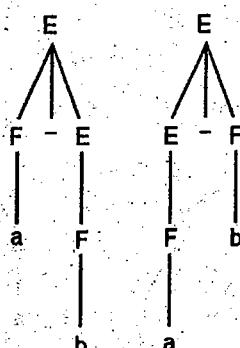


Fig. Ex. 4.3.11 : Two derivations for a - b

(b) The production $E \rightarrow F - E$, makes the evaluation from right to left which is normally not allowed.

Thus the ambiguity can be removed by deleting the production $E \rightarrow F - E$. The resulting unambiguous grammar is :

$$E \rightarrow E - F \mid F$$

$$F \rightarrow a \mid b$$

Example 4.3.12 [SPPU - Dec. 14, 4 Marks]

Test whether the following grammars are ambiguous.

$$1. S \rightarrow 0S1S \mid 1S0S \mid \epsilon$$

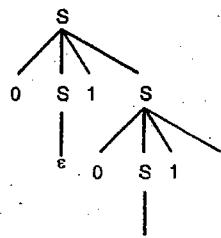
$$2. S \rightarrow AA$$

$$A \rightarrow aAb \mid bAa \mid \epsilon$$

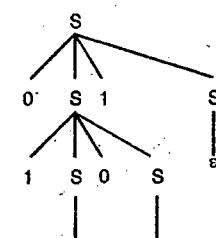
Solution :

$$1. S \rightarrow 0S1S \mid 1S0S \mid \epsilon$$

The given grammar is ambiguous as we can have two different parse trees for the string 0101.



(a)



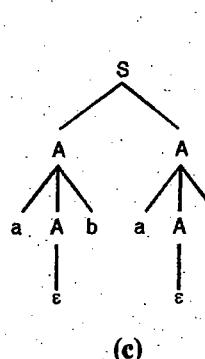
(b)

Fig. Ex. 4.3.12

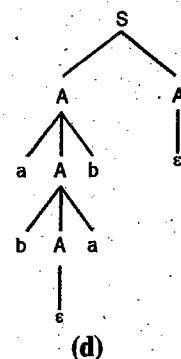
$$2. S \rightarrow AA$$

$$A \rightarrow aAb \mid bAa \mid \epsilon$$

The given grammar is ambiguous. We can derive two different parse trees for the string abab.



(c)



(d)

Fig. Ex. 4.3.12

Example 4.3.13

If G is the grammar $S \rightarrow SbSla$, show that G is ambiguous.

Solution :

The string ababa has two derivations. This shows that the grammar is ambiguous. The two derivations for the string ababa are given below.

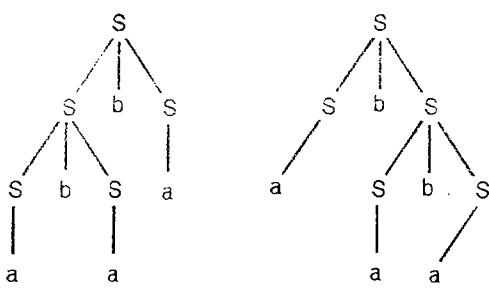


Fig. Ex. 4.3.13

Example 4.3.14

Obtain the unambiguous grammar for the following grammar

$$E \rightarrow E + E \mid E - E$$

$$E \rightarrow E * E \mid E / E$$

$$E \rightarrow (E) \mid I$$

$$I \rightarrow a \mid b \mid c$$

Solution :

There are two problems in the given grammar :

1. All the operators are at the same precedence level.
2. An expression is both left and right associative.

These problems can be removed by introducing more variables, each representing expressions that share a level of "binding strength".

An unambiguous grammar is given below :

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F$$

$$F \rightarrow (E) \mid I$$

$$I \rightarrow a \mid b \mid c$$

Example 4.3.15 SPPU - Dec. 12, 6 Marks

Show that the grammar

$$S \rightarrow aB \mid ab$$

$$A \rightarrow aAB \mid a$$

$$B \rightarrow AbB \mid b$$

Is ambiguous for the string aaabbb.

Solution : We can draw more than 1 derivation trees for the string aaabbb. These derivation trees are given below.

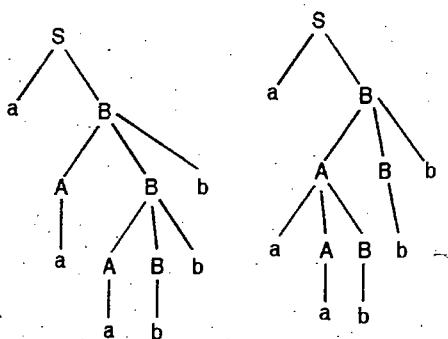


Fig. Ex. 4.3.15

Syllabus Topic : Simplification of CFG**4.4 Simplification of CFG**

A grammar written in a simple form is easy to analyze. Certain restrictions are imposed on simplified grammar. Simplification of CFG involves transforming CFG into an equivalent form that satisfies certain restrictions on its form. A CFG can be simplified by eliminating :

1. Useless symbols.
2. ϵ - productions.
3. Unit productions.

4.4.1 Elimination of Useless Symbols

SPPU - May 12

University Question

Q. Write a note on removal of useless symbols.

(May 2012, 3 Marks)

A grammar may contain symbols and productions which are not useful for derivation of strings. Two types of symbols are useless :

1. Non - generating symbols.
2. Non reachable symbol.

4.4.1.1 Non-generating Symbols

A symbol $X \in V$ (Set of variables) is a generating symbol if $X \xrightarrow[G]{*} w$,

where, $w \in T^*$ i.e. every variable must generate a string of terminals.

Example : Grammar

$$S \rightarrow Aa \mid Bb \mid a \mid b$$

$$A \rightarrow Aa \mid a$$

$$B \rightarrow bB$$

...Grammar (4.14)

Requires simplification as the symbol B is non-generating. Only production for B is

$$B \rightarrow bB$$

and it can not generate a string of terminals. The Grammar (4.14) can be simplified by deleting every production containing the useless symbol B. A simplified grammar is given in (4.15).

$$S \rightarrow Aa \mid a \mid b$$

$$A \rightarrow Aa \mid a$$

...Grammar (4.15)

A grammar containing a non-generating symbol V_i should be simplified by deleting every production containing the non-generating symbol V_i .



Finding non-generating symbols

There are two rules for finding a set of generating symbols for the given grammar.

1. Every symbol in T (terminal) is generating.
2. If there is a production $A \rightarrow \alpha$ and every symbol in α is generating, then A is generating.

Where $\alpha \in (V + T)^*$.

A symbol not in a set of generating symbols is said to be non-generating.

Example 4.4.1

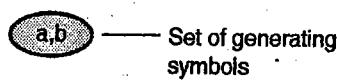
Find non-generating symbols in the grammar given below

$$S \rightarrow AB \mid CA \quad B \rightarrow BC \mid AB$$

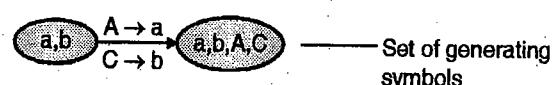
$$A \rightarrow a \quad C \rightarrow aB \mid b$$

Solution :

Step 1 : Every terminal is generating



Step 2 : A and C are generating as $A \rightarrow a$ and $C \rightarrow b$.



Two symbols A and C are added to the set of generating symbols using the Rule 2, which says if there is a production of the form $A \rightarrow \alpha$ and every symbol in α is generating then A is generating.

Step 3 : Symbol S is generating as there is a production $S \rightarrow CA$ with both C and A as generating.

Thus the symbol B is a non-generating symbol as it does not belong to the set of generating symbols.

Grammar in Example 4.0 can be simplified by deleting productions with the useless symbol B. Simplified grammar is given below :

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Example 4.4.2

Eliminate non-generating symbols from the grammar.

1. $P = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb, E \rightarrow aC\}$
2. $P = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC \mid DaA, C \rightarrow abb \mid DD, E \rightarrow aC, D \rightarrow aDA\}$

Solution :

| Grammar | Generating symbols | Non-generating symbols | Simplified grammar |
|---|--------------------|----------------------------|--|
| 1. $S \rightarrow aAa,$ $A \rightarrow Sb \mid bCC,$ $C \rightarrow abb,$ $E \rightarrow aC$ | | Every symbol is generating | Same as the original grammar $S \rightarrow aAa,$ $A \rightarrow Sb \mid bCC$ $C \rightarrow abb$ $E \rightarrow aC$ |



| Grammar | Generating symbols | Non-generating symbols | Simplified grammar |
|---|--|----------------------------|---|
| $\begin{aligned} 2. \quad S &\rightarrow aAa \\ A &\rightarrow Sb \mid bCC \mid DaA \\ C &\rightarrow abb \mid DD \\ E &\rightarrow aC \\ D &\rightarrow aDa \end{aligned}$ | <pre> graph TD a["a,b"] --> ab["a,b,C"] ab --> abc["a,b,C,E,A"] abc --> abcS["a,b,C,E,A,S"] abcS --> aAa["S → aAa"] </pre> | Symbol D is non-generating | $\begin{aligned} S &\rightarrow aAa \\ A &\rightarrow Sb \mid bCC \\ C &\rightarrow abb \\ E &\rightarrow aC \end{aligned}$ |

4.4.1.2 Non-reachable Symbols

A symbol X is reachable if it can be reached from the start symbol S.

i.e. if $S \xrightarrow[G]{*} \alpha$ and α contains a variable X then X is reachable.

A grammar containing a non-reachable symbol V_i should be simplified by deleting every production containing the non-reachable symbol V_i .

Finding non-reachable symbols

Non-reachable symbols can be located with the help of a dependency graph. A variable X is said to be dependent on S if there is a production

$$S \rightarrow \alpha_1 \times \alpha_2$$

The dependency graph is shown in Fig. 4.4.1.

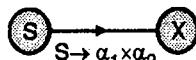


Fig. 4.4.1 : Depending Graph

- We must draw a dependency graph for all productions.
- If there is no path from the start symbol S to a variable X, then X is non-reachable.

Example 4.4.3

Eliminate non-reachable symbols from the given grammar.

$$P = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb, E \rightarrow aC\}$$

Solution :

Step 1 : Drawing of dependency graph :

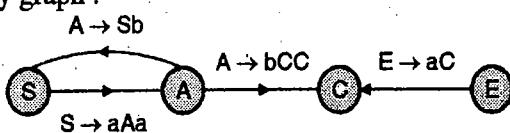


Fig. Ex. 4.4.3 : Dependency Graph

- From the production $S \rightarrow aAa$, A is dependent on S. It is shown using a directed edge from S to A.
- From the production $A \rightarrow bCC$, C is dependent on A. From the production $A \rightarrow Sb$, S is dependent of A.
- From the production $E \rightarrow aC$, C is dependent on E.

There is no path from S to E, E is non-reachable.



Step 2 : Simplification of grammar :

Grammar can be simplified by deleting every production containing the non-reachable symbol E. Simplified grammar is given below :

$$P_1 = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb\}$$

Example 4.4.4

Eliminate non-reachable symbols from the grammar :

1. $P = \{S \rightarrow aBa \mid BC, A \rightarrow aC \mid BCC, C \rightarrow a, B \rightarrow bCC, D \rightarrow E, E \rightarrow d\}$
2. $P = \{S \rightarrow aAa, A \rightarrow bBB, B \rightarrow ab, C \rightarrow aB\}$
3. $P = \{S \rightarrow aS \mid AB, A \rightarrow bA, B \rightarrow AA\}$

Solution :

| Grammar | Dependency graph | Non-reachable symbols | Reduced grammar |
|---|---|------------------------------|---|
| 1. $S \rightarrow aBa \mid BC$ $A \rightarrow aC \mid BCC$ $C \rightarrow a, B \rightarrow bCC$ $D \rightarrow E, E \rightarrow d$ | <pre> graph LR S((S)) -- "S → aBa" --> Ba(()) S((S)) -- "S → BC" --> BC(()) Ba(()) -- "A → aC" --> A(()) BC(()) -- "B → bCC" --> A(()) A(()) -- "D → E" --> E(()) </pre> | A, D and E are non-reachable | $S \rightarrow aBa \mid BC$ $C \rightarrow a$ $B \rightarrow bCC$ |
| 2. $S \rightarrow aAa$ $A \rightarrow bBB$ $B \rightarrow ab$ $C \rightarrow aB$ | <pre> graph LR S((S)) -- "S → aAa" --> Aa(()) Aa(()) -- "A → bBB" --> BB(()) BB(()) -- "B → ab" --> ab(()) ab(()) -- "C → aB" --> aB(()) </pre> | Symbol C is non-reachable | $S \rightarrow aAa$ $A \rightarrow bBB$ $B \rightarrow ab$ |
| 3. $S \rightarrow aS \mid AB$ $A \rightarrow bA$ $B \rightarrow AA$ | <pre> graph LR S((S)) -- "S → aS" --> Sa(()) S((S)) -- "S → AB" --> AB(()) Sa(()) -- "A → bA" --> BA(()) BA(()) -- "B → AA" --> AA(()) </pre> | Every symbol is reachable | Same as the original grammar |

4.4.2 Elimination of ϵ -productions

SPPU - May 12

University Question

Q. Write a note on removal of ϵ productions.

(May 2012, 3 Marks)

A production of the form $A \rightarrow \epsilon$, is called a null production or ϵ -production. For every context free grammar G with ϵ -productions, we can find a context-free grammar G_1 having no ϵ -productions such that

$$L(G_1) = L(G) - \{\epsilon\}$$

The procedure for finding G_1 (equivalent grammar without ϵ -productions) is as follows :

Step 1 : Find nullable variables

Step 2 : Addition of productions with nullable variables removed

Step 3 : Remove ϵ -productions.

Step 1 : Find nullable variables : A symbol X is nullable if

- 1) There is a production of the form $X \rightarrow \epsilon$.
- 2) If there is a production of the form $X \rightarrow \alpha$ and all symbols of α are nullable.



Example : In the grammar with productions $S \rightarrow ABA$, $A \rightarrow aA \mid \epsilon$, $B \rightarrow bB \mid \epsilon$

A is nullable as there is a production $A \rightarrow \epsilon$

B is nullable as there is a production $B \rightarrow \epsilon$

S is nullable as there is a production $S \rightarrow ABA$

With both A and B as nullable.

\therefore The set of nullable symbols = {S, A, B}

Step 2 : Addition of productions with nullable variables removed.

An ϵ -production has an effect. This effect must be added as productions to the existing grammar before ϵ -productions are removed.

Productions compensating the effect of ϵ -productions

For each production of the form $A \rightarrow \alpha_1$, create all possible productions of the form $A \rightarrow \alpha_2$, where α_2 is obtained from α_1 by removing one or more occurrences of nullable variables.

Example

| Sr. No. | Grammar with ϵ -productions | Nullable symbols | Grammar after addition of effect of nullable symbols | | | | | | | | |
|--------------------------|--|--|---|--------------------|--------------------|--------------------------|--|-------------------|--------------------------|--------------------------|--------------------------|
| 1. | $S \rightarrow aS$ $S \rightarrow \epsilon$ | {S} | $S \rightarrow aS$ $S \rightarrow a$ $S \rightarrow \epsilon$ [$S \rightarrow aS$ is written as $S \rightarrow a$ and $S \rightarrow aS$. S is nullable and hence $S \rightarrow aS$ can also be written as $S \rightarrow a$] | | | | | | | | |
| 2. | $S \rightarrow ABA$ $A \rightarrow \epsilon$ $B \rightarrow \epsilon$ | A and B are nullable as $A \rightarrow \epsilon$ and $B \rightarrow \epsilon$. S is null as every symbol on the right of $S \rightarrow ABA$ is nullable. $S \rightarrow ABA$ is nullable. \therefore Nullable symbols = {A, B, S} | $S \rightarrow ABA$ <table border="1"> <tr><td>$S \rightarrow AB$</td></tr> <tr><td>$S \rightarrow AA$</td></tr> <tr><td>$S \rightarrow BA$</td></tr> <tr><td>$S \rightarrow A$</td></tr> <tr><td>$S \rightarrow B$</td></tr> <tr><td>$S \rightarrow \epsilon$</td></tr> <tr><td>$A \rightarrow \epsilon$</td></tr> <tr><td>$B \rightarrow \epsilon$</td></tr> </table> <p>Productions derived from $S \rightarrow ABA$ by making one or more symbols null.</p> | $S \rightarrow AB$ | $S \rightarrow AA$ | $S \rightarrow BA$ | $S \rightarrow A$ | $S \rightarrow B$ | $S \rightarrow \epsilon$ | $A \rightarrow \epsilon$ | $B \rightarrow \epsilon$ |
| $S \rightarrow AB$ | | | | | | | | | | | |
| $S \rightarrow AA$ | | | | | | | | | | | |
| $S \rightarrow BA$ | | | | | | | | | | | |
| $S \rightarrow A$ | | | | | | | | | | | |
| $S \rightarrow B$ | | | | | | | | | | | |
| $S \rightarrow \epsilon$ | | | | | | | | | | | |
| $A \rightarrow \epsilon$ | | | | | | | | | | | |
| $B \rightarrow \epsilon$ | | | | | | | | | | | |
| 3. | $S \rightarrow aS \mid AB$ $A \rightarrow \epsilon$ $B \rightarrow \epsilon$ | $S \rightarrow aS$ <table border="1"> <tr><td>$S \rightarrow a$</td></tr> </table> $S \rightarrow AB$ <table border="1"> <tr><td>$S \rightarrow A$</td></tr> <tr><td>$S \rightarrow B$</td></tr> <tr><td>$S \rightarrow \epsilon$</td></tr> </table> \therefore Nullable set = {A, B, S} | $S \rightarrow a$ | $S \rightarrow A$ | $S \rightarrow B$ | $S \rightarrow \epsilon$ | $S \rightarrow aS$ $S \rightarrow AB$ $S \rightarrow A$ $S \rightarrow B$ $S \rightarrow \epsilon$ $A \rightarrow \epsilon$ $B \rightarrow \epsilon$ <p>Derived from $S \rightarrow As$</p> <p>Derived from $S \rightarrow AB$</p> | | | | |
| $S \rightarrow a$ | | | | | | | | | | | |
| $S \rightarrow A$ | | | | | | | | | | | |
| $S \rightarrow B$ | | | | | | | | | | | |
| $S \rightarrow \epsilon$ | | | | | | | | | | | |

Step 3 : Remove ϵ -productions from the grammar obtained in step 2.

1. $\{S \rightarrow aS \mid a\}$
2. $\{S \rightarrow ABA \mid AB \mid AA \mid BA \mid A \mid B\}$
3. $\{S \rightarrow aS \mid a \mid AB \mid S \rightarrow A \mid S \rightarrow B\}$

**Example 4.4.5**

Remove ϵ -productions from the given grammar.

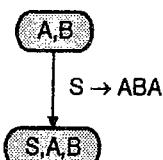
$S \rightarrow ABA, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid \epsilon$

Solution :

Step 1 : Find nullable variables

A and B are nullable as $A \rightarrow \epsilon$ and $B \rightarrow \epsilon$.

S is nullable as A and B are nullable and there is a production $S \rightarrow ABA$, with every symbol on the right as nullable.



\therefore Nullable set of symbols = {A, B, S}

Step 2 : Addition of productions with nullable variables removed.

$$S \rightarrow ABA \text{ can be written as } \left\{ \begin{array}{l} S \rightarrow ABA \\ S \rightarrow AB \\ S \rightarrow BA \\ S \rightarrow AA \\ S \rightarrow A \\ S \rightarrow B \\ S \rightarrow \epsilon \end{array} \right\}$$

$$A \rightarrow aA \text{ can be written as } \left\{ \begin{array}{l} A \rightarrow aA \\ A \rightarrow a \end{array} \right\}$$

$$B \rightarrow bB \text{ can be written as } \left\{ \begin{array}{l} B \rightarrow bB \\ B \rightarrow b \end{array} \right\}$$

\therefore Equivalent set of productions.

$$P_1 = \left\{ \begin{array}{l} S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \mid \epsilon \\ A \rightarrow aA \mid a \mid \epsilon \\ B \rightarrow bB \mid b \mid \epsilon \end{array} \right\}$$

Step 3 : Remove ϵ – productions

The final set of productions is given by

$$P_2 = \left\{ \begin{array}{l} S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{array} \right\}$$

Example 4.4.6

Remove ϵ -productions from the given grammar.

$S \rightarrow AB \quad A \rightarrow aAA \mid \epsilon \quad B \rightarrow bBB \mid \epsilon$

Solution :

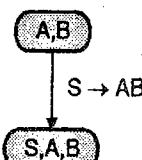
Step 1 : Find nullable variables

A and B are nullable as $A \rightarrow \epsilon$ and $B \rightarrow \epsilon$.

S is nullable as A and B are nullable

and $S \rightarrow AB$.

\therefore Nullable set of symbols = {S, A, B}



Step 2 : Addition of production with nullable variable removed.

$$S \rightarrow AB \text{ can be written as } \left\{ \begin{array}{l} S \rightarrow AB \\ S \rightarrow A \\ S \rightarrow B \\ S \rightarrow \epsilon \end{array} \right\}$$

By making one or more nullable symbols as null.

$$A \rightarrow aAA \text{ can be written as } \left\{ \begin{array}{l} A \rightarrow aAA \\ A \rightarrow aA \\ A \rightarrow a \end{array} \right\}$$

By making one or more nullable symbols as null.

$$B \rightarrow bBB \text{ can be written as } \left\{ \begin{array}{l} B \rightarrow bBB \\ B \rightarrow bB \\ B \rightarrow b \end{array} \right\}$$

By making one or more nullable symbols as null.

Step 3 : Remove ϵ -productions

\therefore Equivalent set of productions.

$$P_1 = \left\{ \begin{array}{l} S \rightarrow AB \mid A \mid B \\ A \rightarrow aAA \mid aA \mid a \\ B \rightarrow bBB \mid bB \mid b \end{array} \right\}$$

4.4.3 Elimination of Unit Productions

SPPU - May 12

University Question

Q. Write a note on removal of unit productions.

(May 2012, 3 Marks)

A production of the form

$$A \rightarrow B$$

It is known as a unit production. A and B are variables.

For every context free grammar G with unit productions, we can find a context free grammar G_1 having no unit productions such that

$$L(G) = L(G_1)$$

The procedure for finding G_1 (an equivalent grammar without unit productions) is as follows :

- The technique is based on expansion of unit production until it disappears. This technique works in most of the cases. This technique does not work if there is a cycle of unit productions such as

$$A_1 \rightarrow A_2, A_2 \rightarrow A_3, A_3 \rightarrow A_4 \text{ and } A_4 \rightarrow A_1$$



- The steps for elimination of unit production are as follows :

Step 1 : Add all non-unit production of G to G_1 .

Step 2 : Locate every pair of variables (A_i, A_j) such that $A_i \xrightarrow[G]{*} A_j$.

Step 3 : From pairs constructed in step 3, we can construct a chain like

$A_1 \rightarrow A_2 \dots A_j \rightarrow \alpha$, where $A_j \rightarrow \alpha$ is a non unit production.

Each variable A_i to A_j will derive α .

Example 4.4.7 SPPU - Dec. 14, 4 Marks

Eliminate unit productions from

$$P = \{ S \rightarrow ABA \mid BA \mid AA \mid AB \mid A \mid B \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \}$$

Solution :

Let G_1 with productions P_1 is an equivalent grammar without unit productions.

Step 1 : Add all non-unit productions of the given grammar to P_1 .

$$P_1 = \left\{ S \rightarrow ABA \mid BA \mid AA \mid AB \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \right\}$$

Step 2 : Locate every pair of variables (A_i, A_j) such

that $A_i \xrightarrow[G]{*} A_j$

1. (S, A) [Due to unit production $S \rightarrow A$]
2. (S, B) [Due to unit production $S \rightarrow B$]

Step 3 : Unit production $S \rightarrow A$ can be removed by expanding A.

A derives $aA \mid a$.

Unit production $S \rightarrow B$ can be removed by expanding $S \rightarrow B$. B derives $bB \mid b$.

Set of final productions for G_1 are given below :

| | Productions |
|-------------|---|
| From step 1 | $S \rightarrow ABA \mid BA \mid AA \mid AB$ $A \rightarrow aA \mid a$ $B \rightarrow bB \mid b$ |
| Pair (S, A) | $S \rightarrow aA \mid a$ |
| Pair (S, B) | $S \rightarrow bB \mid b$ |

Thus,

$$P_1 = \left\{ S \rightarrow ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bB \mid b \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \right\}$$

Example 4.4.8

Eliminate unit productions from the grammar

$$P = \{ E \rightarrow E+T \mid T, T \rightarrow T*T \mid F, \\ F \rightarrow (E) \mid I \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \}$$

Solution : Let the grammar G_1 , with productions P_1 is an equivalent grammar without unit productions.

Step 1 : Add all non-unit productions of the given grammar to P_1 .

$$P_1 = \left\{ E \rightarrow E+T \\ T \rightarrow T*T \\ F \rightarrow (E) \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \right\}$$

Step 2 : Locate every pair of variables (A_i, A_j) such

that $A_i \xrightarrow[G]{*} A_j$

1. (E, T) [Due to unit production $E \rightarrow T$]
2. (T, F) [Due to unit production $T \rightarrow F$]
3. (F, I) [Due to unit production $F \rightarrow I$]
4. (E, F) [Due to (E, T) and (T, F)]
5. (E, I) [Due to (E, T), (T, F) and (F, I)]
6. (T, I) [Due to (T, F) and (F, I)]

Step 3 : Unit productions are removed as shown below :

| | |
|-------------|---|
| From step 1 | $E \rightarrow E+T$ $T \rightarrow T*T$ $F \rightarrow (E)$ $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$ |
| Pair (E, T) | $E \rightarrow T*T$ |
| Pair (T, F) | $T \rightarrow (E)$ |
| Pair (F, I) | $F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$ |
| Pair (E, F) | $E \rightarrow (E)$ |
| Pair (E, I) | $E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$ |
| Pair (T, I) | $T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$ |

Thus, the final grammar is given by

$$P_1 = \left\{ E \rightarrow E+T \mid T*T \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ T \rightarrow T*T \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F \rightarrow (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \right\}$$

**Example 4.4.9**

Simplify the following grammar

$$S \rightarrow ASB \mid \epsilon \quad A \rightarrow aAS \mid a \quad B \rightarrow SbS \mid A \mid bb$$

Solution : Following sequence should be followed to simplify a grammar.

1. Eliminate to ϵ - productions from G and obtain G_1 .
2. Eliminate unit productions from G_1 and obtain G_2 .
3. Eliminate useless symbols from G_2 and obtain G_3 .

Step 1 : Eliminate ϵ - productions from G and obtain G_1 .

The set of nullable symbols = {S} as $S \rightarrow \epsilon$

| Grammar with ϵ production (G) | Grammar after taking effect of nullable symbols (G ₁) |
|---|--|
| $S \rightarrow ASB$ | $S \rightarrow ASB$ $S \rightarrow AB$] By making symbol S as null. |
| $S \rightarrow \epsilon$ $A \rightarrow aAS$ | $S \rightarrow \epsilon$ $A \rightarrow aAS$ $A \rightarrow aA$] By making symbol S as null |
| $A \rightarrow a$ $B \rightarrow SbS$ | $A \rightarrow a$ $B \rightarrow SbS$ $B \rightarrow Sb$] By making one or more S null $B \rightarrow bS$ $B \rightarrow b$ |
| $B \rightarrow A$ $B \rightarrow bb$ | $B \rightarrow A$ $B \rightarrow bb$ |

Final G_1 , after deletion of ϵ -productions is given below :

$$G_1 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid b \mid A \mid bb \end{array} \right\}$$

Step 2 : Eliminate unit productions from G_1 and obtain G_2 .

The grammar G_1 contains a unit production $B \rightarrow A$.

A derives $aAS \mid aA \mid b$

$$B \rightarrow A \rightarrow can be written as \rightarrow B \rightarrow aAS \mid aA \mid b$$

G_2 , after elimination of unit productions from G_1 is given by :

$$G_2 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \end{array} \right\}$$

Step 3 : Eliminate useless symbols from G_2 and obtain G_3 .

All the three variables S, A and B are both reachable and generating. Hence the grammar G_2 does not have useless symbols.

$$\therefore G_3 = G_2 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \mid a \end{array} \right\}$$

Example 4.4.10

Simplify the following grammar :

$$S \rightarrow 0A0 \mid 1B1 \mid BB$$

$$A \rightarrow C$$

$$B \rightarrow S \mid A$$

$$C \rightarrow S \mid \epsilon$$

Solution :

Step 1 : Eliminate ϵ -production from G (given grammar) and obtain G_1 . Set of nullable symbols are as shown in Fig. Ex. 4.4.10.

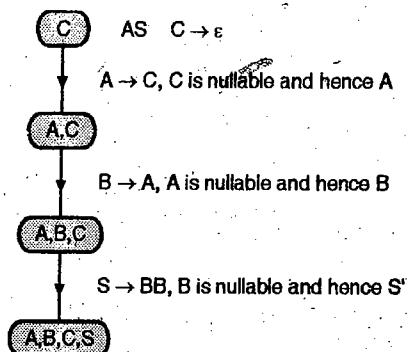


Fig. Ex. 4.4.10

The grammar after removal of null production is given below :

$$\left\{ \begin{array}{l} S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ A \rightarrow C, B \rightarrow S \mid A, C \rightarrow S \end{array} \right\} - \text{grammar } G_1$$

Step 2 : Following unit productions are there in G_1 .

$$S \rightarrow B, A \rightarrow C, B \rightarrow S, B \rightarrow A, C \rightarrow S$$

There is a chain.

$$B \rightarrow A \rightarrow C \rightarrow S \rightarrow B$$

$$\text{and } S \rightarrow B \rightarrow S$$

These chains are cyclic in nature of each is terminating in S.

The grammar without unit productions is given below :

$$\left\{ \begin{array}{l} S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ A \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ B \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ C \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \end{array} \right\}$$



Step 3 : The symbol C is not reachable and hence it can be deleted. The set of final productions is given below :

$$\begin{aligned} S &\rightarrow 0A0|00|1B1|11|BB \\ A &\rightarrow 0A0|00|1B1|11|BB \\ B &\rightarrow 0A0|00|1B1|11|BB \end{aligned}$$

Example 4.4.11

Simplify the following grammar

$$\begin{aligned} S \rightarrow Ab & \quad A \rightarrow a & B \rightarrow C \mid b \\ C \rightarrow D & \quad D \rightarrow E & E \rightarrow a \end{aligned}$$

Solution : Following sequence should be followed to simplify a grammar :

1. Eliminate ϵ productions from G and obtain G_1 .
2. Eliminate unit productions from G_1 and obtain G_2 .
3. Eliminate useless symbols from G_2 and obtain G_3 .

Step 1 : Eliminate ϵ productions from G and obtain G_1 .

Grammar does not have ϵ -productions.

$$\therefore G_1 = G$$

Step 2 : Eliminate unit productions from G_1 and obtain G_2 .

Step 2.1 : Add all non unit productions of the given grammar G_1 to productions P_2 of G_2 .

$$P_2 = \left\{ \begin{array}{l} S \rightarrow Ab, A \rightarrow a \\ E \rightarrow a \end{array} \right\}$$

Step 2.2 : Locate every pair of variables (A_i, A_j)

such that $A_i \xrightarrow[G_1]{*} A_j$

| | | |
|----|--------|---|
| 1. | (B, C) | [Due to unit production $B \rightarrow C$] |
| 2. | (C, D) | [Due to unit production $C \rightarrow D$] |
| 3. | (D, E) | [Due to unit production $D \rightarrow E$] |
| 4. | (B, D) | [Due to (B, C) and (C, D)] |
| 5. | (B, E) | [Due to (B, C), (C, D) and (B, E)] |
| 6. | (C, E) | [Due to (C, D) and (D, E)] |

Step 2.3 : Unit productions are removed through expansion.

From Step 2.1 $S \rightarrow Ab, A \rightarrow a, E \rightarrow a$

| | | |
|-------------|-------------------|---|
| Pair (B, C) | B \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |
| Pair (C, D) | C \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |
| Pair (D, E) | D \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |
| Pair (B, D) | B \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |
| Pair (B, E) | B \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |
| Pair (C, E) | C \rightarrow a | [there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$] |

Thus the productions after elimination of unit productions is

$$P_2 = \left\{ \begin{array}{l} S \rightarrow Ab \\ A \rightarrow a \\ E \rightarrow a \\ B \rightarrow a \\ C \rightarrow a \\ D \rightarrow a \end{array} \right\}$$

Step 3 : Elimination of useless symbols :

- o Every symbol, in the given grammar is generating.
- o Reachable symbols can be located by drawing a dependency graph.

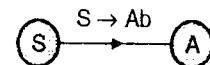


Fig. Ex. 4.4.11 : Dependency graph

Two symbols {S, A} are reachable. Other symbols {B, C, D, E} are non-reachable.

Final grammar G_3 with productions P_3 is obtained by deleting useless symbols.

$$P_3 = \left\{ \begin{array}{l} S \rightarrow Ab \\ A \rightarrow a \end{array} \right\}$$

Example 4.4.12

Find a reduced grammar equivalent to

$$S \rightarrow aC \mid SB$$

$$A \rightarrow bSCa$$

$$B \rightarrow aSB \mid bBC$$

$$C \rightarrow aBC \mid ad$$

Solution : Set of generating symbols can be found as shown in Fig. Ex. 4.4.12 :

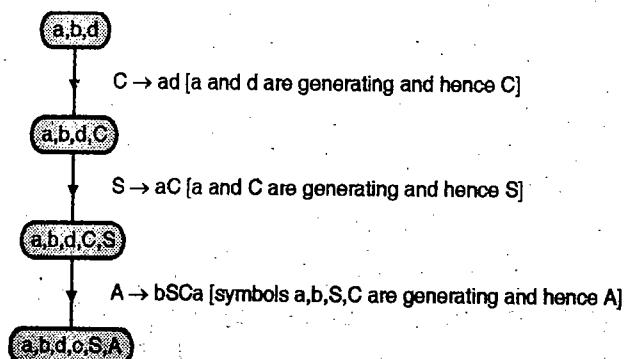


Fig. Ex. 4.4.12

∴ Non generating symbols = {B}

Thus the productions after elimination of non-generating symbols is :

$$P_1 = \left\{ \begin{array}{l} S \rightarrow aC \\ A \rightarrow bSCa \\ C \rightarrow ad \end{array} \right\}$$



The set of reachable symbols can be found by drawing a dependency graph.

The symbol {A} is non-reachable.

Thus the final set of productions after elimination of non-reachable symbols is :

$$P_2 = \{ S \rightarrow aC \\ C \rightarrow ad \}$$

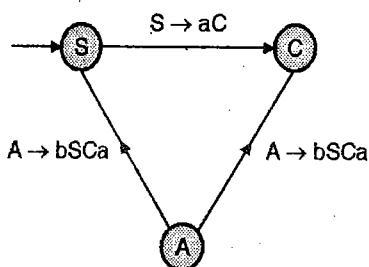


Fig. Ex. 4.4.12(a) : Dependency graph

Example 4.4.13 SPPU - May 16, 4 Marks

Simplify given grammar

$$S \rightarrow AaA$$

$$A \rightarrow Sb/bcc/\epsilon$$

$$C \rightarrow cc/abb$$

Solution :

Symbol A is nullable

∴ Simplified grammar is

$$S \rightarrow AaA \mid aA \mid a$$

$$A \rightarrow Sb/bcc$$

$$C \rightarrow cc/abb$$

Syllabus Topic : Normal Forms

4.5 Normal Forms for CFG

Productions in G, satisfying certain restrictions are said to be in normal form.

There are two normal forms for CFG.

1. Chomsky Normal Form (CNF)
2. Greibach Normal Form (GNF)

4.5.1 Chomsky Normal Form (CNF)

SPPU - Dec. 14

University Question

- Q. Describe Chomsky Normal Form (CNF) with suitable examples.** (Dec. 2014, 2 Marks)

A context free grammar (CFG) without ϵ -production is said to be in CNF if every production is of the form :

1. $A \rightarrow BC$, where $A, B, C \in V$.
2. $A \rightarrow a$, where $A \in V$ and $a \in T$.

The grammar should have no useless symbols.

- Every CFG without ϵ productions can be converted into an equivalent CNF form.

4.5.1.1 Algorithm for CFG to CNF Conversion

1. Eliminate ϵ -productions, unit productions and useless symbols from the grammar.
2. Every variable deriving a string of length 2 or more should consist only of variables.
i.e. every production of the form $A \rightarrow \alpha$ with $|\alpha| \geq 2$, α should consist only of variables.

Example : Consider a production $A \rightarrow V_1V_2aV_3bV_4$.

Terminal symbols a and b can be removed by rewriting the production

$$A \rightarrow V_1V_2aV_3bV_4$$

$$A \rightarrow V_1V_2C_aV_3C_bV_4$$

And adding two productions

$$C_a \rightarrow a$$

$$\text{and } C_b \rightarrow b$$

3. Every production deriving 3 or more variables ($A \rightarrow \alpha$ with $|\alpha| \geq 3$) can be broken down into a cascade of productions with each deriving a string of two variables.

Example : Consider a production $A \rightarrow X_1X_2\dots X_n$ where $n \geq 3$ and as X_i 's are variables.

The production $A \rightarrow X_1X_2\dots X_n$ should be broken down as given below :

$$A \rightarrow X_1C_1$$

$$C_1 \rightarrow X_2C_2$$

$$C_2 \rightarrow X_3C_3$$

:

$$C_{n-2} \rightarrow X_{n-1}X_n$$

Each with two variables on the right.

Example 4.5.1

Find the CNF equivalent to $S \rightarrow aAbB$, $A \rightarrow aA \mid a$, $B \rightarrow bB \mid b$.

Solution :

Step 1 : Simplification of grammar.

The grammar is already in a simple form without

1. ϵ -productions.
2. Unit-productions.
3. Useless symbols.

Step 2 : Every symbol in α , in a production of the form $A \rightarrow \alpha$ where $|\alpha| \geq 2$ should be a variable :



This can be done by adding two productions.

$$\begin{aligned} C_a &\rightarrow a \\ \text{and } C_b &\rightarrow b \end{aligned}$$

The set of productions after the above changes is :

$$P_2 = \left\{ \begin{array}{l} S \rightarrow C_a A C_b B \\ A \rightarrow C_a A \mid a \\ B \rightarrow C_b B \mid b \\ C_a \rightarrow a \\ C_b \rightarrow b \end{array} \right\}$$

Step 3 : Convert to CNF

| | |
|---|--|
| $S \rightarrow C_a C_1$ $C_1 \rightarrow AC_2$ $C_2 \rightarrow C_b B$ $A \rightarrow C_a A \mid a$ $B \rightarrow C_b B \mid b$ $C_a \rightarrow a$ $C_b \rightarrow b$ | Corresponding to $S \rightarrow C_a A C_b B$ |
| $\therefore P_3 = \left\{ \begin{array}{l} S \rightarrow C_a C_1, C_1 \rightarrow AC_2, C_2 \rightarrow C_b B \\ A \rightarrow C_a A \mid a, B \rightarrow C_b B \mid b, C_a \rightarrow a \\ C_b \rightarrow b \end{array} \right\}$ | |

Example 4.5.2 SPPU - Dec. 13, 8 Marks

Convert the grammar given below to its equivalent CNF :

$$S \rightarrow PQP \quad P \rightarrow 0P \mid \epsilon \quad Q \rightarrow 1Q \mid \epsilon$$

Solution :

Step 1 : Simplify the grammar

Step 1.1 : Eliminate ϵ productions Null symbols = {P, Q, S}

P and Q are nullable as $P \rightarrow \epsilon$ and $Q \rightarrow \epsilon$. S is also nullable as $S \rightarrow PQP$ with every symbol on the right as nullable.

Eliminating ϵ -productions, we get a set of productions P_1

$$P_1 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid PIQ \\ P \rightarrow 0P \mid 0 \\ Q \rightarrow 1Q \mid 1 \end{array} \right\}$$

Step 1.2 : Eliminate unit productions :

There are two unit productions $S \rightarrow P$ and $S \rightarrow Q$.

Eliminating unit productions from P_1 , we get a set of productions P_2

$$P_2 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid 0P \mid 0 \mid 1Q \mid 1 \mid PP \\ P \rightarrow 0P \mid 0 \\ Q \rightarrow 1Q \mid 1 \end{array} \right\}$$

Step 2 : Every symbol in α , in a production of the form $A \rightarrow \alpha$ with $|\alpha| \geq 2$ should be a variable.

This can be done by adding two productions

$$C_0 \rightarrow 0 ; \quad C_1 \rightarrow 1$$

The set of productions after the changes is given by P_3 .

$$P_3 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid PP \mid C_0 P \mid 0 \mid C_1 Q \mid 1 \\ P \rightarrow C_0 P \mid 0 \\ Q \rightarrow C_1 Q \mid 1 \\ C_0 \rightarrow 0 \\ C_1 \rightarrow 1 \end{array} \right\}$$

Step 3 : Convert to CNF :

| Original production | Equivalent productions in CNF |
|------------------------------|--|
| $S \rightarrow PQP$ | $S \rightarrow PC_2$ $C_2 \rightarrow QP$ |
| $S \rightarrow PQ$ | $S \rightarrow PQ$ |
| $S \rightarrow QP$ | $S \rightarrow QP$ |
| $S \rightarrow PP$ | $S \rightarrow PP$ |
| $S \rightarrow C_0 P$ | $S \rightarrow C_0 P$ |
| $S \rightarrow 0$ | $S \rightarrow 0$ |
| $S \rightarrow C_1 Q$ | $S \rightarrow C_1 Q$ |
| $S \rightarrow 1$ | $S \rightarrow 1$ |
| $P \rightarrow C_0 P \mid 0$ | $P \rightarrow C_0 P \mid 0$ |
| $Q \rightarrow C_1 Q \mid 1$ | $Q \rightarrow C_1 Q \mid 1$ |
| $C_0 \rightarrow 0$ | $C_0 \rightarrow 0$ |
| $C_1 \rightarrow 1$ | $C_1 \rightarrow 1$ |

\therefore The set of final productions in CNF is :

$$P_3 = \left\{ \begin{array}{l} S \rightarrow PC_2 \mid PQ \mid QP \mid PP \mid C_0 P \mid 0 \mid C_1 Q \mid 1 \\ C_2 \rightarrow QP \\ P \rightarrow C_0 P \mid 0 \\ Q \rightarrow C_1 Q \mid 1 \\ C_0 \rightarrow 0 \\ C_1 \rightarrow 1 \end{array} \right\}$$

Example 4.5.3 SPPU - Dec. 16, 6 Marks

Check whether the given grammar is in CNF

$$S \rightarrow bA \mid aB$$

$$A \rightarrow bAA \mid aS \mid a$$

$$B \rightarrow aBB \mid bS \mid b$$

If it is not in CNF, find the equivalent CNF.



Solution : The given grammar is not in CNF. Following productions are not allowed in CNF.

$$S \rightarrow bA \mid aB ; A \rightarrow bAA \mid aS ; B \rightarrow aBB \mid bS$$

Finding an equivalent CNF

Step 1 : Simplification of grammar – already simplified

Step 2 : Every symbol in α , in production of the form $A \rightarrow \alpha$ where $|\alpha| \geq 2$ should be a variable.

This can be done by adding two productions.

$$C_a \rightarrow a \quad \text{and} \quad C_b \rightarrow b$$

The set of productions after the above changes is

$$\left\{ \begin{array}{l} S \rightarrow C_bA \mid C_aB \\ A \rightarrow C_bAA \mid C_aS \mid a \\ B \rightarrow C_bBB \mid C_bS \mid b \\ C_a \rightarrow a, C_b \rightarrow b \end{array} \right\}$$

Step 3 : Finding an equivalent CNF:

| Original production | Equivalent productions in CNF |
|--------------------------------|--|
| $S \rightarrow C_bA \mid C_aB$ | $S \rightarrow C_bA \mid C_aB$ |
| $A \rightarrow C_bAA$ | $A \rightarrow C_bC_1$ $C_1 \rightarrow AA$ |
| $A \rightarrow C_aS \mid a$ | $A \rightarrow C_aS \mid a$ |
| $B \rightarrow C_bBB$ | $B \rightarrow C_bC_2$ $C_2 \rightarrow BB$ |
| $B \rightarrow C_bS \mid b$ | $B \rightarrow C_bS \mid b$ |
| $C_a \rightarrow a$ | $C_a \rightarrow a$ |
| $C_b \rightarrow b$ | $C_b \rightarrow b$ |

\therefore The set of final productions in CNF is :

$$\left\{ \begin{array}{l} S \rightarrow C_bA \mid C_aB \\ A \rightarrow C_bC_1 \mid C_aS \mid a \\ C_1 \rightarrow AA \\ B \rightarrow C_bC_2 \mid C_bS \mid b \\ C_2 \rightarrow BB \\ C_a \rightarrow a \\ C_b \rightarrow b \end{array} \right\}$$

Example 4.5.4

Design a CNF grammar for the set of strings of balanced parenthesis.

Solution : A context free grammar for balanced parentheses is given by

$$V = \{ \{S\}, \{(,)\}, \{S \rightarrow (S), S \rightarrow SS, S \rightarrow \epsilon\}, S \}$$

where the set of productions

$$P = \left\{ \begin{array}{l} S \rightarrow (S) \\ S \rightarrow SS \\ S \rightarrow \epsilon \end{array} \right\}$$

An equivalent set of productions without ϵ productions can be written as

$$P_1 = \left\{ \begin{array}{l} S \rightarrow (S) \mid () \\ S \rightarrow SS \end{array} \right\}$$

Conversion of P_1 in CNF can be done in two steps.

Step 1 : Addition of two productions $C_1 \rightarrow ($ and $C_2 \rightarrow)$ will change the set of productions to :

$$P_2 = \left\{ \begin{array}{l} S \rightarrow C_1SC_2 \mid C_1C_2 \\ S \rightarrow SS, C_1 \rightarrow (, C_2 \rightarrow) \end{array} \right\}$$

Step 2 : Finding an equivalent CNF :

| Production in CFG | Equivalent productions in CNF |
|-------------------------|--|
| $S \rightarrow C_1SC_2$ | $S \rightarrow C_1C_3$ $C_3 \rightarrow SC_2$ |
| $S \rightarrow C_1C_2$ | $S \rightarrow C_1C_2$ |
| $S \rightarrow SS$ | $S \rightarrow SS$ |
| $C_1 \rightarrow ($ | $C_1 \rightarrow ($ |
| $C_2 \rightarrow)$ | $C_2 \rightarrow)$ |

\therefore The set of final productions in CNF is

$$\left\{ \begin{array}{l} S \rightarrow C_1C_3 \mid C_1C_2 \mid SS \\ S \rightarrow SC_2 \\ C_1 \rightarrow (\\ C_2 \rightarrow) \end{array} \right\}$$

Example 4.5.5

Convert the following grammar to CNF. $S \rightarrow Aba, S \rightarrow aab, B \rightarrow Ac$

Solution :

Step 1 : Symbol B is non-reachable. The symbol can be deleted. The grammar after simplification is : $S \rightarrow Abalaab$

Step 2 : Every symbol is α , in a production of the form $A \rightarrow \alpha$ with $|\alpha| \geq 2$ should be a variable :

This can be done by adding two productions

$$C_a \rightarrow a \quad \text{and} \quad C_b \rightarrow b$$

The set of productions after above changes is :

$$S \rightarrow AC_bC_a \mid C_aC_aC_b$$



Step 3 : Convert to CNF

$$\begin{aligned} S &\rightarrow AC_1 \\ C_1 &\rightarrow C_b C_a \\ S &\rightarrow C_a C_2 \\ C_2 &\rightarrow C_a C_b \end{aligned}$$

Example 4.5.6

Convert the following grammar to CNF

$$\begin{aligned} S &\rightarrow AACD \\ A &\rightarrow aAb \mid \epsilon \\ C &\rightarrow aC \mid a \\ A &\rightarrow aDa \mid bDb \mid \epsilon \end{aligned}$$

Solution :

First of all, the grammar must be simplified.

Step 1 : Removing null productions.

$$\text{Nullable set} = \{A\}$$

Null productions are removed with the resulting set of production as :

$$\begin{aligned} S &\rightarrow AACD \mid ACD \mid CD \\ A &\rightarrow aAb \\ C &\rightarrow aC \\ A &\rightarrow aDa \mid bDb \end{aligned}$$

Step 2 : Removing non-generating symbol

Symbol S and D are non-generating.

Since, the starting symbol itself is non-generating, it is an **invalid grammar**.

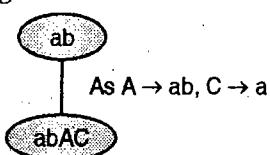


Fig. Ex. 4.5.6

Example 4.5.7

Convert the following grammar to Chomsky normal form (CNF).

$$G = (\{S\}, \{a\}, P, \{S\})$$

$$P = \{S \rightarrow a \ a \ a \ a \ a \ S, S \rightarrow a \ a \ a\}$$

Solution :

Step 1 : Re-writing of productions after addition of the new production $C_1 \rightarrow a$

$$\begin{aligned} S &\rightarrow C_1 C_1 C_1 C_1 C_1 S \\ S &\rightarrow C_1 C_1 C_1 \\ C_1 &\rightarrow a \end{aligned}$$

Original production

Equivalent productions in CNF

| | |
|--|---|
| 1. $S \rightarrow C_1 C_1 C_1 C_1 C_1 S$ | $S \rightarrow C_1 C_2$ $C_2 \rightarrow C_1 C_3$ $C_3 \rightarrow C_1 C_4$ $C_4 \rightarrow C_1 C_5$ $C_5 \rightarrow C_1 S$ |
| 2. $S \rightarrow C_1 C_1 C_1$ | $S \rightarrow C_1 C_6$ $C_6 \rightarrow C_1 C_1$ |
| 3. $C_1 \rightarrow a$ | $C_1 \rightarrow a$ |

Example 4.5.8 SPPU- Dec. 12, Aug. 15, 8 Marks

(1) Convert the following CFG to CNF

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid aa \mid bb$$

(2) Convert the following grammar in CNF

$$\begin{aligned} A &\rightarrow 01XY \\ X &\rightarrow 1XY \mid \epsilon \\ Y &\rightarrow YXa \mid X \mid \epsilon \end{aligned}$$

Solution :

(1)

Step 1 : Substituting 'A' for 'a' and 'B' for 'b', we get,

$$\begin{aligned} S &\rightarrow ASA \\ S &\rightarrow BSB \\ S &\rightarrow a \\ S &\rightarrow b \\ S &\rightarrow AA \\ S &\rightarrow BB \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Step 2 : Converting each production in CNF.

| | Production (original) | Productions in CNF |
|----|-----------------------|--|
| 1. | $S \rightarrow ASA$ | $S \rightarrow AA_1, A_1 \rightarrow SA$ |
| 2. | $S \rightarrow BSB$ | $S \rightarrow BA_2, A_2 \rightarrow SB$ |

Other productions are already in CNF.

(2)

Step 1 : Removing ϵ - productions

Symbol X and Y are nullable, hence the modified grammar without ϵ - productions will be

$$A \rightarrow 01 \mid 01X \mid 01Y \mid 01XY$$

$$X \rightarrow 1XY \mid 1 \mid 1X \mid 1Y$$

$$Y \rightarrow YXa \mid a \mid Ya \mid Xa \mid X$$



Step 2 : Unit production $Y \rightarrow X$ is removed and the equivalent grammar is re-produced.

$$A \rightarrow 01|01X|01Y|01XY$$

$$X \rightarrow 1XY|1|1X|1Y$$

$$Y \rightarrow YXa|a|Ya|Xa|1XY|1|1X|1Y$$

Step 3 : Grammar is written after substituting C_0 for 0, C_1 for 1 and C_a for a

$$A \rightarrow C_0C_1|C_0C_1X|C_0C_1Y|C_0C_1XY$$

$$X \rightarrow C_1XY|1|C_1X|C_1Y$$

$$Y \rightarrow YXC_a|a|YC_a|XC_a|C_1XY|1|C_1X|C_1Y$$

Step 4 :

| Sr. No. | Production in step 3 | Equivalent production in CNF |
|---------|---|--|
| 1. | $A \rightarrow C_0C_1$ | $A \rightarrow C_0C_1$ |
| 2. | $A \rightarrow C_0C_1X$ | $A \rightarrow C_0D_1, D_1 \rightarrow C_1X$ |
| 3. | $A \rightarrow C_0C_1Y$ | $A \rightarrow C_0D_2, D_2 \rightarrow C_1Y$ |
| 4. | $A \rightarrow C_0C_1XY$ | $A \rightarrow C_0D_3, D_3 \rightarrow C_1D_4, D_4 \rightarrow XY$ |
| 5. | $X \rightarrow C_1XY$ | $X \rightarrow C_1D_5, D_5 \rightarrow XY$ |
| 6. | $X \rightarrow 1 C_1X C_1Y$ | $X \rightarrow 1 C_1X C_1Y$ |
| 7. | $Y \rightarrow YXC_a$ | $Y \rightarrow YD_6, D_6 \rightarrow XC_a$ |
| 8. | $Y \rightarrow a YC_a XC_a 1 C_1X C_1Y$ | $Y \rightarrow a YC_a XC_a 1 C_1X C_1Y$ |
| 9. | $Y \rightarrow C_1XY$ | $Y \rightarrow C_1D_7, D_7 \rightarrow XY$ |

4.5.2 Greibach Normal Form (GNF)

SPPU - Dec. 14

University Question

Q. Describe Greibach Normal Form (GNF) with suitable examples. (Dec. 2014, 2 Marks)

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

$$A \rightarrow a\alpha,$$

where $a \in T$ is a terminal and α is a string of zero or more variables.

The language $L(G)$ should be without ϵ .

- Right hand side of each production should start with a terminal followed by a string of non-terminals of length zero or more.

4.5.2.1 Removing Left Recursion

Elimination of left recursion is an important step in algorithm used in conversion of a CFG into GNF form.

Left recursive grammar

A production of the form $A \rightarrow A\alpha$ is called left recursive as the left hand side variable

appears as the first symbol on the right hand side.

Language generated by left recursive grammar

Let us consider a CFG containing productions of the form

$$A \rightarrow A\alpha \dots \text{[Left recursive]}$$

$$A \rightarrow \beta \dots \text{[For termination of recursion]}$$

The language generated by above production is :

$$A \rightarrow A\alpha \quad \text{[From production } A \rightarrow A\alpha]$$

$$\rightarrow A\alpha\alpha \quad \text{[From production } A \rightarrow A\alpha]$$

$$\rightarrow A\alpha\alpha\alpha \quad \text{[From production } A \rightarrow A\alpha]$$

$$\vdots \quad \vdots$$

$$\rightarrow A\alpha^n \quad \text{[From production } A \rightarrow A\alpha]$$

$$\rightarrow \beta\alpha^n \quad \text{[From production } A \rightarrow \beta]$$

Right recursive grammar for $\beta\alpha^n$

A right recursive grammar for $\beta\alpha^n$ can be written as :

$$A \rightarrow \beta B | \beta$$

[where B generates a string α^n , production $A \rightarrow \beta$ is for termination of recursion]

$$B \rightarrow \alpha B | \alpha$$

Thus a left recursive grammar

$$A \rightarrow A\alpha | \beta$$

can be written using a right recursive grammar as :

$$A \rightarrow \beta B | \beta$$

Right recursive grammar

$$B \rightarrow \alpha B | \alpha$$

Example : A number of examples are given below for removing left-recursion.

| Grammar with left recursion | Language generated by grammar | Grammar without left recursion |
|--|---|--|
| 1. $A \rightarrow Aa b$ | {b, ba, baa, ... ba^n } | $A \rightarrow b bB$ $B \rightarrow aB a$ |
| 2. $A \rightarrow Aa b c$ | {b, ba, baa, ... ba^n , c, ca, caa, ... ca^n } | $A \rightarrow b c bB cB$ $B \rightarrow aB a$ |
| 3. $A_1 \rightarrow A_1A_2A_3$ A_2A_3 | { $A_2A_3, A_2A_3A_2A_3, \dots$, ..., $(A_2A_3)^n$ } | $A_1 \rightarrow A_2A_3$ $A_2A_3B_1$ $B_1 \rightarrow A_2A_3B_1$ A_2A_3 |



| Grammar with left recursion | Language generated by grammar | Grammar without left recursion |
|--|--|--|
| 4. $A_1 \rightarrow A_1A_2A_3 A_4A_1 A_5A_3$ | $\{A_4A_1, A_4A_1A_2A_3, \dots, A_4A_1(A_2A_3)^n, A_5A_3, A_5A_3A_2A_3, \dots, A_5A_3(A_2A_3)^n\}$ | $A_1 \rightarrow A_4A_1 A_5A_3 A_4A_1B_1 A_5A_3B_1 B_1 \rightarrow A_2A_3B_1 A_2A_3$ |
| 5. $S \rightarrow S10 0$ | $\{0, 010, 01010, \dots, 0(10)^n\}$ | $S \rightarrow 0B 0 B \rightarrow 10B 10$ |

4.5.2.2 Algorithm for Conversion from CFG to GNF

1. Eliminate ϵ -productions, unit productions and useless symbols from the grammar.
2. In production of the form $A \rightarrow X_1X_2\dots X_i\dots X_n$, other than X_1 , every other symbol should be a variable. X_1 could be a terminal.

Example : Consider a production

$$A \rightarrow V_1V_2aV_3bV_4$$

Terminal symbols a and b can be removed by rewriting the production

$$A \rightarrow V_1V_2aV_3bV_4 \text{ as}$$

$$A \rightarrow V_1V_2C_aV_3C_bV_4 \text{ and adding two productions.}$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

Thus, at the end of step 2 all productions must be of the forms.

- (a) $A \rightarrow \alpha$ (b) $A \rightarrow a$ (c) $A \rightarrow a\alpha$

where 'a' is a terminal and α is a string of non-terminals.

3. Rename variables as $A_1, A_2, A_3 \dots A_n$ to create A-productions.

Example : Consider a grammar given below

$$S \rightarrow aXS\bar{Y} | YS\bar{X} | b$$

The variables S, X and Y can be renamed as A_1, A_2 and A_3 respectively. Then the productions become

$$A_1 \rightarrow aA_2A_1A_3 | A_3A_1A_2 | b \quad [\text{A-productions}]$$

4. Modify the productions to ensure that if there is a production $A_i > A_j \alpha$ then i should be $\leq j$. If there is a production $A_i \rightarrow A_j \alpha$ with $i > j$, then we must generate productions substituting for A_j .
5. Repeating step 4, several times will guarantee that for every production $A_i \rightarrow A_j \alpha$, $i \leq j$.
6. Remove left recursion from every production of the form $A_k \rightarrow A_k \alpha$. B-productions should be added to remove left recursion.

7. Modify A_i -productions to the form $A_i \rightarrow a\alpha$, where a is a terminal and α is a string of non-terminals.
8. Modify B_i -productions to the form $B_i \rightarrow a\alpha$, where a is a terminal and α is a string of non-terminals.

Example 4.5.9 SPPU - Dec. 16, 6 Marks

Construct a grammar in GNF which is equivalent to the grammar $S \rightarrow AA | a, A \rightarrow SS | b$.

Solution :

Step 1 : Grammar is already in a simple form without

1. ϵ -productions.
2. Unit productions.
3. Useless symbol.

We can proceed for renaming of variables. Variables S and A are renamed as A_1 and A_2 respectively. The set of productions after renaming becomes :

| | |
|--|---|
| $A_1 \rightarrow A_2A_2$ $A_1 \rightarrow a$ $A_2 \rightarrow A_1A_1$ $A_2 \rightarrow b$ | Productions after renaming  |
|--|---|

Step 2 : Every production of the form $A_i \rightarrow A_j \alpha$ with $i > j$ must be modified to make $i \leq j$.

A_2 - production $A_2 \rightarrow A_1A_1$ should be modified.

↓

We must substitute $A_2A_2 | a$ for the first A_1 . We should not touch the second A_1 of A_1A_1 .

$$[A_2 \rightarrow A_1A_1] \Rightarrow \begin{bmatrix} A_2 \rightarrow A_2A_2A_1 \\ A_2 \rightarrow aA_1 \end{bmatrix}$$

The resulting set of productions is :

$$\begin{aligned} A_1 &\rightarrow A_2A_2 | a \\ A_2 &\rightarrow A_2A_2A_1 | aA_1 | b \end{aligned}$$

Step 3 : Removing left recursion :

The A_2 - productions $A_2 \rightarrow A_2A_2A_1 | aA_1 | b$ contains left recursion. Left recursion from A_2 -production can be removed through introduction of B_2 -production.

$$A_2 \rightarrow aA_1B_2 | bB_2$$

$$B_2 \rightarrow A_2A_1B_2 | A_2A_1$$

The resulting set of productions is :

$$A_1 \rightarrow A_2A_2 | a$$

$$A_2 \rightarrow aA_1B_2 | bB_2 | aA_1 | b$$

$$B_2 \rightarrow A_2A_1B_2 | A_2A_1$$



Step 4 : A_2 – productions are in GNF.

A_1 and B_2 productions can be converted to GNF with the help of A_2 –productions.

$$A_2 \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b \dots \text{in GNF}$$

$$A_1 \rightarrow A_2 A_2$$

↓ Substitute $aA_1B_2 \mid bB_2 \mid aA_1 \mid b$ for first A_2

$$A_1 \rightarrow aA_1B_2A_2 \mid bB_2A_2 \mid aA_1A_2 \mid bA_2$$

$$A_1 \rightarrow a \dots \text{in GNF}$$

Now, for B_2 – Production

$$B_2 \rightarrow A_2A_1B_2$$

↓ Substitute $aA_1B_2 \mid bB_2 \mid aA_1 \mid b$ for the first A_2

$$B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2$$

$$B_2 \rightarrow A_2A_1$$

↓ Substitute $aA_1B_2 \mid bB_2 \mid aA_1 \mid b$ for the first A_2

$$B_2 \rightarrow aA_1B_2A_1 \mid bB_2A_1 \mid aA_1A_1 \mid bA_1$$

The final set of productions is :

$$A_2 \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b$$

$$A_1 \rightarrow aA_1B_2A_1 \mid bB_2A_1 \mid aA_1A_2 \mid bA_2 \mid a$$

$$B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2 \mid aA_1B_2A_1 \mid bB_2A_1 \mid aA_1A_1 \mid bA_1$$

*A set of
productions
P*

where, Set of variables $V = (A_1, A_2, B_2)$

Set of terminals $T = (a, b)$

Start symbol = A_1

Set of productions P = Given above

Example 4.5.10

Find a grammar in GNF for the given CFG.

$$E \rightarrow E + T \mid T \quad T \rightarrow T * F \mid F \quad F \rightarrow (E) \mid a$$

Solution :

Step 1 : Grammar does not contain ϵ -productions and useless symbol.

Grammar has unit productions.

We first eliminate unit productions :

$E \rightarrow T$ and $T \rightarrow F$ are two unit productions with the chain $E \rightarrow T \rightarrow F$

Non unit productions are taken as it is :

$$E \rightarrow E + T$$

$$T \rightarrow T * F$$

$$F \rightarrow (E) \mid a$$

Productions for the following pairs are added :

$$(E, T) \Rightarrow \{E \rightarrow T * F\}$$

$$(E, F) \Rightarrow \{E \rightarrow (E) \mid a\}$$

$$(T, F) \Rightarrow \{T \rightarrow (E) \mid a\}$$

The resulting set of production is :

$$E \rightarrow E + T \mid T * F \mid (E) \mid a$$

$$T \rightarrow T * F \mid (E) \mid a$$

$$F \rightarrow (E) \mid a$$

Step 2 : Bringing every productions to the form $A \rightarrow A\alpha$, where α is a string of variables.

We can make the following substitutions : A for $+$, B for $*$ and C for (E) .

The resulting set of productions after the above substitutions is :

$$E \rightarrow EAT \mid TBF \mid (EC) \mid a$$

$$T \rightarrow TBF \mid (EC) \mid a$$

$$F \rightarrow (EC) \mid a$$

$$A \rightarrow +, B \rightarrow *, C \rightarrow)$$

Step 3 : Renaming of variables :

The variables E, A, T, B, F, C are renamed as $A_1, A_2, A_3, A_4, A_5, A_6$.

The resultant set of productions after the above substitutions is :

$$A_1 \rightarrow A_1A_2A_3 \mid A_3A_4A_5 \mid (A_1A_6) \mid a$$

$$A_3 \rightarrow A_3A_4A_5 \mid (A_1A_6) \mid a$$

$$A_5 \rightarrow (A_1A_6) \mid a, A_2 \rightarrow +, A_4 \rightarrow *, A_6 \rightarrow)$$

Step 4 : Every productions of the form $A_i \rightarrow A_j \alpha$ with $i > j$ must be modified to make $i \leq j$.

This step is not required as the productions are in the required form.

Step 5 : Remove left recursion.

A_1 production and A_3 production have left recursion.

- o Left recursion from A_1 -production can be removed through introduction of B_1 – production.

$$A_1 \rightarrow A_3A_4A_5B_1 \mid (A_1A_6B_1 \mid aB_1)$$

$$B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$$

- o Left recursion from A_3 -production can be removed through introduction of B_3 – production

$$A_3 \rightarrow (A_1A_6B_3 \mid aB_3)$$

$$B_3 \rightarrow A_4A_5B_3 \mid A_4A_5$$

The resulting set of productions is :

$$A_1 \rightarrow A_3A_4A_5B_1 \mid (A_1A_6B_1 \mid aB_1) \\ A_3A_4A_5 \mid (A_1A_6) \mid a$$

$$B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$$

$$A_3 \rightarrow (A_1A_6B_3 \mid aB_3) \mid (A_1A_6) \mid a$$

$$B_3 \rightarrow A_4A_5B_3 \mid A_4A_5$$



$$\begin{aligned}A_5 &\rightarrow (A_1 A_6 \mid a) \\A_2 &\rightarrow + \\A_4 &\rightarrow * \\A_6 &\rightarrow)\end{aligned}$$

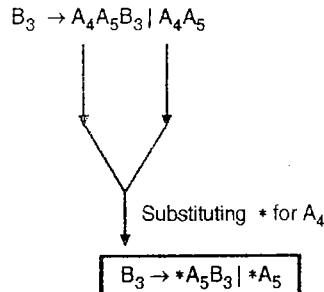
Step 6 : Productions for A_2 , A_3 , A_4 , A_5 and A_6 are in GNF. Productions for A_1 , B_1 and B_2 can be converted to GNF.

$$\begin{aligned}A_2 &\rightarrow + \\A_3 &\rightarrow (A_1 A_6 B_3 \mid a B_3 \mid (A_1 A_6 \mid a) \quad \text{Already in} \\A_4 &\rightarrow * \quad \text{GNF} \\A_5 &\rightarrow (A_1 A_6 \mid a) \\A_6 &\rightarrow) \\A_1 &\rightarrow A_3 A_4 A_5 B_1 \\&\quad \Downarrow \text{Substituting } (A_1 A_6 B_3 \mid a B_3 \mid (A_1 A_6 \mid a) \text{ for first } A_3 \\A_1 &\rightarrow (A_1 A_6 B_3 A_4 A_5 B_1 \mid a B_3 A_4 A_5 B_1 \mid \\&\quad (A_1 A_6 A_4 A_5 B_1 \mid a A_4 A_5 B_1) \\A_1 &\rightarrow (A_1 A_6 B_1 \mid a B_1 \dots \text{in GNF} \\A_1 &\rightarrow A_3 A_4 A_5 \\&\quad \Downarrow \text{Substituting } (A_1 A_6 B_3 \mid a B_3 \mid (A_1 A_6 \mid a) \text{ for the first } A_3 \\A_1 &\rightarrow (A_1 A_6 B_3 A_4 A_5 \mid a B_3 A_4 A_5 \mid \\&\quad (A_1 A_6 A_4 A_5 \mid a A_4 A_5) \\A_1 &\rightarrow (A_1 A_6 \mid a \dots \text{in GNF}\end{aligned}$$

The final set of productions is :

$$\begin{aligned}A_2 &\rightarrow + \\A_3 &\rightarrow (A_1 A_6 B_3 \mid a B_3 \mid (A_1 A_6 \mid a) \\A_4 &\rightarrow * \\A_5 &\rightarrow (A_1 A_6 \mid a) \\A_6 &\rightarrow) \\A_1 &\rightarrow (A_1 A_6 B_3 A_4 A_5 B_1 \mid a B_3 A_4 A_5 B_1 \mid \\&\quad (A_1 A_6 A_4 A_5 B_1 \mid a A_4 A_5 B_1 \mid \\&\quad (A_1 A_6 B_1 \mid a B_1 \mid \\&\quad (A_1 A_6 B_3 A_4 A_5 \mid a B_3 A_4 A_5 \mid \\&\quad (A_1 A_6 A_4 A_5 \mid a A_4 A_5 \mid (A_1 A_6 \mid a) \\B_1 &\rightarrow + A_3 B_1 \mid + A_3 \\B_3 &\rightarrow * A_5 B_3 \mid * A_5.\end{aligned}$$

$$\begin{aligned}B_1 &\rightarrow A_2 A_3 B_1 \mid A_2 A_3 \\&\quad \Downarrow \\&\quad \text{Substituting } + \text{ for } A_2 \\B_1 &\rightarrow + A_3 B_1 \mid + A_3\end{aligned}$$



Example 4.5.11 SPPU - May 15, 8 Marks

Give the GNF for following CFG

$$S \rightarrow AB \quad A \rightarrow BS \mid b \quad B \rightarrow SA \mid a$$

Solution :

Step 1 : Renaming of variables :

The variables S , A and B are renamed as A_1 , A_2 and A_3 respectively.

The resultant set of productions after above substitutions is :

$$\begin{aligned}A_1 &\rightarrow A_2 A_3 \\A_2 &\rightarrow A_3 A_1 \mid b \\A_3 &\rightarrow A_1 A_2 \mid a\end{aligned}$$

Step 2 : Every production of the form $A_i \rightarrow A_j \alpha$ with $i > j$. must be modified to make $i \leq j$.

A_3 - production $A_3 \rightarrow A_1 A_2$ should be modified

$$\begin{aligned}&\quad \Downarrow A_1 \text{ is changed as per } A_1 \rightarrow A_2 A_3 \\&\quad A_2 A_3 A_2 \\&\quad \Downarrow A_2 \text{ is changed as per } A_2 \rightarrow A_3 A_1 \mid b \\&\quad A_3 A_1 A_3 A_2 \mid b A_3 A_2.\end{aligned}$$

The resulting set of productions is :

$$\begin{aligned}A_1 &\rightarrow A_2 A_3 \\A_2 &\rightarrow A_3 A_1 \mid b \\A_3 &\rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a\end{aligned}$$

Step 3 : Remove left recursion

$A_3 \rightarrow$ Production has left recursion. Left recursion from A_3 - production can be removed through introduction of B_3 - Production

$$\begin{aligned}A_3 &\rightarrow b A_3 A_2 B_3 \mid a B_3 \\B_3 &\rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2\end{aligned}$$

The resulting set of production is

$$\begin{aligned}A_1 &\rightarrow A_2 A_3 \\A_2 &\rightarrow A_3 A_1 \mid b \\A_3 &\rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a \\B_3 &\rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2\end{aligned}$$



- Step 4 :**
- A₃ – productions are already in GNF
 - A₂ – production can be converted to GNF with the help of A₃ – productions.

$$A_2 \rightarrow A_3 A_1 | b$$

\Downarrow Substituting bA₃A₂B₃ | aB₃ | bA₃A₂ | a for A₃.

$$A_2 \rightarrow bA_3 A_2 B_3 A_1 | aB_3 A_1 | bA_3 A_2 A_1 | aA_1 | b$$

- A₁ – production can be converted to GNF with the help of A₂ – productions :

$$A_1 \rightarrow A_2 A_3$$

\Downarrow Substituting bA₃A₂B₃A₁ | aB₃A₁ | bA₃A₂A₁ | aA₁ | b for A₂

$$A_1 \rightarrow bA_3 A_2 B_3 A_1 A_3 | aB_3 A_1 A_3 | bA_3 A_2 A_1 A_3 | aA_1 A_3 | bA_3$$

- B₃ – productions can be converted to GNF with the help of A₁ – productions.

$$B_3 \rightarrow A_1 A_3 A_2 B_3 | A_1 A_3 A_2$$

\Downarrow Substituting bA₃A₂B₃A₁A₃ | aB₃A₁A₃A₂A₁A₃ | bA₃A₂A₁A₃A₂A₁A₃ | aA₁A₃ for A₁

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_2 B_3 | aB_3 A_1 A_3 A_2 B_3 | bA_3 A_2 A_1 A_3 A_2 B_3 | aA_1 A_3 A_2 B_3 | bA_3 A_2 A_1 A_3 A_2 B_3 | aA_1 A_3 A_2 B_3$$

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_2 A_1 | aB_3 A_1 A_3 A_2 A_1 | bA_3 A_2 A_1 A_3 A_2 A_1 | aA_1 A_3 A_2 A_1 | bA_3 A_2 A_1$$

The set of final productions in GNF is :

$$A_1 \rightarrow bA_3 A_2 B_3 A_1 A_3 | aB_3 A_1 A_3 | bA_3 A_2 A_1 A_3 | aA_1 A_3 | bA_3$$

$$A_2 \rightarrow bA_3 A_2 B_3 A_1 | aB_3 A_1 | bA_3 A_2 A_1 | aA_1 | b$$

$$A_3 \rightarrow bA_3 A_2 B_3 | aB_3 | bA_3 A_2 | a$$

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_2 B_3$$

| aB₃A₁A₃A₂B₃

| bA₃A₂A₁A₃A₂B₃

| aA₁A₃A₂B₃

| bA₃A₂A₁B₃

| bA₃A₂B₃A₁A₃A₂

| aB₃A₁A₃A₂A₁

| bA₃A₂A₁A₃A₂

| aA₁A₃A₂A₁

| bA₃A₂A₁

Example 4.5.12

Reduce the following grammar to GNF.

$$S \rightarrow AB, A \rightarrow BSB | BB | b$$

$$B \rightarrow aAb | a$$

Solution :

- Step 1 :** Making every symbol other than the first symbol (in derived string α in $A \rightarrow \alpha$) as a variable :

Variables C_b is substituted for b with resulting set of productions give as :

$$S \rightarrow AB$$

$$A \rightarrow BSB | BB | b$$

$$B \rightarrow aAC_b | a, C_b \rightarrow b$$

- Step 2 :** The variables S, A, B and C_b are renamed as A₁, A₂, A₃ and A₄ respectively. The resulting set of productions is to be given below.

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 A_3 | A_3 A_3 | b$$

$$A_3 \rightarrow aA_1 A_4 | a$$

$$A_4 \rightarrow b$$

Step 3 : Convert to CFG

| Given production | Equivalent Production in GNF |
|------------------|---------------------------------|
|------------------|---------------------------------|

$$A_4 \rightarrow b \rightarrow A_4 \rightarrow b$$

$$A_3 \rightarrow aA_1 A_4 | a \rightarrow A_3 \rightarrow aA_1 A_4 | a$$

$$A_2 \rightarrow A_3 A_1 A_3$$

\Downarrow

$$\xrightarrow{\text{Substituting } A_3} A_2 \rightarrow aA_1 A_4 A_1 A_3 | aA_1 A_3$$

$$A_2 \rightarrow A_3 A_3$$

\Downarrow

$$\xrightarrow{\text{Substituting } A_3} A_2 \rightarrow aA_1 A_4 A_3 | aA_3$$

$$A_2 \rightarrow b \rightarrow A_2 \rightarrow b$$

$$A_1 \rightarrow A_2 A_3$$

\Downarrow

$$\xrightarrow{\text{Substituting } A_2} A_1 \rightarrow aA_1 A_4 A_1 A_3 A_3 | aA_1 A_3 A_3 | aA_3 A_3 | bA_3$$

\therefore The final set of productions is :

$$A_1 \rightarrow aA_1 A_4 A_1 A_3 A_3 | aA_1 A_3 A_3 | aA_1 A_4 A_3 A_3 | aA_3 A_3 | bA_3$$

$$A_2 \rightarrow aA_1 A_4 A_1 A_3 | aA_1 A_3 | aA_1 A_4 A_3 | aA_3 | b$$



$$A_3 \rightarrow aA_1A_4 | a$$

$$A_4 \rightarrow b$$

Example 4.5.13

Convert the following grammar to Greiback normal form (GNF),

$$S \rightarrow BS, S \rightarrow Aa, A \rightarrow bc, B \rightarrow Ac$$

Solution :

Step 1 : Modifying productions to ensure that on R.H.S, other than the first symbol every other symbol must be a variable.

$$X \rightarrow a$$

$$Y \rightarrow c$$

$$S \rightarrow BS | AX$$

$$A \rightarrow bY$$

$$B \rightarrow AY$$

Step 2 : Remaining of variables.

The variables S, A, B, X, Y are renamed as A₁, A₂, A₃, A₄, A₅

$$A_4 \rightarrow a$$

$$A_5 \rightarrow c$$

$$A_1 \rightarrow A_3 A_1 | A_2 A_4$$

$$A_2 \rightarrow b A_5$$

$$A_3 \rightarrow A_2 A_5$$

Step 3 : Productions for A₄, A₅, A₂ are in the required form. Productions for A₁ and A₃ can be converted to the required form.

$$A_3 \rightarrow A_2 A_5$$

\Downarrow Substituting bA₅ for A₂

$$A_3 \rightarrow bA_5 A_5$$

\therefore The final set of productions is given by

$$A_1 \rightarrow bA_5 A_5 A_1 | bA_5 A_4$$

$$A_2 \rightarrow bA_5$$

$$A_3 \rightarrow bA_5 A_5$$

$$A_4 \rightarrow a$$

$$A_5 \rightarrow c$$

$$A_1 \rightarrow A_3 A_1 | A_2 A_4$$



Substituting bA₅A₅ for A₃
and bA₅ for A₂

$$A_1 \rightarrow bA_5 A_5 A_1 | bA_5 A_4$$

Example 4.5.14

Find GNF of the grammar given below

$$S \rightarrow ABAb | ab, B \rightarrow ABA | a, A \rightarrow a | b$$

Solution :

Step 1 : Making every symbol other than the first symbol (in derived string α in $A \rightarrow \alpha$) as a variable:

Variable C_b is substituted for b with resulting set of productions given as :

$$S \rightarrow AB AC_b | aC_b$$

$$B \rightarrow ABA | a$$

$$A \rightarrow a | b$$

$$C_b \rightarrow b$$

Step 2 : Substituting A $\rightarrow a | b$, in S $\rightarrow ABAC_b$ and B $\rightarrow ABA$, we can convert productions in GNF. The final set of productions is given by :

$$S \rightarrow aBAC_b | bBAC_b | aC_b$$

$$B \rightarrow aBA | bBA | a$$

$$A \rightarrow a | b$$

$$C_b \rightarrow b$$

Example 4.5.15

Find the GNF equivalent to the CFG

$$S \rightarrow AB, A \rightarrow aA | bB | b, B \rightarrow b$$

Solution :

A-productions A $\rightarrow aA | bB | b$ are in GNF

B-production B $\rightarrow b$ is in GNF

S-production S $\rightarrow AB$ is not in GNF.

It can be brought into GNF through substitution.

$$S \rightarrow AB$$



$$S \rightarrow aAB | bBB | bb$$

\therefore The required set of productions in GNF is :

$$S \rightarrow aAB | bBB | bb$$

$$A \rightarrow aA | bB | b$$

$$B \rightarrow b$$

Example 4.5.16 [SPPU- Dec. 15, 5 Marks]

Convert given CFG into Greibach Normal Form

$$S \rightarrow ABA, A \rightarrow aA | \epsilon, B \rightarrow bB | \epsilon$$

Solution :

Given grammar is

$$S \rightarrow ABA$$

$$A \rightarrow aA | \epsilon$$

$$B \rightarrow bB | \epsilon$$



Step 1 : Removing ϵ -productions, we get

$$\begin{aligned} S &\rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

Step 2 : Removing unit productions, we get

$$\begin{aligned} S &\rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

Step 3 : Writing productions in GNF.

To obtain productions in GNF, we can substitute $aA \mid a$ for leading A's and $bB \mid b$ for levelling B's.

| Productions in step 2 | Equivalent productions in GNF |
|---|--|
| 1. $S \rightarrow ABA$ | $S \rightarrow aABA \mid aBA$ |
| 2. $S \rightarrow AB$ | $S \rightarrow aAB \mid aB$ |
| 3. $S \rightarrow BA$ | $S \rightarrow bBA \mid bA$ |
| 4. $S \rightarrow AA$ | $S \rightarrow aAA \mid aA$ |
| 5. $S \rightarrow aA \mid a \mid bB \mid b$ | $S \rightarrow aA \mid a \mid bB \mid b$ |
| 6. $A \rightarrow aA \mid a$ | $A \rightarrow aA \mid a$ |
| 7. $B \rightarrow bB \mid b$ | $B \rightarrow bB \mid b$ |

Syllabus Topic : Chomsky Hierarchy

4.6 Chomsky Classification for Grammar

SPPU - Dec. 13, Aug. 15

University Questions

- Q. Explain Chomsky Hierarchy of Grammar.
 (Dec. 2013, 6 Marks)
- Q. Write short note on Chomsky Hierarchy.
 (Aug. 2015/In Sem.), 4 Marks

A grammar can be classified on the basis of production rules. Chomsky classified grammars into the following types :

1. Type 3 : Regular grammar
2. Type 2 : Context free grammar
3. Type 1 : Context sensitive grammar
4. Type 0 : Unrestricted grammar.

4.6.1 Type 3 or Regular Grammar

A grammar is called type 3 or regular grammar if all its productions are of the following forms :

$$\begin{array}{ll} A \rightarrow \epsilon & A \rightarrow a \\ A \rightarrow aA & A \rightarrow Ba \end{array}$$

where $a \in \Sigma$ and $A, B \in V$.

A language generated by type 3 grammar is known as regular language.

4.6.2 Type 2 or Context Free Grammar

A grammar is called type 2 or context free grammar if all its productions are of the following form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup T)^*$.

V is a set of variables and T is a set of terminals.

The language generated by a type 2 grammar is called a context free language. A regular language but not the reverse.

4.6.3 Type 1 or Context Sensitive Grammar

A grammar is called a type 1 or context sensitive grammar if all its productions are of the following form.

$$\alpha \rightarrow \beta,$$

where β is atleast as long as α .

Example 4.6.1

Write a set of production for the strings of the form $a^n b^n c^n$.

Solution : The set of productions is given by :

$$P = \left\{ \begin{array}{l} S \rightarrow aSBC \mid aBC \\ CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, \\ bC \rightarrow bc, cC \rightarrow cc \end{array} \right\}$$

A string of the form $a^n b^n c^n$ can be generated as given below.

$$S \xrightarrow{*} a^{n-1} S(BC)^{n-1}$$

$$\Rightarrow a^{n-1} aBC(BC)^{n-1}$$

$$\Rightarrow a^{n-1} abC(BC)^{n-1}$$

$$\Rightarrow a^{n-1} ab^{n-1} C^n$$

[by applying $CB \rightarrow BC, n-1$ times]

[by applying $aB \rightarrow ab$]

$$\Rightarrow a^{n-1} ab^{n-1} bC^n$$

$$\Rightarrow a^n b^{n-1} bcC^{n-1}$$

[by applying $bB \rightarrow bb$ several times]

$$\Rightarrow a^n b^{n-1} bcC^{n-1}$$

$$\Rightarrow a^n b^n c^n$$

[by applying $cC \rightarrow cc$ several times]

4.6.4 Type 0 or Unrestricted Grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the $\alpha \rightarrow \beta$, then length of α could be more than length of β .

- Every grammar also is a type 0 grammar.
- A type 2 grammar is also a type 1 grammar
- A type 3 grammar is also a type 2 grammar.



CHAPTER

5

Regular Grammar

Syllabus

Regular grammar, equivalence of RG(LRG and RLG) and FA.

Syllabus Topic : Regular Grammar

5.1 Regular Grammar

SPPU - Dec. 15

University Question

Q. Define : Regular Grammar with suitable example.
(Dec. 2015, 3 Marks)

The language accepted by finite automata can be described using a set of productions known as regular grammar. The productions of a regular grammar are of the following form :

$$A \rightarrow a$$

$$A \rightarrow aB$$

$$A \rightarrow Ba$$

$$A \rightarrow \epsilon$$

Where, $a \in T$ and $A, B \in V$.

A language generated by a regular grammar is known as regular language.

A regular grammar could be written in two forms :

1. Right-linear form
2. Left-linear form.

Right-linear form :

A right linear regular grammar will have production of the given form.

$$A \rightarrow a$$

$$A \rightarrow aB$$

Variable B in $A \rightarrow aB$ is the second symbol on the right.

$$A \rightarrow \epsilon$$

Left-linear form :

A left linear regular grammar will have productions of the following form :

$$A \rightarrow a$$

$$A \rightarrow Ba$$

Variable B in $A \rightarrow Ba$ is the first symbol on the right.

$$A \rightarrow \epsilon$$

Syllabus Topic : Equivalence of RLG and FA

5.2 DFA to Right Linear Regular Grammar

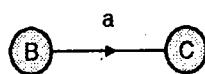
Every DFA can be described using a set of production using the following steps :-

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

Let the corresponding right linear grammar

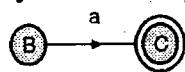
$$G = (V, T, P, S)$$

1. Rename $q_0 \in Q$ as $S \in V$, relating start state of M with starting symbol of G .
2. Rename states of Q as $A, B, C, D, \dots \in V$
Where, $A, B, C, D, \dots \in V$
3. Creating a set of productions P .
 - 3.1 If $q_0 \in F$ then add a production
 $S \rightarrow \epsilon$ to P .
 - 3.2 For every transition of the form,



add a production $B \rightarrow aC$,
where C is a non-accepting state.

- 3.3 For every transition of the form



add two productions $B \rightarrow aC$, $B \rightarrow a$,
where C is an accepting state.

Example 5.2.1

Give a right linear grammar for the DFA shown in Fig. Ex. 5.2.1.

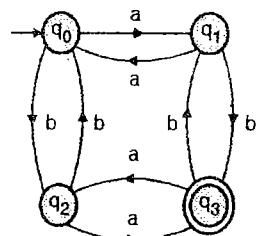


Fig. Ex. 5.2.1

Solution :

Step 1 : Renaming of states, we get the DFA shown in Fig. Ex. 5.2.1(a).

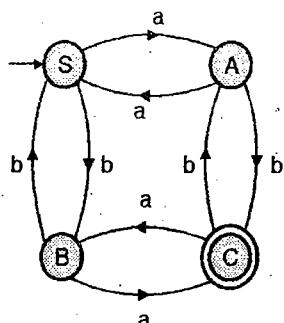


Fig. Ex. 5.2.1(a)

Step 2 : Set of productions are given by :

$$P = \left\{ \begin{array}{l} S \rightarrow aA \mid bB \\ A \rightarrow aS \mid bC \mid b \\ B \rightarrow bs \mid aC \mid a \\ C \rightarrow aB \mid bA \end{array} \right\}$$

Where,

The set of variables $V = \{S, A, B, C\}$

The set of terminals $T = \{a, b\}$

The start symbol = S

Example 5.2.2

Give a right linear grammar for the DFA shown in Fig. Ex. 5.2.2.

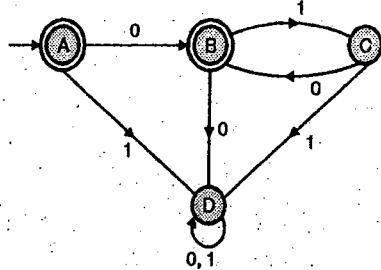


Fig. Ex. 5.2.2

Solution :

Step 1 : State D is a dead state and it can be removed. The FA after deletion of dead state D is shown in Fig. Ex. 5.2.2(a).

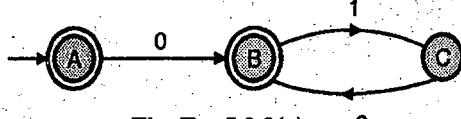


Fig. Ex. 5.2.2(a)

Step 2 : The set of productions are given below.

$$P = \left\{ \begin{array}{ll} A \rightarrow \epsilon, & - \text{Start state is a final state} \\ A \rightarrow 0B \mid 0 & - \text{Transition from A to B on 0} \\ B \rightarrow 1C & - \text{Transition from B to C on 1} \\ C \rightarrow 0B \mid 0 & - \text{Transition from C to B on 0} \end{array} \right.$$

where, Set of variables $V = \{A, B, C\}$

Set of terminals $T = \{0, 1\}$

Start symbol = A.

Example 5.2.3

Obtain a grammar to generate a string consisting of any number of a's and b's with at least one a.

Solution :

Step 1 : DFA for the given language.

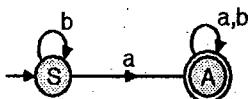


Fig. Ex. 5.2.3

Step 2 : Grammar from the DFA

$$\begin{aligned} S &\rightarrow bS \mid aA \mid a \\ A &\rightarrow aA \mid bA \mid a \mid b \end{aligned}$$

Example 5.2.4

Construct a regular grammar G generating the regular set represented by $P = a^*b(a + b)^*$.

Solution : We can draw an FA for $a^*b(a + b)^*$ and from the F.A., we can easily write the regular grammar.

Step 1 : FA for $a^*b(a + b)^*$

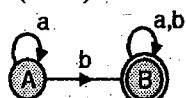


Fig. Ex. 5.2.4

Step 2 : Regular grammar from FA.

$$\begin{aligned} A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow aB \mid bB \mid a \mid b \end{aligned}$$

Example 5.2.5 SPPU - May 16, 4 Marks

Obtain regular expression to the following regular grammar

$$S \rightarrow aA \mid bB$$

$$A \rightarrow bA \mid a$$

$$B \rightarrow aB \mid b$$

Solution :

Drawing FA for the given regular grammar, we get

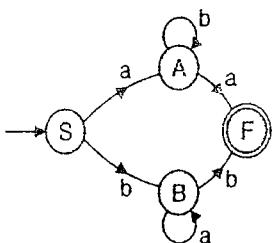


Fig. Ex. 5.2.5

$$\therefore \text{RE} = ab^*a + ba^*b$$

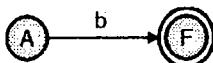
5.2.1 Right Linear Grammar to DFA

Every right linear grammar can be represented using a DFA

- A production of the form $A \rightarrow aB$
Will generate a transition
- A production of the form $A \rightarrow aB \mid a$ will generate a transition , provided every transition entering B terminates in B.
- A production of the form $A \rightarrow \epsilon$ will make A a final state.



- An independent production of the form $A \rightarrow b$, will generate a transition



Where, F is a new state and it should be a final state.

Example 5.2.6

Convert the following right-linear grammar to an equivalent DFA.

$$S \rightarrow bB \quad B \rightarrow Bc$$

$$B \rightarrow aB \quad C \rightarrow a$$

$$B \rightarrow b$$

Solution : Re-writing the production we get

$$S \rightarrow bB, \quad B \rightarrow bC \mid b \quad B \rightarrow aB \quad C \rightarrow a$$

Step 1 : Adding transitions corresponding to every production, we get

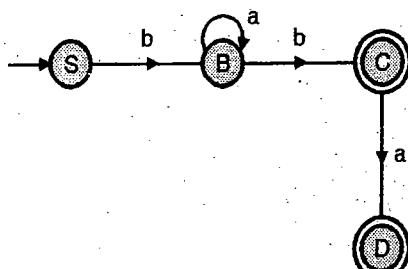


Fig. Ex. 5.2.6(a)

Step 2 : Adding a state E to handle ϕ -transitions, we get the final DFA

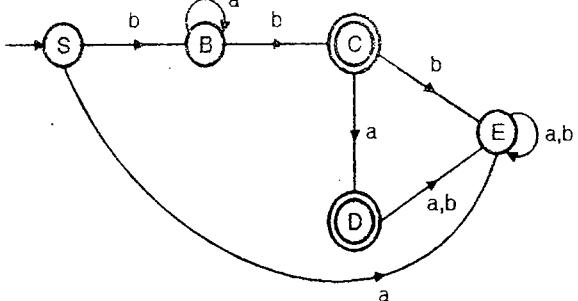


Fig. Ex. 5.2.6(b) : Final DFA

Example 5.2.7

Convert following RG to DFA

$$S \rightarrow 0A \mid 1B$$

$$A \rightarrow 0C \mid 1A \mid 0$$

$$B \rightarrow 1B \mid 1A \mid 1$$

$$C \rightarrow 0 \mid 0A.$$

Solution : A new final state F is being introduced to handle productions like,

$$A \rightarrow 0, B \rightarrow 1, C \rightarrow 0.$$

Step 1 : Adding transitions corresponding to every production, we get the FA shown in Fig. Ex. 5.2.7.

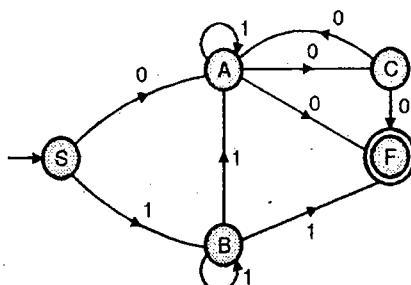


Fig. Ex. 5.2.7

Step 2 : Drawing an equivalent DFA, we get

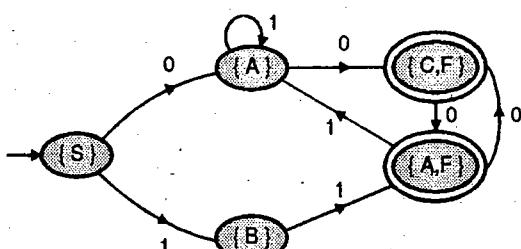


Fig. Ex. 5.2.7(a)

Step 3 : States $\{S\}$, $\{A\}$, $\{B\}$, $\{C,F\}$, and $\{A,F\}$ are renamed as q_0 , q_1 , q_2 , q_3 , q_4 and a dead state q_ϕ is introduced to handle ϕ - transitions. The resulting DFA is shown in Fig. Ex. 5.2.7(b).

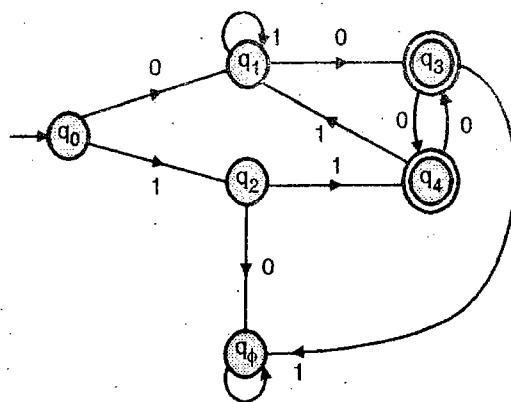


Fig. Ex. 5.2.7(b) : Final DFA

Example 5.2.8

Let $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$ where

$$\begin{aligned} P = & \{A_0 \rightarrow a A_1 \\ & A_1 \rightarrow b A_1 \\ & A_1 \rightarrow a \\ & A_1 \rightarrow b A_0 \\ & \} \end{aligned}$$

Construct a FA equivalent to G

Solution :

Step 1 : From grammar to transition system

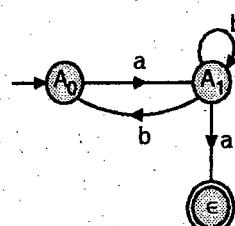


Fig. Ex. 5.2.8

Step 2 : Renaming the state e as A_2 and constructing an equivalent FA we get:

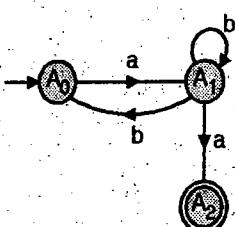


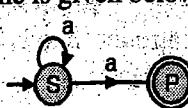
Fig. Ex. 5.2.8(a)

Example 5.2.9

If a regular grammar G is given by $S \rightarrow aS \mid a$, find M accepting $L(G)$.

Solution :

The required machine is given below.

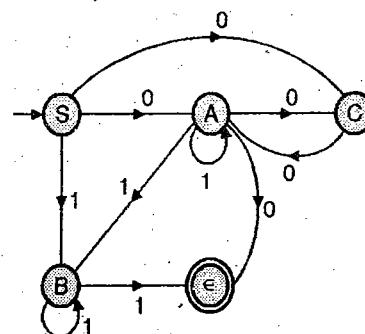
**Example 5.2.10 SPPU - Dec. 12, 6 Marks**

Construct a DFA to accept the language generated by the left linear grammar given below : $S \rightarrow B1 \mid A0 \mid C0$

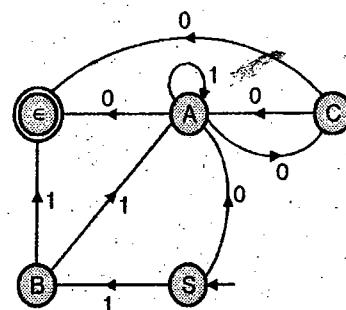
$$\begin{aligned} B &\rightarrow B1 \mid 1 \\ A &\rightarrow A1 \mid B1 \mid C0 \mid 0 \\ C &\rightarrow A0 \end{aligned}$$

Solution :

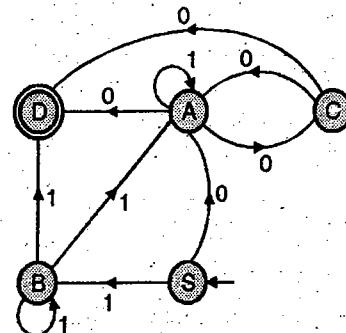
Step 1 : Transition system for the left linear grammar



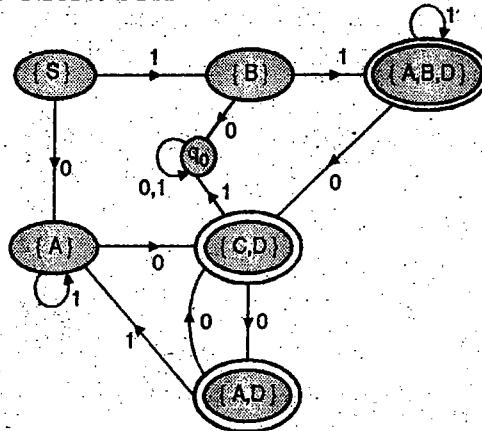
Step 2 : Interchanging the start state and final state and changing direction of all transitions, we get



Step 3 : Rename ϵ -state as state D



Step 4 : NFA to DFA



Syllabus Topic : Equivalence of LRG and FA

5.3 DFA to Left-linear Grammar

Following steps are required to write a left linear grammar corresponding to a DFA :

1. Interchange starting state and the final state.
2. Reverse the direction of all the transitions.
3. Write the grammar from the transition graph in left-linear form.

Example 5.3.1

Give a left linear grammar for the DFA shown in Fig. Ex. 5.3.1.

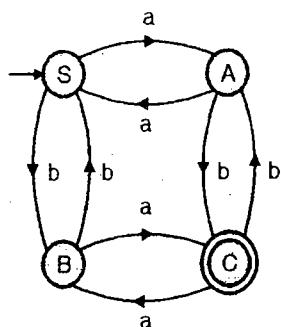


Fig. Ex. 5.3.1

Solution :

Step 1 : Interchanging the starting state and the final state and reversing the direction of transitions, we get a transition graph as shown in Fig. Ex. 5.3.1(a).

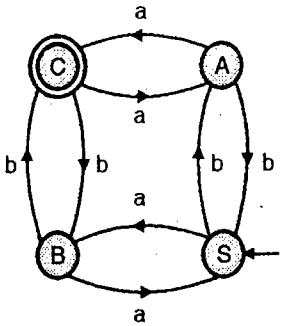


Fig. Ex. 5.3.1(a)

Step 2 : Writing an equivalent left linear grammar, we get :

$$\begin{aligned} S &\rightarrow Ba \mid Ab, A \rightarrow Sb \mid Ca \mid a \\ B &\rightarrow Sa \mid Cb \mid b, C \rightarrow Bb \mid Aa. \end{aligned}$$

Example 5.3.2

Give a left linear grammar for the DFA shown in Fig. Ex. 5.3.2.

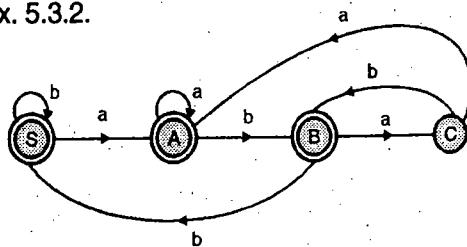


Fig. Ex. 5.3.2

Solution :

Step 1 : The given DFA contains three final states S, A and B. We can draw an equivalent DFA with a single final state D by adding ϵ - transitions from S to D, A to D and B to D.

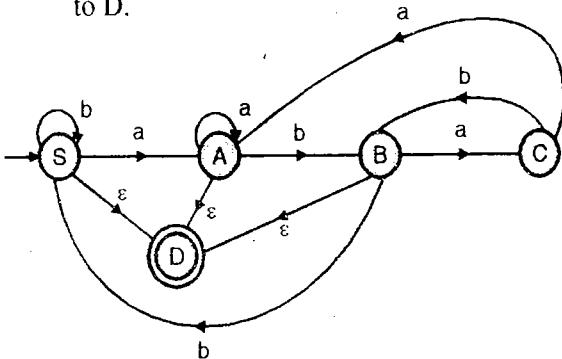


Fig. Ex. 5.3.2(a)

Step 2 : Interchanging the starting state and the final state and reversing direction of transitions, we get.

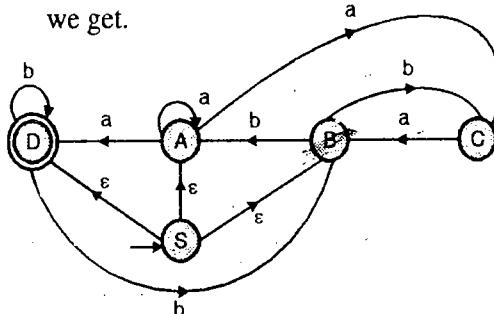


Fig. Ex. 5.3.2(b)

Step 3 : Writing an equivalent left linear grammar we get :

$$\begin{aligned} S &\rightarrow D \\ S &\rightarrow A \\ S &\rightarrow B \\ C &\rightarrow Ba \\ B &\rightarrow Cb \mid Ab \\ A &\rightarrow Aa \mid Da \mid Ca \mid a \\ D &\rightarrow Bb \mid Db \mid b \end{aligned}$$

Step 4 : Removing unit productions, the resulting productions are :

$$\begin{aligned} S &\rightarrow Cb \mid Ab \mid Aa \mid Da \mid Ca \mid a \mid Bb \mid Db \mid b \\ C &\rightarrow Ba \\ B &\rightarrow Cb \mid Ab \\ A &\rightarrow Aa \mid Da \mid Ca \mid a \\ D &\rightarrow Bb \mid Db \mid b \end{aligned}$$

5.3.1 Left Linear Grammar to DFA

Every left linear grammar can be represented using an equivalent DFA. Following steps are required to draw a DFA for a given left linear grammar.

1. Draw a transition graph from the given left linear grammar.
2. Reverse the direction of all the transitions.
3. Interchange starting state and the final state.
4. Carry out conversion from FA to DFA.

Example 5.3.3 SPPU - May 12, 4 Marks

Construct DFA accepting the regular language generated by the left linear grammar given below.

$$S \rightarrow Ca \mid Bb$$

$$C \rightarrow Bb$$

$$B \rightarrow Ba \mid b$$

Solution :

Step 1 : Draw a transition graph from the given left linear grammar.

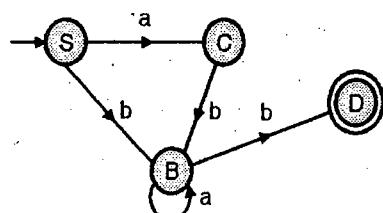


Fig. Ex. 5.3.3 : Transition graph

D is an accepting state. It is added for the production $B \rightarrow b$.

Step 2 : Reverse the direction of transitions and interchange starting state and the final state.

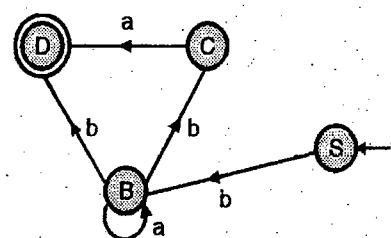


Fig. Ex. 5.3.3(a)

Step 3 : Conversion from FA to DFA

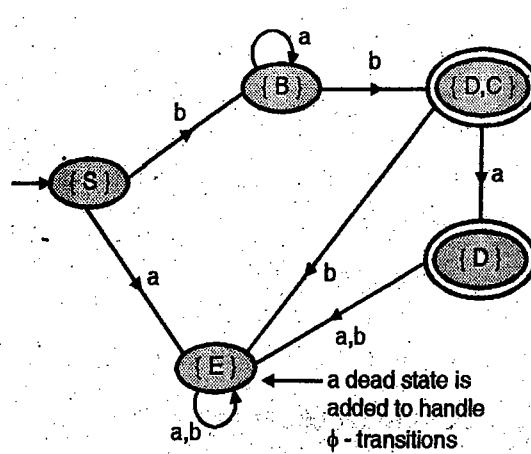


Fig. Ex. 5.3.3(b) : DFA

Example 5.3.4

Construct DFA accepting the language generated by the left linear grammar given below.

$$S \rightarrow B1 \mid A0 \mid C0$$

$$B \rightarrow B1 \mid 1$$

$$A \rightarrow A1 \mid B1 \mid C0 \mid 0$$

$$C \rightarrow A0$$

Solution :

Step 1 : Draw a transition graph from the given left linear grammar

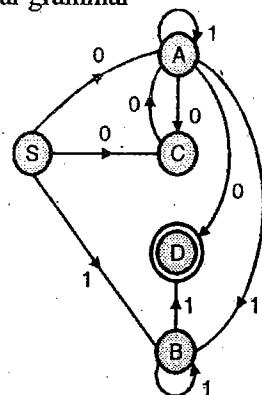


Fig. Ex. 5.3.4(a)

State D is a final state. It is added to handle the following transitions :

$$B \rightarrow 1, A \rightarrow 0$$

Step 2 : Reverse the direction of transitions and interchange starting state and the final state.

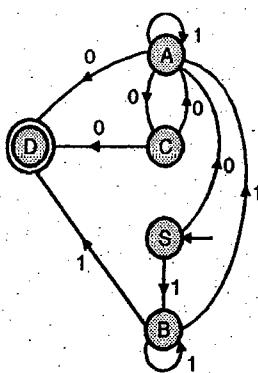


Fig. Ex. 5.3.4(b)

Step 3 : Conversion from FA to DFA.

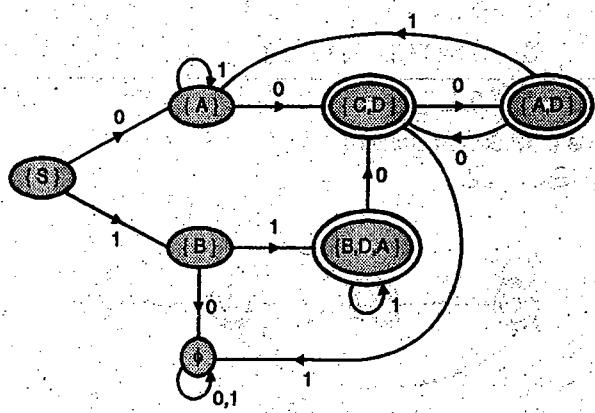


Fig. Ex. 5.3.4(c) : DFA

5.4 Right Linear Grammar to Left Linear Grammar

Every right linear grammar can be represented by an equivalent left linear grammar. Conversion process involves drawing of an intermediate transition graph. Following steps are required :

1. Represent the right linear grammar using a transition graph. Mark the final state as ϵ

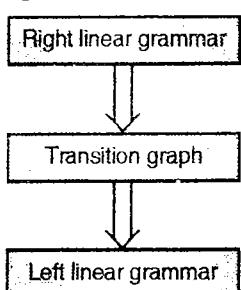


Fig. 5.4.1 : From right linear to left linear grammar

2. Interchange the start state and the final state.
3. Reverse the direction of all transitions.
4. Write left linear grammar from the transitions graph.

Example 5.4.1

SPPU - May 12, Aug. 15(In Sem), 4 Marks

Convert the following right linear grammar to an equivalent left-linear grammar.

$$\begin{array}{lll} S \rightarrow bB \mid b & B \rightarrow bC \\ B \rightarrow aB & C \rightarrow a & B \rightarrow b \end{array}$$

Solution :

Step 1 : Conversion of right linear grammar to transition system.

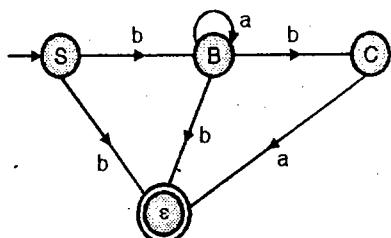


Fig. Ex. 5.4.1(a)

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get :

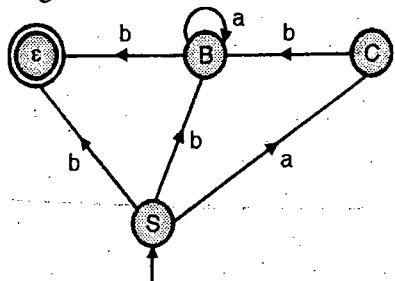


Fig. Ex. 5.4.1(b)

Step 3 : Writing of left linear grammar from the transition system, we get :

$$S \rightarrow b \mid Bb \mid Ca$$

$$B \rightarrow Ba \mid b$$

$$C \rightarrow Bb$$

Example 5.4.2

Write an equivalent left linear grammar from the given right linear grammar.

$$S \rightarrow 0A \mid 1B$$

$$A \rightarrow 0C \mid 1A \mid 0$$

$$B \rightarrow 1B \mid 1A \mid 1$$

$$C \rightarrow 0 \mid 0A$$

Solution :

Step 1 : Transition system for the given right linear grammar is as shown in Fig. Ex. 5.4.2(a).

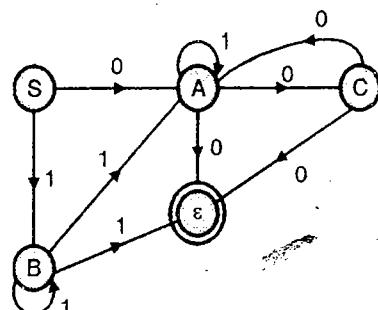


Fig. Ex. 5.4.2(a) : Transition graph

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get

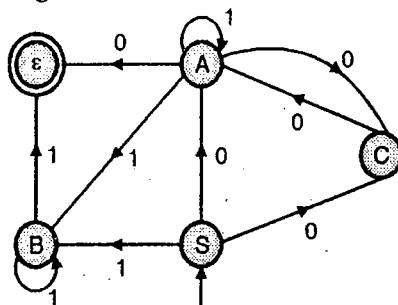


Fig. Ex. 5.4.2(b)

Step 3 : Writing of left linear grammar from the transition system, we get :

$$S \rightarrow C0 \mid A0 \mid B1$$

$$A \rightarrow A1 \mid C0 \mid B1 \mid 0$$

$$B \rightarrow B1 \mid 1$$

$$C \rightarrow A0.$$

Example 5.4.3

For right linear grammar given below, obtain an equivalent left linear grammar.

$$S \rightarrow 10A \mid 01$$

$$A \rightarrow 00A \mid 11$$

Solution :

Step 1 : Transition system from the given right linear grammar is as shown in Fig. Ex. 5.4.3(a) :

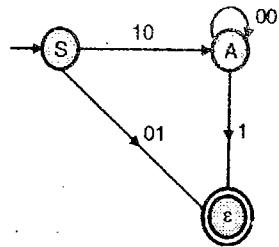


Fig. Ex. 5.4.3(a)

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get :

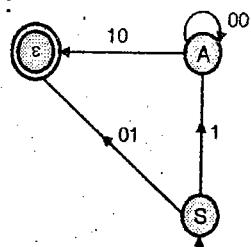


Fig. Ex. 5.4.3(b)

Step 3 : Writing of left linear grammar from the transition system, we get

$$S \rightarrow A1101$$

$$A \rightarrow A00110$$

Example 5.4.4

Convert the following right linear grammar to left linear grammar

$$S \rightarrow 0A \quad A \rightarrow 1A \quad A \rightarrow \epsilon$$

Solution :

Step 1 : From right linear grammar to transition system.

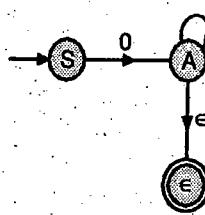


Fig. Ex. 5.4.4

Step 2 : Reversing direction of arrow and interchanging starting and the final state.

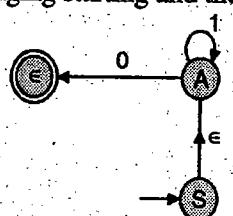


Fig. Ex. 5.4.4(a)

Step 3 : Writing of left linear grammar.

$$S \rightarrow A$$

$$A \rightarrow A110$$

5.5 Left Linear Grammar to Right Linear Grammar

Every left linear grammar can be represented by an equivalent right linear grammar. The conversion process involves drawing of an intermediate transition graph. Following steps are required.

1. Represent the left linear grammar using a transition graph. Mark the final state as
2. Interchange the start state and the final state.
3. Reverse the direction of all transitions.
4. Write right - linear grammar from the transition graph.

Fig. 5.5.1 : From left linear grammar to right linear grammar

Example 5.5.1

Write an equivalent right linear grammar from the given left-linear grammar.

$$S \rightarrow C0 | A0 | B1, \quad A \rightarrow A1 | C0 | B1 | 0$$

$$B \rightarrow B1 | 1, \quad C \rightarrow A0$$

Solution :

Step 1 : Transition system for the left-linear grammar is as shown in Fig. Ex. 5.5.1.

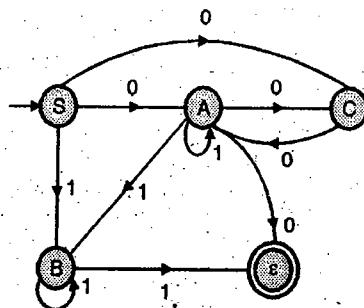


Fig. Ex. 5.5.1

Step 2 : Interchanging the start state and the final state and changing direction of all transitions, we get :

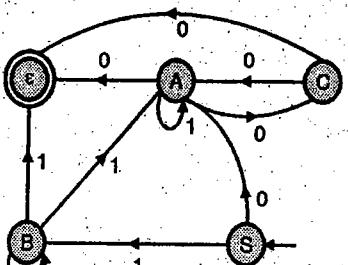


Fig. Ex. 5.5.1(a)



Step 3 : A right linear grammar can be written from the transition system. A set of productions is given below :

$$\begin{aligned} S &\rightarrow 1B \mid 0A \\ A &\rightarrow 0C \mid 1A \mid 0 \\ B &\rightarrow 1B \mid 1A \mid 1 \\ C &\rightarrow 0C \mid 0 \end{aligned}$$

Example 5.5.2

Construct the right linear grammar corresponding to the regular expression

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

Solution :

Step 1 : The given expression can be represented using a transition system as shown in Fig. Ex. 5.5.2.

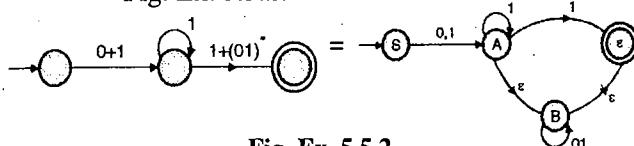


Fig. Ex. 5.5.2

Step 2 : Writing of right linear grammar from the transition system, we get :

$$S \rightarrow 0A \mid 1A, A \rightarrow 1A \mid 1 \mid B, B \rightarrow 01B \mid \epsilon$$

The ϵ -transition, $B \rightarrow \epsilon$, makes both A and B nullable. The ϵ -transitions, $B \rightarrow \epsilon$ is removed and the resulting productions are given below.

$$\begin{aligned} S &\rightarrow 0A \mid 1A \mid 0 \mid 1 \\ A &\rightarrow 1A \mid 1 \\ B &\rightarrow 01B \mid 0 \end{aligned}$$

Example 5.5.3

Write an equivalent right recursive grammar for the given left recursive grammar : $S \rightarrow S10 \mid 0$

Solution :

Step 1 : Transition system for the left-linear grammar is shown in Fig. Ex. 5.5.3(a) :

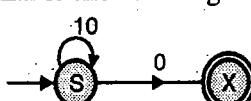


Fig. Ex. 5.5.3(a)

Step 2 : Interchanging the start state and the final state and changing direction of all transitions, we get :

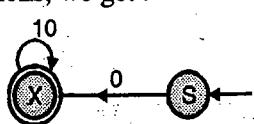


Fig. Ex. 5.5.3(b)

Step 3 : A right recursive grammar can be written from the transition system. A set of productions is given below :

$$\begin{aligned} S &\rightarrow 0X \mid 0 \\ X &\rightarrow 10X \mid 10 \end{aligned}$$

Example 5.5.4

Construct the right linear grammar corresponding to the regular expression. $R = (1 + (01)^*) 1^*(0 + 1)$

Solution : The R.E. $(1 + (01)^*) 1^*(0 + 1)$ can be represented using a transition system as shown in Fig. Ex. 5.5.4.

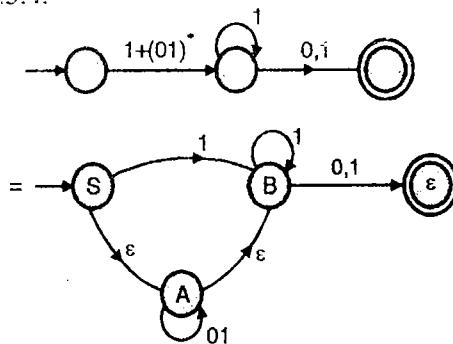


Fig. Ex. 5.5.4

Writing of right linear grammar from the transition system, we get :

$$\begin{aligned} S &\rightarrow 1BIA \\ A &\rightarrow 01AIB \\ B &\rightarrow 1B \mid 0C \mid 1C \mid 0I \mid 1 \end{aligned}$$

Removing unit productions $S \rightarrow A$ and $A \rightarrow B$:

$$\begin{aligned} S &\rightarrow 1B \mid 0C \mid 1C \mid 0I \mid 1 \\ A &\rightarrow 01A \mid 1B \mid 0C \mid 1C \mid 0I \mid 1 \\ B &\rightarrow 1B \mid 0C \mid 1C \mid 0I \mid 1 \end{aligned}$$

Example 5.5.5

Draw NFA accepting the language generated by grammar with productions :

$$\begin{aligned} S &\rightarrow abA \mid bB \mid aba \quad A \rightarrow b \mid aB \mid bA \\ B &\rightarrow aB \mid aA \end{aligned}$$

Solution :

Step 1 : Transitions system corresponding to $S \rightarrow abA \mid bB \mid aba$ is given by

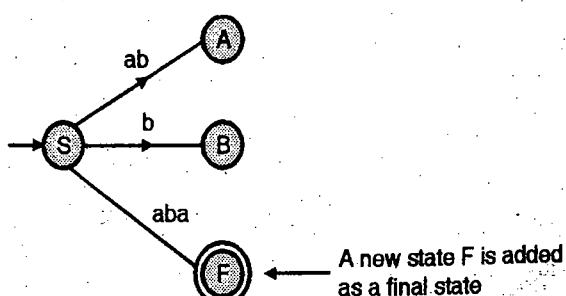


Fig. Ex. 5.5.5 : Transition graph

Step 2 : Transitions for $A \rightarrow b \mid aB \mid bA$ and $B \rightarrow aB \mid aA$ are added to the transition graph

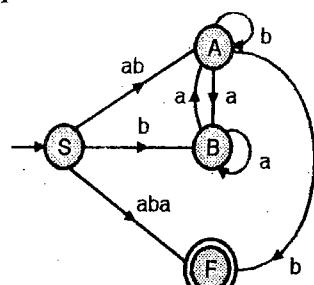


Fig. Ex. 5.5.5(a)

Step 3 : Transitions on ab and aba are expanded with resulting NFA as shown in Fig. Ex. 5.5.5(b).

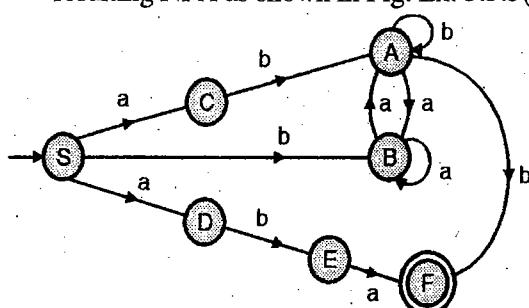


Fig. Ex. 5.5.5(b)

Example 5.5.6

Construct right linear and left linear grammar for the language $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$

Solution :

Step 1 : A DFA for $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$ is shown in Fig. Ex. 5.5.6.

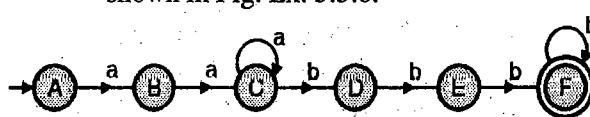


Fig. Ex. 5.5.6

Step 2 : Right linear grammar can be written from the DFA.

$$A \rightarrow aB$$

$$B \rightarrow aC$$

$$C \rightarrow aC \mid bD$$

$$D \rightarrow bE$$

$$E \rightarrow bF \mid b$$

$$F \rightarrow bF \mid b$$

Step 3 : Left linear grammar can be written after making the following modification in transition graph.

1. Interchange start state and the final state

2. Reverse direction of transitions.

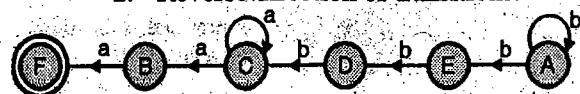


Fig. Ex. 5.5.6(a) : DFA after modifications

Step 4 : Left linear grammar can be written from the transition graph.

$$A \rightarrow Ab \mid Eb$$

$$E \rightarrow Db$$

$$D \rightarrow Cb$$

$$C \rightarrow Ca \mid Ba$$

$$B \rightarrow a$$

Example 5.5.7

Construct left linear and right linear grammar for the language $0^* (1(0+1))^*$

Solution :

Step 1 : We will first draw a transition diagram from the given regular expression.

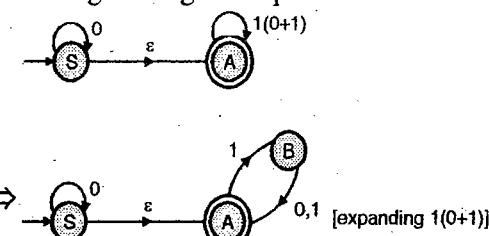


Fig. Ex. 5.5.7 : Transition graph for given RE

Step 2 : A right linear grammar can be written from the transition graph.

$$S \rightarrow 0S \mid A \mid \epsilon$$

$$A \rightarrow 1B$$

$$B \rightarrow 0A \mid 1A \mid 0 \mid 1$$

Removing the unit production, $S \rightarrow A$, the resulting grammar is given below :

$$S \rightarrow 0S \mid 1B \mid \epsilon$$

$$A \rightarrow 1B$$

$$B \rightarrow 0A \mid 1A \mid 0 \mid 1$$

Step 3 : Left linear grammar can be written after making the following modifications :

1. Interchange start state and the final state

2. Reverse direction of transitions.

The resulting transition diagram with above modifications is shown in Fig. Ex. 5.5.7(a).

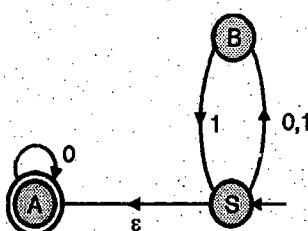


Fig. Ex. 5.5.7(a)

Step 4 : A left linear grammar can be written from the transition diagram

$$S \rightarrow B0 \mid B1 \mid A \mid \epsilon$$

$$B \rightarrow S1$$



$$A \rightarrow A010$$

Removing the unit production, $S \rightarrow A$, the resulting grammar is given below.

$$S \rightarrow B01B11A0101\epsilon$$

$$B \rightarrow S1$$

$$A \rightarrow A010$$

Example 5.5.8 SPPU - Dec. 12, 8 Marks

Construct left linear and right linear grammar for the language $(0+1)^*00(0+1)^*$

Solution :

Step 1 : We will first draw a transition diagram from the given regular expression.

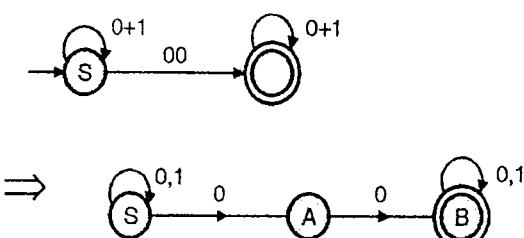


Fig. Ex. 5.5.8(a) : Transition graph for RE

Step 2 : Right linear grammar, from the transition graph can be written as given below :

$$S \rightarrow 0S11S10A$$

$$A \rightarrow 0B10$$

$$B \rightarrow 0B11B1011$$

Right linear grammar

Step 3 : Left linear grammar can be written after making the following modifications :

1. Interchange start state and the final state
2. Reverse direction of transitions.

Resulting transition diagram with above modifications is shown in Fig. Ex. 5.5.8(b) :

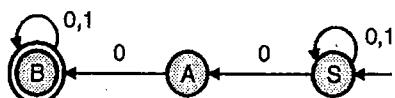


Fig. Ex. 5.5.8(b)

Step 4 : A left linear grammar can be written from the transition graph. It is given below :

$$S \rightarrow S01S11A0$$

$$A \rightarrow B010$$

$$B \rightarrow B01B1110$$

Example 5.5.9

Describe the language generated by the following grammar.

$$S \rightarrow bS1aA1\epsilon \quad A \rightarrow aA1bB1b$$

$$B \rightarrow bS$$

Solution :

A language can be understood easily from a regular expression. Thus the above problem should be solved as shown in Fig. Ex. 5.5.9.

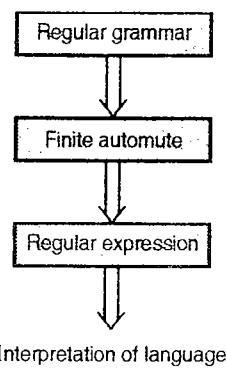


Fig. Ex. 5.5.9

Step 1 : We will first draw a transition diagram from the given regular grammar.

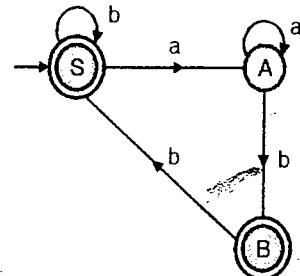


Fig. Ex. 5.5.9(a) : Transition graph

S is a final state, as $S \rightarrow \epsilon$

B is a final state, as $A \rightarrow bB1b$

Step 2 : A regular expression can be written easily by moving the effect of loop $S \rightarrow A \rightarrow B \rightarrow S$ on S. The resulting transition diagram is shown in Fig. Ex. 5.5.9(b).

$$\therefore R.E. = (a + aa^*bb)^*[\epsilon + aa^*b]$$

Language : A string of length 0 or more ending in either ab or b's but never containing aba as a substring.

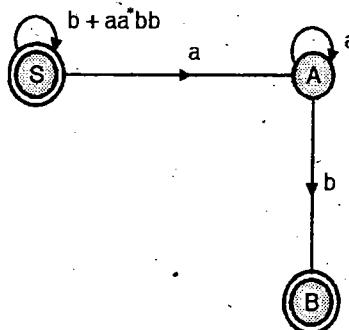


Fig. Ex. 5.5.9(b)

Example 5.5.10

Describe the language generated by the following grammar.

$$S \rightarrow aA1bC1b \quad A \rightarrow aS1bB$$

$$B \rightarrow aC1bA1a \quad C \rightarrow aB1bS$$

Solution : A language can be understood easily from a regular expression or finite automata.

Step 1 : We will first draw a transition diagram from the given regular grammar.

Language generated : From the transition diagram, it is clear that any path from S to C will involve :

1. Even number of a's
2. Odd number of b's

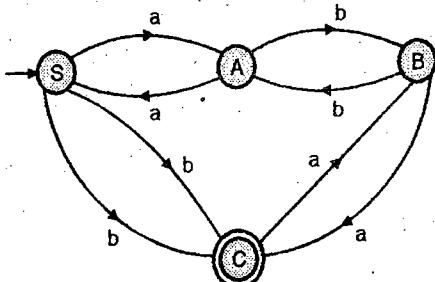


Fig. Ex. 5.5.10 : Transition graph for RE

Thus the language is,

$L = \{\omega \in \{a, b\}^* \mid \omega \text{ contains even number of } a's \text{ and odd number of } b's\}$.

Example 5.5.11

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) $S \rightarrow aSalsbSblaAblbAa$
 $A \rightarrow aAalbAbablble$

(2) $S \rightarrow bTlaTle$
 $T \rightarrow aSlbs$

Solution :

(1) It generates a string in which there is an 'a' and if it is rewritten as 'b' then the string will become a palindrome as shown in Fig. Ex. 5.5.11.

The string abaabbabba is in $L(G)$. If the third character 'a' is node b then the modified string abbabbabba is a palindrome.

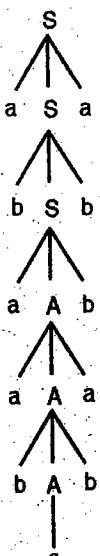


Fig. Ex. 5.5.11

(2) $S \rightarrow bTlaTle$

$A \rightarrow aslbs$

The FA representing the grammar is shown in Fig. Ex. 5.5.11(a).

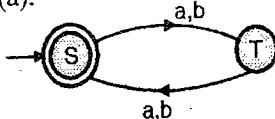


Fig. Ex. 5.5.11(a)

Thus the grammar generates the following language :

$$L(G) = \{\omega \in \{a, b\}^* \mid |\omega| \text{ is even}\}$$

Example 5.5.12

Find the CFL associated with CFG

$$S \rightarrow 0Q \mid 1P \quad P \rightarrow 0 \mid 1 \mid 0S \mid 1PP$$

$$Q \rightarrow 1 \mid 1S \mid 0QQ$$

Solution :

This is a case of indirect recursive. S is related back to itself through P or Q.

There are the following relations :

$$S \rightarrow 01S \mid 10S \mid 01 \mid 10 \mid 00QQ \mid 11PP$$

- Strings generated by S have equal number of 0's and 1's.

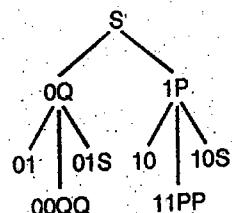


Fig. Ex. 5.5.12

- P generates those strings, where number of 0's is one more than number of 1's.

- Q generates those strings, where number of 1's is one more than number of 0's.

∴ CFL associated with the given CFG

$$= \{w \in \{0, 1\}^* \mid w \text{ contain equal number of 0's and 1's}\}.$$



CHAPTER

6

Pushdown Automata (PDA)

Syllabus

Push Down Automata : Introduction and Definition of PDA, Construction (Pictorial/ Transition diagram) of PDA, Instantaneous Description and ACCEPTANCE of CFL by empty stack and final state, Deterministic PDA Vs Nondeterministic PDA, Closure properties of CFLs, pumping lemma for CFL.

Post Machine- Definition and construction.

Syllabus Topic : Introduction to PDA, Construction (Pictorial/Transition Diagram) of PDA

6.1 Introduction to Pushdown Automata (PDA)

SPPU - Dec. 12

University Question

Q. Define PDA.

(Dec. 2012, 4 Marks)

Informally, pushdown automata can be viewed as finite automata with stack. An added stack provides memory and increases the capability of the machine.

A pushdown automata can do the followings :

1. Read input symbol [as in case of FA]
2. Perform stack operations.
 - 2.1 Push operation
 - 2.2 Pop operation
 - 2.3 Check empty condition of a stack through an initial stack symbol.
 - 2.4 Read top symbol of stack without a pop.
3. Make state changes.

PDA is more powerful than FA. A context-free language (CFL) can be recognized by a PDA. Only a subset of CFL that are regular can be recognized by finite automata.

- A context free language can be recognized by PDA.
- For every context-free language, there exists a PDA.
- The language of PDA is a context-free language.

Example : A string of the form $a^n b^n$ can not be handled by a finite automata. But the same can be handled by a PDA.

- Any machine recognizing a string of the form $a^n b^n$, must keep track of a's [first half of $a^n b^n$] as number of b's must be equal to the number of a's.
- First half of the string can be remembered through a stack.

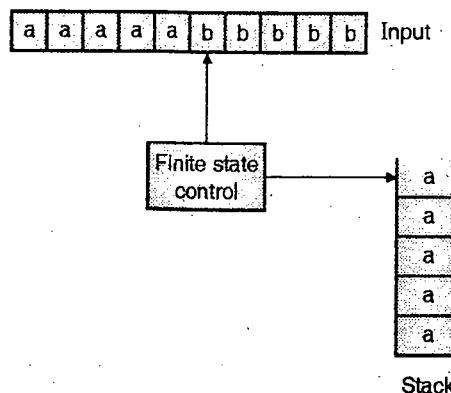


Fig. 6.1.1 : Stack after reading the first half of $a^5 b^5$

- As the machine reads the first half of $a^n b^n$, it remembers it by pushing it on top of the stack. As shown in Fig. 6.1.1, after reading first 5 a's, the stack contains 5 a's.
 - While reading the second half of the input string consisting of b's, the machine pops out an 'a' from the stack for every 'b' as input.
 - After reading 5 b's, input will finish and the stack will become empty. This will indicate that the input string is of the form $a^n b^n$.
 - The machine will have two states q_0 and q_1 . State q_0 – while the machine is reading a's State q_1 – While the machine is reading b's.
- While in state q_1 , an input 'a' is not allowed and hence there is a need for two states.

- A transition in PDA depends on :
 1. Current state
 2. Current input
 3. Top symbol of the stack.

A transition in PDA can be shown as a directed edge from the state q_i to q_j . While moving to state q_j , the machine can also perform stack operation. A transition edge from q_i to q_j should be marked with current input, current stack symbol (topmost symbol of the stack) and the stack operation. It is shown in Fig. 6.1.2.

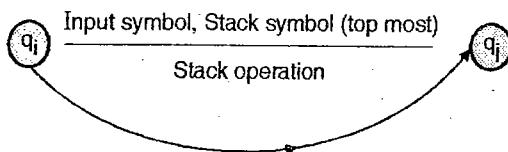


Fig. 6.1.2 : A transition from q_i to q_j

- A PDA uses three stack operations :
 - 1) **Pop** operation, it removes the top symbol from the stack.
 - 2) **Push** operation, it inserts a symbol onto the top of the stack.
 - 3) **Nop** operation, it does nothing to stack.
- The language $\{a^n b^n \mid n \geq 1\}$ can be accepted by the PDA of Fig. 6.1.3.

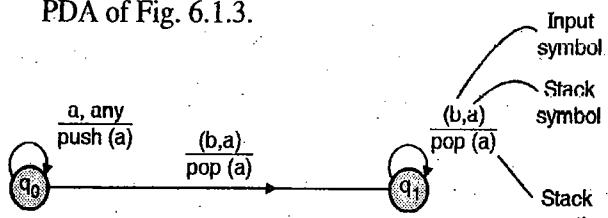


Fig. 6.1.3 : PDA for $a^n b^n$

- The state q_0 will keep track of the number of 'a's in an input string, by pushing symbol 'a' onto the stack for each input 'a'. A second state q_1 is used to pop an 'a' from the stack for each input symbol 'b'. Finally, after consuming the entire input the stack will become empty.

Syllabus Topic : Definition of PDA

6.2 The Formal Definition of PDA

SPPU - Dec. 12, Dec. 14, Dec. 15, May 16

University Questions

- Q.** Define PDA. (Dec. 2012, Dec. 2015, 4 Marks)
Q. Write formal definition of PDA. Explain its elements.
 What are different types of PDA?
 (Dec. 2014, May 2016, 6 Marks)

A pushdown automata M is defined as 7-tuple :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where, Q = The set of states

Σ = Input alphabet

Γ = Stack symbols

δ = The transition function is a transition form $Q \times (\Sigma \cup \epsilon) \times \Gamma$ to $Q \times \Gamma^*$

$q_0 = q_0 \in Q$ is the initial state

$F = F \subseteq Q$ is the set of final states

z_0 = An initial stack symbol

Transition function in detail

A transition function is a mapping from

$$Q \times (\Sigma \cup \epsilon) \times \Gamma \text{ to } Q \times \Gamma^*$$

Where

$Q \times (\Sigma \cup \epsilon) \times \Gamma$ implies that a transition is based on :

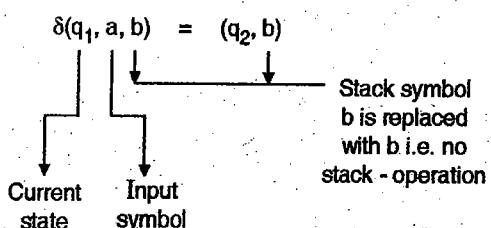
1. Current state
 2. Next input (including ϵ)
 3. Stack symbol (topmost)
- and

$Q \times \Gamma^*$ implies that :

1. The next state could be any state belonging to Q .
2. It can perform push, pop or no-operation (NOP) on stack.

In general, we can have the following types of transition behaviours.

1. Read input with no-operation on stack



- Left hand side of the transition, $\delta(q_1, a, b)$ implies :

q_1 is the current state,

a is the current input,

b is the stack symbol (topmost).

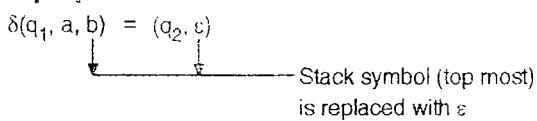
- Right hand side of the transition, (q_2, b) implies that

[The machine enters a new state q_2 . The top symbol of the stack, which is b is replaced with b .]

Thus, the transition $\delta(q_1, a, b) = (q_2, b)$ does not modify a stack.

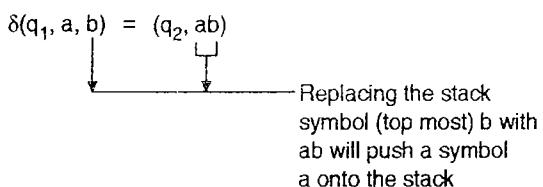


2. Pop operation



The above transition will erase the stack symbol (topmost). Replacing b with ϵ amounts to erasing b from the stack top.

3. Push operation



The above transition will perform a push operation. It will push ' a ' onto the stack. Replacing b with ab amounts to push ' a ' onto the stack.

The three operations for stack are shown in Table 6.2.1.

Table 6.2.1 : Transitions rules for stack operations

| Stack before transition (assumed) | Transition rule | Stack after transition |
|-----------------------------------|---------------------------------------|------------------------|
| | $\delta(q_1, a, b) = (q_2, b)$ | No - operation |
| | $\delta(q_1, a, b) = (q_2, \epsilon)$ | Pop - operation |
| | $\delta(q_1, a, b) = (q_2, ab)$ | Push - operation |

Example : Transition function for a PDA accepting a language

$L = \{\omega \in \{a, b\}^* \mid \omega \text{ is of the } a^n b^n \text{ with } n \geq 1\}$, can be given in various forms :

- Using a set of rules.
- Using graphical representation (transition diagram)

Using a set of equations

$\delta(q_0, a, z_0) = (q_0, az_0)$ [First a of $a^n b^n$ is pushed onto the stack]

$\delta(q_0, a, a) = (q_0, aa)$ [Subsequent a 's of $a^n b^n$ are pushed one by one onto the stack.]

$\delta(q_0, b, a) = (q_1, \epsilon)$ [On Seeing the first b , the machine will make a move to q_1 with a POP]

$\delta(q_1, b, a) = (q_1, \epsilon)$ [For each subsequent b , a POP operation is performed]

$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$ [After the end of input string, machine will transit to a final state]

Note : Failure transitions are not shown as it is a standard practice.

The transition rules mentioned above can be written in two different forms.

Form 1

$$\begin{aligned}\delta(q_0, a, z_0) &= (q_0, az_0) \Rightarrow ((q_0, az_0), (q_0, az_0)) \\ \delta(q_0, a, a) &= (q_0, aa) \Rightarrow ((q_0, a, a), (q_0, aa)) \\ \delta(q_0, b, a) &= (q_1, \epsilon) \Rightarrow ((q_0, b, a), (q_1, \epsilon)) \\ \delta(q_1, b, a) &= (q_1, \epsilon) \Rightarrow ((q_1, b, a), (q_1, \epsilon)) \\ \delta(q_1, \epsilon, z_0) &= (q_2, z_0) \Rightarrow ((q_1, \epsilon, z_0), (q_2, z_0))\end{aligned}$$

Form 2

Using graphical representation (transition diagram)

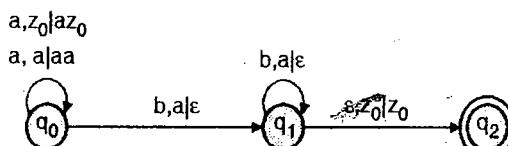


Fig. 6.2.1 : Transition diagram

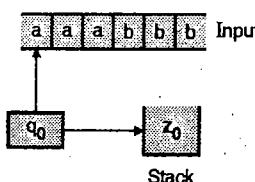
Syllabus Topic : Instantaneous Description

6.3 Instantaneous Description of a PDA

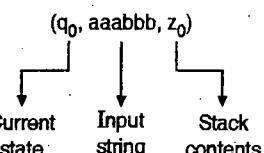
Instantaneous behaviour is useful in showing the processing of a string by PDA. Processing of the input string $a^3 b^3$ by the PDA of Fig. 6.2.1 is shown in Fig. 6.3.1.

Graphical representation (instantaneous behaviour)

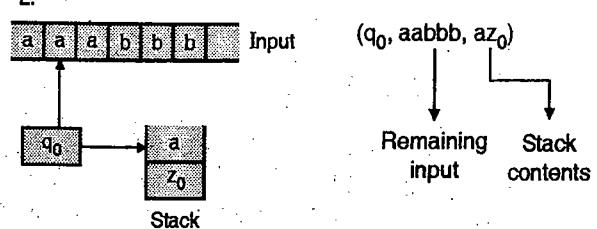
1.

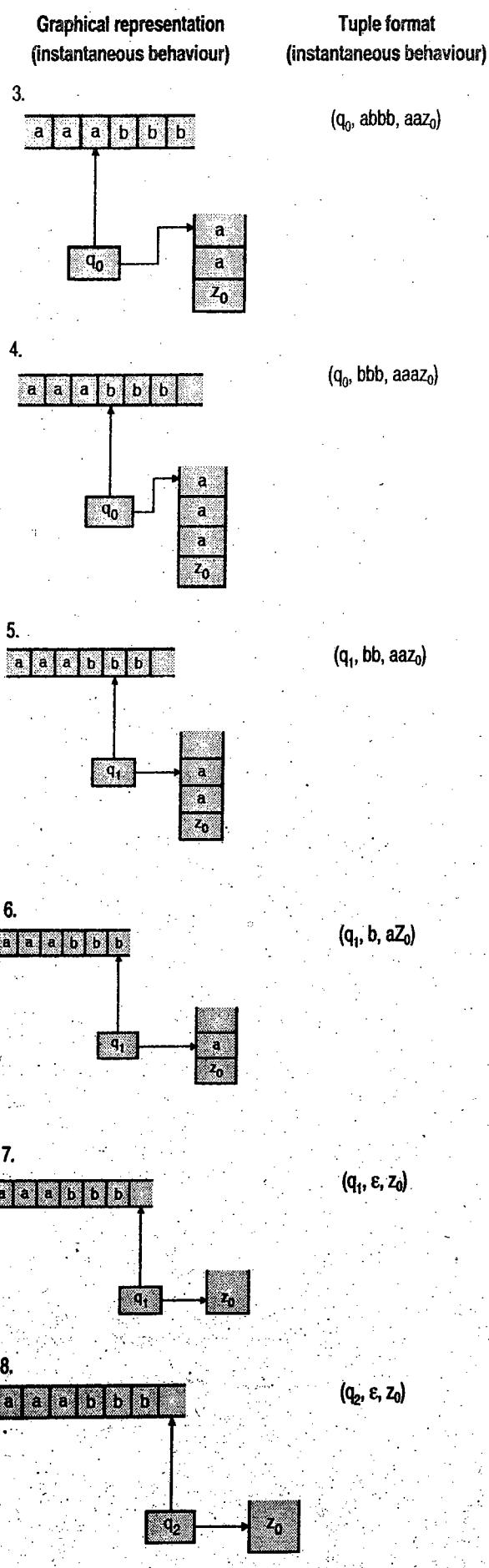


Tuple format (instantaneous behaviour)

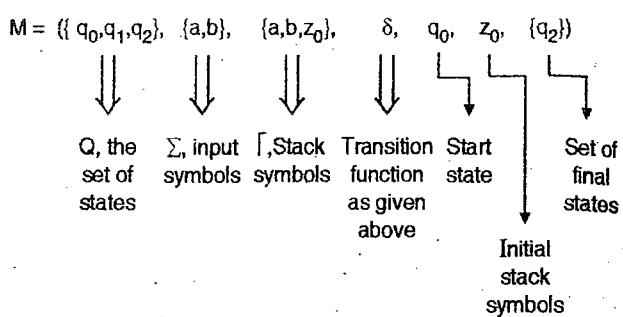


2.



Fig. 6.3.1 : Processing of a^3b^3 by the PDA of Fig. 6.2.1

The PDA for recognizing a string of the form $a^n b^n$ can be described as :



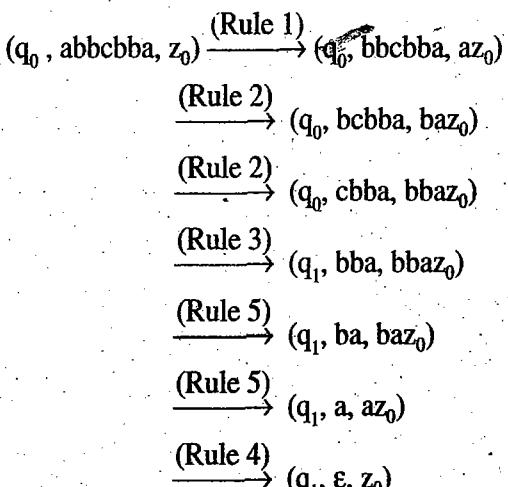
Example 6.3.1

Suppose the PDA $M = (\{q_0, q_1\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_1\})$ has the following transition function.

1. $\delta(q_0, a, \epsilon) = (q_0, a)$
2. $\delta(q_0, b, \epsilon) = (q_0, b)$
3. $\delta(q_0, c, \epsilon) = (q_1, \epsilon)$
4. $\delta(q_1, a, a) = (q_1, \epsilon)$
5. $\delta(q_1, b, b) = (q_1, \epsilon)$

Show the acceptance of $abbcbba$ by the above PDA through an instantaneous description.

Solution :



Since, the machine is in a final state q_1 , the string $abbcbba$ is accepted.

6.4 The Language of a PDA

A language L can be accepted by a PDA in two ways :

1. Through final state.
2. Through empty stack.

It is possible to convert between the two classes.

1. From final state to empty stack.
2. From empty stack to final state.

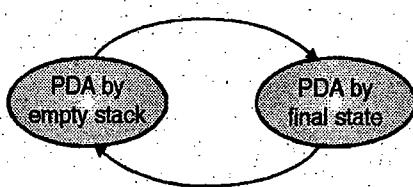


Fig. 6.4.1 : Equivalence of two PDAs

**Syllabus Topic : Acceptance of CFL by Final State****6.4.1 Acceptance by Final State**

SPPU - Dec. 16

University Question**Q.** Define PDA through final state.

(Dec. 2016, 2 Marks)

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ then the language accepted by M through a final state is given by :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \xrightarrow[M]{*} (q_1, \epsilon, \alpha) \right\}$$

Where the state $q_1 \in F$. α , the final contents of the stack are irrelevant as a string is accepted through a final state.

Syllabus Topic : Acceptance of CFL by Empty Stack**6.4.2 Acceptance by Empty Stack**

SPPU - Dec. 16

University Question**Q.** Define PDA through empty stack.

(Dec. 2016, 2 Marks)

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$ then the language accepted through an empty stack is given by :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \xrightarrow[M]{*} (q_1, \epsilon, \epsilon) \right\}$$

Where q_1 is any state belonging to Q and the stack becomes empty on application of input string w .

Example 6.4.1

Give a PDA to accept the language

$$L = \{0^n 1^m \mid n \leq m\}$$

1. Through empty stack.
2. Through final state.

Solution :**Algorithm**

1. Sequence of 0's should be pushed onto the stack in state q_0 .

$$\delta(q_0, 0, z_0) = (q_0, 0z_0) \text{ [Push the first 0]}$$

$$\delta(q_0, 0, 0) = (q_0, 00) \text{ [Push subsequent 0's]}$$

2. A '0' should be popped for every 1 as input till the stack becomes empty.

$$\delta(q_0, 1, 0) = (q_1, \epsilon) \text{ [Pop on first 1 and change the state to } q_1]$$

$\delta(q_1, 1, 0) = (q_1, \epsilon) \text{ [Pop on subsequent 1 as input till every 0 is erased from the stack]}$

3. Subsequent 1's ($m - n$) will have no effect on the stack.

$$\delta(q_1, 1, z_0) = (q_1, z_0)$$

4. Finally, the symbol Z_0 should be popped out to make the stack empty.

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

This step is required if the language is to be accepted through an empty stack.

Transition behaviour of the PDA is shown in Fig. Ex. 6.4.1(a to d)

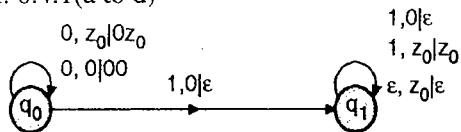


Fig. Ex. 6.4.1(a) : Transition diagram for acceptance through an empty stack

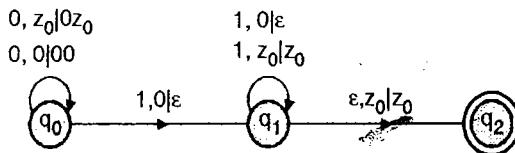


Fig. Ex. 6.4.1(b) : Transition diagram for acceptance through a final state

1. $\delta(q_0, 0, z_0) = (q_0, 0z_0)$
2. $\delta(q_0, 0, 0) = (q_0, 00)$
3. $\delta(q_0, 1, 0) = (q_1, \epsilon)$
4. $\delta(q_1, 1, 0) = (q_1, \epsilon)$
5. $\delta(q_1, 1, z_0) = (q_1, z_0)$
6. $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Fig. Ex. 6.4.1(c) : State transition rules for acceptance through an empty stack

1. $\delta(q_0, 0, z_0) = (q_0, 0z_0)$
2. $\delta(q_0, 0, 0) = (q_0, 00)$
3. $\delta(q_0, 1, 0) = (q_1, \epsilon)$
4. $\delta(q_1, 1, 0) = (q_1, \epsilon)$
5. $\delta(q_1, 1, z_0) = (q_1, z_0)$
6. $\delta(q_1, \epsilon, z_0) = (q_2, z_0)$

Fig. Ex. 6.4.1(d) : State transition rules for acceptance through a final state

The PDA accepting through an empty stack is given by :

$$M = ((q_0, q_1), \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \phi)$$



Where δ is given in Fig. Ex. 6.4.1(c).

The PDA accepting through final state is given by:

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Where δ is given in Fig. Ex. 6.4.1(d).

Example : Processing of string 00111 by the PDA

Case I : Acceptance through empty stack.

$$\begin{aligned} (q_0, 00111, z_0) &\xrightarrow{\text{(Rule 1)}} (q_0, 0111, 0z_0) \\ &\xrightarrow{\text{(Rule 2)}} (q_0, 111, 00z_0) \\ &\xrightarrow{\text{(Rule 3)}} (q_1, 11, 0z_0) \\ &\xrightarrow{\text{(Rule 4)}} (q_1, 1, z_0) \\ &\xrightarrow{\text{(Rule 5)}} (q_1, \epsilon, z_0) \\ &\xrightarrow{\text{(Rule 6)}} (q_1, \epsilon, \epsilon) \end{aligned}$$

Case II : Acceptance through final state :

$$\begin{aligned} (q_0, 00111, z_0) &\xrightarrow{\text{(Rule 1)}} (q_0, 0111, 0z_0) \\ &\xrightarrow{\text{(Rule 2)}} (q_0, 111, 00z_0) \\ &\xrightarrow{\text{(Rule 3)}} (q_1, 11, 0z_0) \\ &\xrightarrow{\text{(Rule 4)}} (q_1, 1, z_0) \\ &\xrightarrow{\text{(Rule 5)}} (q_1, \epsilon, z_0) \\ &\xrightarrow{\text{(Rule 6)}} (q_2, \epsilon, z_0) \end{aligned}$$

Example 6.4.2

Let $L = \{a^m b^n \mid n < m\}$

Construct

(i) PDA accepting L by empty store.

(ii) PDA accepting L by final state.

Solution :

(i) Algorithm (Accepting L by empty store)

- Sequence of a 's should be pushed onto the stack in state q_0 .

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

- An 'a' should be popped for every b as input till the end of input.

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

- Additional $m - n$ a's should be popped from the stack.

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

- Finally, the symbol z_0 should be popped out to make the state empty.

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

Transition diagram and the transition table are given below.

Table Ex. 6.4.2

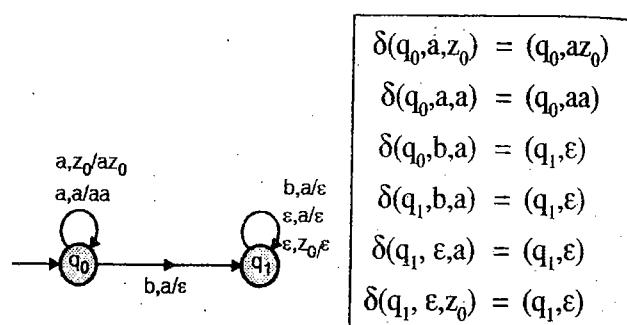


Fig. Ex. 6.4.2

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \emptyset)$$

(ii) Accepting L through final state

Following modifications are required to accept through a final state.

In step 3, when the input ends with stack containing $m - n$ a's, the machine should enter the final state.

Table Ex. 6.4.2(a)

| |
|--|
| $\delta(q_0, a, z_0) = (q_0, az_0)$ |
| $\delta(q_0, a, a) = (q_0, aa)$ |
| $\delta(q_0, b, a) = (q_1, \epsilon)$ |
| $\delta(q_1, b, a) = (q_1, \epsilon)$ |
| $\delta(q_1, \epsilon, a) = (q_2, \epsilon)$ |

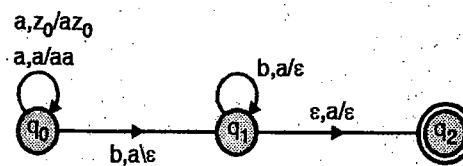


Fig. Ex. 6.4.2(a)

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Example 6.4.3 SPPU - Dec. 13, 8 Marks

Design a pushdown automata for the following language

$$L = \{a^n b^{2n} : n > 0\}$$

Solution : Algorithm

- Sequence of a 's should be pushed onto the stack.
- An 'a' should be popped for every pair of b 's.

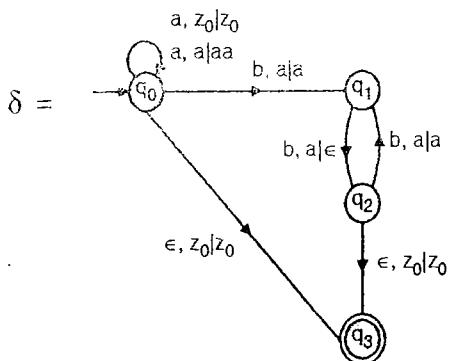


Fig. Ex. 6.4.3

The PDA accepting through the final state is given by,

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \{q_3\})$$

Example 6.4.4

Design a PDA for accepting the set of all strings over $\{a, b\}$ with an equal number of a's and b's. The string should be accepted both by

(1) Final state

(2) Empty stack.

Solution :

- Stack will be used to store excess of a's over b's or excess of b's over a's out of input seen so far.
- The status of stack on input abaabbbaa is shown in the Fig. Ex. 6.4.4.

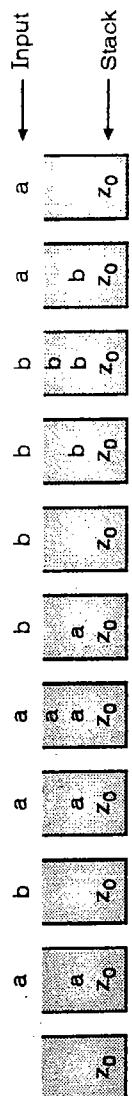
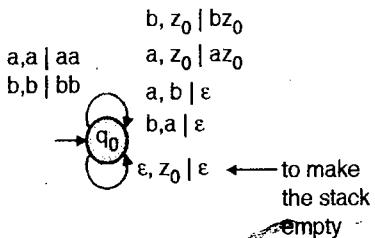


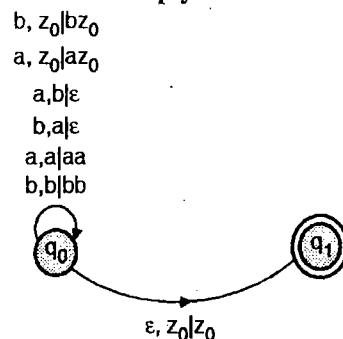
Fig. Ex. 6.4.4 : Stack preserving excess of a's over b's or b's over a's

3. Stack contains a(topmost) and the input is a.
 $\delta(q_0, a, a) = (q_0, aa)$ [Excess a's will increase by 1]
4. Stack contains a(topmost) and the input is b
 $\delta(q_0, b, a) = (q_0, \epsilon)$ [Excess a's will decrease by 1]
5. Stack contains b(topmost) and the input is a.
 $\delta(q_0, a, b) = (q_0, \epsilon)$ [Excess b's will decrease by 1]
6. Stack contains b(topmost) and the input is b.
 $\delta(q_0, b, b) = (q_0, bb)$ [Excess b's will increase by 1]

The PDA is shown in Fig. Ex. 6.4.4(a, b).



(a) Transition diagram for the PDA accepting through an empty stack



(b) Transition diagram for the PDA accepting through a final state

Fig. Ex. 6.4.4

- $$\begin{aligned}\delta(q_0, a, z_0) &= (q_0, az_0) \\ \delta(q_0, b, z_0) &= (q_0, bz_0) \\ \delta(q_0, a, b) &= (q_0, \epsilon) \\ \delta(q_0, b, a) &= (q_0, \epsilon) \\ \delta(q_0, a, a) &= (q_0, aa) \\ \delta(q_0, b, b) &= (q_0, bb) \\ \delta(q_0, \epsilon, z_0) &= (q_0, \epsilon)\end{aligned}$$

Fig. Ex. 6.4.4(c) : Transition rules for the PDA accepting through empty stack



$$\begin{aligned}
 \delta(q_0, a, z_0) &= (q_0, az_0) \\
 \delta(q_0, b, z_0) &= (q_0, bz_0) \\
 \delta(q_0, a, b) &= (q_0, \epsilon) \\
 \delta(q_0, b, a) &= (q_0, \epsilon) \\
 \delta(q_0, a, a) &= (q_0, aa) \\
 \delta(q_0, b, b) &= (q_0, bb) \\
 \delta(q_0, \epsilon, z_0) &= (q_1, z_0)
 \end{aligned}$$

Fig. Ex. 6.4.4(d) : Transition rules for the PDA accepting through final state

Fig. Ex. 6.4.4 (a to d) shows state transition behaviour of PDA of Example 6.4.4.

The PDA accepting through empty stack is given by :

$$M = (\{q_0\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \emptyset)$$

Where, δ is given in Fig. Ex. 6.4.4(c).

The PDA accepting through final state is given by :

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\}),$$

Where, δ is given in Fig. Ex. 6.4.4(d).

Example 6.4.5

Construct pushdown automata for the following language :

$L = \{\text{the set of strings over alphabet } \{a, b\} \text{ with exactly twice as many } a's \text{ as } b's\}$

Solution :

To solve this example, we can take three stack symbols :

$$\Gamma = (x, y, z_0)$$

Where, x stands for 2 a 's,

y stands for a ' b ',

z_0 is initial stack symbol.

- Stack will be used to store excess of x 's over y 's or excess of y 's over x 's.
- Since, a ' x ' will be pushed onto the stack after 2 a 's, after first ' a ' the machine will transit to q_1 to remember that one ' a ' has already come and the second ' a ' in q_1 will complete 2 a 's.
- Status of the stack and the state of the machine is shown in Fig. Ex. 6.4.5. Input applied to the machine is abaabbaaaaabaab.

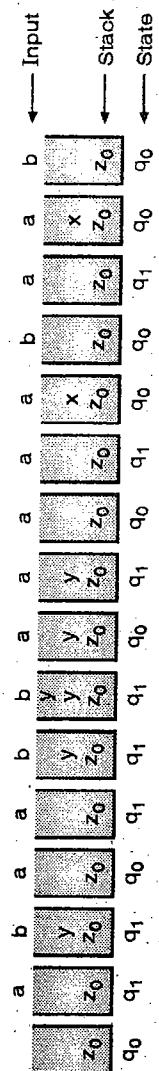


Fig. Ex. 6.4.5 : Status of stack for input abaabbaaaaabaab

Implementation of PDA using final state :

Let the PDA $M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

Where, $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{x, y, z_0\}$

q_0 is initial state, z_0 is initial stack symbol.

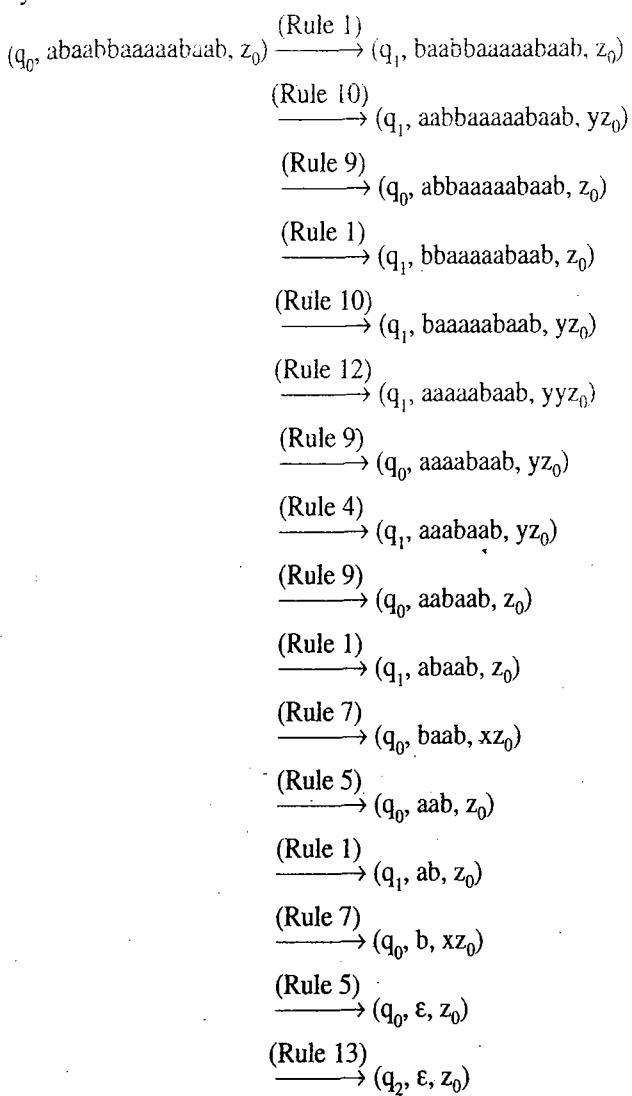
$$F = \{q_2\}$$

Transition function δ is given by :

1. $\delta(q_0, a, z_0) = (q_1, z_0)$ [First input could be either a or b]
2. $\delta(q_0, b, z_0) = (q_0, yz_0)$
3. $\delta(q_0, a, x) = (q_1, x)$ [goes to q_1 to remember first a of aa]
4. $\delta(q_0, a, y) = (q_1, y)$ [goes to q_1 to remember first a of 2 a 's]
5. $\delta(q_0, b, x) = (q_0, \epsilon)$ [Excess x is removed]
6. $\delta(q_0, b, y) = (q_0, yy)$ [Excess y increases by 1]
7. $\delta(q_1, a, z_0) = (q_0, xz_0)$ [Push x for 2 a 's]
8. $\delta(q_1, a, x) = (q_0, xx)$ [Push x for 2 a 's]
9. $\delta(q_1, a, y) = (q_0, \epsilon)$ [Two a 's will remove a ' y ']
10. $\delta(q_1, b, z_0) = (q_1, yz_0)$ [Excess y is saved]
11. $\delta(q_1, b, x) = (q_1, \epsilon)$ [Excess x is removed]
12. $\delta(q_1, b, y) = (q_1, yy)$ [Excess y increases by 1]
13. $\delta(q_0, \epsilon, z_0) = (q_2, z_0)$ [goes to final state to accept the string]



Example : Processing of string abaabbaaaaabaab by the PDA



Example 6.4.6

Give the transition table for PDA recognizing the following language

$$L = \{ a^n b^n c^m d^m \mid n, m \geq 1 \}$$

Solution :

Algorithm

- Sequence of initial a's should be pushed onto the stack in state q_0 .
- For every symbol in x, an 'a' should be erased from the stack.
 1. Push the first 'a' using the transition $\delta(q_0, a, z_0) = (q_0, az_0)$
 2. Push subsequent a's using the transition $\delta(q_0, a, a) = (q_0, aa)$.
 3. On the first 'b' as input, the machine will transit to q_1 with a pop operation using the transition $\delta(q_0, b, a) = (q_1, \epsilon)$.

4. On subsequent a's or b's in x, an 'a' should be erased from the stack.

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

5. After the end of input string, contents of stack should be erased one by one.

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

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

The PDA is given by :

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \emptyset)$$

Where, δ is given below :

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

$$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon) \quad [\text{To accept a null string}]$$

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

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

erase, everything from the stack

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

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

Example 6.4.7

Let $L = \{ a^n b^n c^m d^m \mid n, m \geq 1 \}$ find a PDA that accepts L.

Solution :

Algorithm

1. Sequence of a's should be pushed onto the stack.
2. For every b as input, an 'a' should be erased from the stack.
3. Sequence of c's should be pushed onto the stack.
4. For every d as input, a 'c' should be erased from the stack.

The PDA is given by :

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c, d\}, \{a, c, z_0\}, \delta, q_0, z_0, \emptyset)$$

Where the transition function δ is given below :

$$\delta(q_0, a, z_0) = (q_0, az_0) \quad [\text{Push the first a}]$$

$$\delta(q_0, a, a) = (q_0, aa) \quad [\text{Push remaining a's}]$$

$$\delta(q_0, b, a) = (q_1, \epsilon) \quad [\text{Erase an 'a' on first b}]$$

$$\delta(q_1, b, a) = (q_1, \epsilon) \quad [\text{Erase remaining a's on subsequent b's}]$$

$$\delta(q_1, c, z_0) = (q_2, cz_0) \quad [\text{First c is pushed}]$$

$$\delta(q_2, c, c) = (q_2, cc) \quad [\text{Subsequent c's are pushed}]$$

$$\delta(q_2, d, c) = (q_3, \epsilon) \quad [\text{On first d, machine transits to } q_3 \text{ with a pop}]$$

$$\delta(q_3, d, c) = (q_3, \epsilon) \quad [\text{For every d, a 'c' is erased}]$$

$$\delta(q_3, \epsilon, z_0) = (q_3, \epsilon) \quad [\text{String is accepted through empty stack}]$$

**Example 6.4.8 SPPU - May 15, 6 Marks**

Let $L = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i + j = k \}$

- Find a PDA accepting through final state.
- Find a PDA accepting through empty stack.

Solution :

Algorithm

- For every input symbol 'a', a symbol x is pushed onto the stack.
- For every input symbol 'b', a symbol x is pushed onto the stack.
- For every input symbol 'c', x is erased from the stack.

(i) PDA accepting through final state is given by

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \{x, z_0\}, q_0, z_0, \delta, \{q_3\})$$

Where the transition function δ is :

- $\delta(q_0, a, z_0) = (q_0, xz_0)$ [x is pushed for the first a]
- $\delta(q_0, a, x) = (q_0, xx)$ [x is pushed for every subsequent a]
- $\delta(q_0, b, z_0) = (q_1, xz_0)$ [First b without a's]
- $\delta(q_0, b, x) = (q_1, xx)$ [First b after a's]
- $\delta(q_1, b, x) = (q_1, xx)$ [Subsequent b's]
- $\delta(q_1, c, x) = (q_2, \epsilon)$ [x is erased for the first c]
- $\delta(q_2, c, x) = (q_2, \epsilon)$ [x is erased for subsequent c's]
- $\delta(q_2, \epsilon, z_0) = (q_3, z_0)$ [Accept through q_3]
- $\delta(q_0, \epsilon, z_0) = (q_3, z_0)$ [Accept a null string]

(ii) PDA accepting through empty stack is given by,

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{x, z_0\}, q_0, z_0, \delta, \emptyset)$$

where the transition function δ is :

- $\delta(q_0, a, z_0) = (q_0, xz_0)$
- $\delta(q_0, a, x) = (q_0, xx)$
- $\delta(q_0, b, z_0) = (q_1, xz_0)$
- $\delta(q_0, b, x) = (q_1, xx)$
- $\delta(q_1, b, x) = (q_1, xx)$
- $\delta(q_1, c, x) = (q_2, \epsilon)$
- $\delta(q_2, c, x) = (q_2, \epsilon)$
- $\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$ [Stack is made empty]
- $\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$ [Accept a null string]

Example 6.4.9 SPPU - Dec. 14, Dec. 16, 8 Marks

Construct a PDA accepting $\{ a^n b^m a^n \mid m, n \geq 1 \}$ by null store.

Solution : Algorithm

- The sequence of a's should be pushed onto the stack in state q_0 .

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

- On first b, the machine moves to q_1 and remains there for b's. b's will have no effect on the stack.
- For every 'a', an 'a' is erased from the stack.

The PDA accepting through empty stack is given by

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \emptyset)$$

Where the transition function δ is :

- $\delta(q_0, a, z_0) = (q_0, az_0)$ [First 'a' is pushed]
- $\delta(q_0, a, a) = (q_0, aa)$ [Subsequent a's are pushed]
- $\delta(q_0, b, a) = (q_1, a)$ [Input symbols b's are skipped]
- $\delta(q_1, b, a) = (q_1, a)$
- $\delta(q_1, a, a) = (q_2, \epsilon)$ [An a is erased on first a of last a's]
- $\delta(q_2, a, a) = (q_2, \epsilon)$ [An a is erased on subsequent a's of last a's]
- $\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$ [Accepting through empty stack]

Example 6.4.10 SPPU - May 16, 8 Marks

Design a PDA for accepting a language

$$L = \{ WcW^T \mid W \in \{a, b\}^* \}$$

Solution :

W^T stands for reverse of W. A string of the form WcW^T is an odd length palindrome with the middle character as c.

Algorithm

If the length of the string is $2n + 1$, then the first n symbols should be matched with the last n symbols in the reverse order. A stack can be used to reverse the first n input symbols.

- Status of the stack and state of the machine is shown in Fig. Ex. 6.4.10. Input applied is abbcbbba.

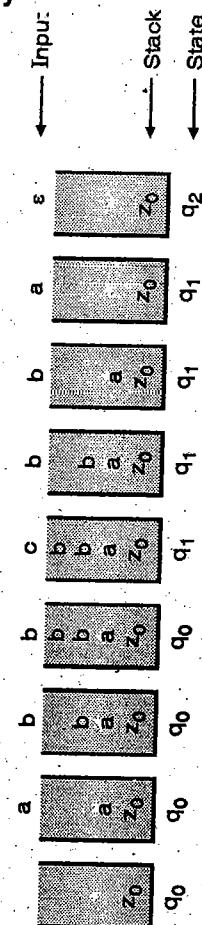


Fig. Ex. 6.4.10 : A PDA on input abbcbbba



The PDA accepting through final state is given by

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Where the transition function δ is given below :

1. $\delta(q_0, a, \epsilon) = (q_0, a)$ First n symbols are pushed onto the stack
2. $\delta(q_0, b, \epsilon) = (q_0, b)$
3. $\delta(q_0, c, \epsilon) = (q_1, \epsilon)$ [State changes on c]
4. $\delta(q_1, a, a) = (q_1, \epsilon)$ Last n symbols are matched with first n symbols in reverse order
5. $\delta(q_1, b, b) = (q_1, \epsilon)$
6. $\delta(q_1, \epsilon, z_0) = (q_2, z_0)$ [Accepted through final state]

A transition of the form $\delta(q_0, a, \epsilon) = (q_0, a)$ implies that always push a, irrespective of stack symbol.

Example 6.4.11

Design a PDA to check whether a given string over $\{a, b\}$ ends in abb.

Solution : Strings ending in abb form a regular language. A regular language is accepted by DFA.

A PDA for a regular language can be constructed in two steps :

1. Design a DFA.
2. Convert DFA to PDA by appending no-stack operation to every transition in DFA.

Step 1 : DFA for the given language is :

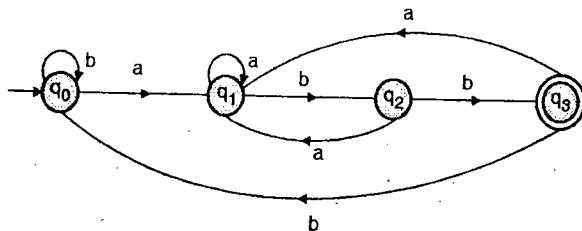


Fig. Ex. 6.4.11 : A DFA for string ending in abb

Step 2 : From DFA to PDA

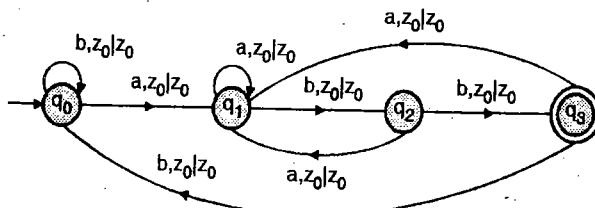


Fig. Ex. 6.4.11(a) : A PDA constructed from DFA

- In every transition of DFA, a stack operation z_0/z_0 is added. z_0/z_0 implies no-stack operation.

The PDA accepting strings ending in abb through final state is given by :

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{z_0\}, \delta, q_0, z_0, \{q_3\})$$

Where the transition function δ is given below,

$$\delta(q_0, b, z_0) = (q_0, z_0)$$

$$\delta(q_0, a, z_0) = (q_1, z_0)$$

$$\delta(q_1, a, z_0) = (q_1, z_0)$$

$$\delta(q_1, b, z_0) = (q_2, z_0)$$

$$\delta(q_2, a, z_0) = (q_1, z_0)$$

$$\delta(q_2, b, z_0) = (q_3, z_0)$$

$$\delta(q_3, a, z_0) = (q_1, z_0)$$

$$\delta(q_3, b, z_0) = (q_0, z_0)$$

Example 6.4.12 SPPU - Dec. 15, 9 Marks

Design a PDA which accepts only odd number of a's over $\Sigma = \{a, b\}$.

Simulate PDA for the string "aabab".

Solution : It is a regular language. DFA for the given language is,

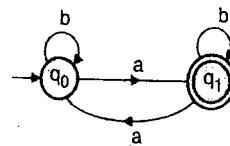


Fig. Ex. 6.4.12

We can draw an equivalent PDA from the given DFA.

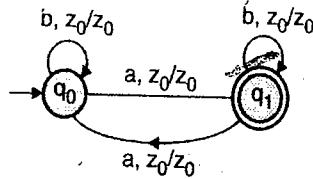


Fig. Ex. 6.4.12(a)

The PDA for accepting strings containing odd number of a's is given by,

$$M = (\{q_0, q_1\}, \{a, b\}, \{z_0\}, \delta, z_0, z_0, \{q_1\})$$

Where the transition function δ is given below.

$$1. \delta(q_0, b, z_0) = (q_0, z_0)$$

$$2. \delta(q_1, a, z_0) = (q_1, z_0)$$

$$3. \delta(q_1, a, z_0) = (q_0, z_0)$$

$$4. \delta(q_0, b, z_0) = (q_1, z_0)$$

Simulating PDA for the string "aabab".

$$\delta(q_0, aabab, z_0) \xrightarrow{\text{Rule 2}} (q_1, ab, z_0)$$

$$\xrightarrow{\text{Rule 3}} (q_0, bab, z_0)$$

$$\xrightarrow{\text{Rule 1}} (q_0, ab, z_0)$$

$$\xrightarrow{\text{Rule 2}} (q_1, b, z_0)$$

$$\xrightarrow{\text{Rule 4}} (q_1, \epsilon, z_0)$$

String is accepted through the final state q_1 .

Example 6.4.13 SPPU - Dec. 14, 8 Marks

Construct a PDA that accepts the following language using CNF.

$$L = \{a^{2n} \mid n > 0\}$$



Solution : A string of the form $a^n b^n$ becomes a^{2n} if b is equal to a.

We can generate a string of the form $a^n B^n$ and when the variable B is replaced by a, the $a^n B^n$ will become a^{2n} .

∴ The grammar for $a^{2n} \mid n > 0$ can be written as,

$$S \rightarrow aSB \mid aB$$

$$B \rightarrow a$$

The PDA for the above CFG is given by :

$$\delta(q, \epsilon, S) \Rightarrow \{(q, aSB), (a, B)\}$$

$$\delta(q, \epsilon, B) \Rightarrow (q, a)$$

$$\delta(q, a, a) = (q, \epsilon)$$

Example 6.4.14 SPPU - Dec. 14, 8 Marks

Design a PDA accepting $\{a^n b^m a^n \mid m, n \geq 1\}$. Simulate a PDA for the string "aabaa".

Solution :

PDA for the string "aabaa"

$$\begin{aligned}\delta(q_0, aabaa, z_0) &= (q_0, abaa, a z_0) && [\text{Rule 1}] \\ \delta(q_0, abaa, a z_0) &= (q_0, baa, aa z_0) && [\text{Rule 2}] \\ \delta(q_0, baa, aa z_0) &= (q_1, aa, aa z_0) && [\text{Rule 3}] \\ \delta(q_1, aa, aa z_0) &= (q_2, a, a z_0) && [\text{Rule 5}] \\ \delta(q_2, a, a z_0) &= (q_2, \epsilon, z_0) && [\text{Rule 6}] \\ \delta(q_2, \epsilon, z_0) &= (q_2, \epsilon, \epsilon) && [\text{Rule 7}]\end{aligned}$$



Accept through the empty stack.

6.5 Non-deterministic PDA (NPDA)

SPPU - May 12

University Question

Q. Explain with example NPDA.

(May 2012, 4 Marks)

There are two types of push down automata :

1. DPDA (Deterministic PDA)
2. NPDA (Non-deterministic PDA)

A NPDA provides non-determinism to PDA.

In a DPDA there is only one move in every situation. Where as, in case of NPDA there could be multiple moves under a situation.

- DPDA is less powerful than NPDA. Every context free language can not be recognized by a DPDA but it can be recognized by NPDA. The class of language a DPDA can accept lies in between a regular language and CFL. A palindrome can be accepted by NPDA but it can not be accepted by a DPDA.

Example 6.5.1

Design a PDA for detection of odd palindrome over $\{a, b\}$.

Solution :

An odd palindrome will be of the form

1. waw^R

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| w | w | R | w | w | R | w | w | w |
2. wbw^R

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| w | w | R | w | w | R | w | w | R | w |

If the length of w is n then a palindrome of odd length is :

First n characters are equal to the last n characters in reverse order with middle character as 'a' or 'b'.

Algorithm

There is no way of finding the middle position of a string by a PDA, therefore the middle position is fixed non-deterministically.

1. First n characters are pushed onto the stack, where n is non-deterministic.
2. The n characters on the stack are matched with the last n characters of the input string.
3. n is decided non-deterministically. Every character out of first n characters should be considered for two cases :
 - (a) It is not the middle character – push the current character using the transition :

$$\delta(q_0, a, \epsilon) \Rightarrow (q_0, a)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_0, b)$$

- (b) It is a middle character – go for matching of second half with the first half.

$$\delta(q_0, a, \epsilon) \Rightarrow (q_1, \epsilon)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_1, \epsilon)$$

The status of the stack and the state of the machine is shown in the Fig. Ex. 6.5.1. Input applied is ababa.

- Left child → current input is taken as the middle character
- Right child → current input is not a middle character.

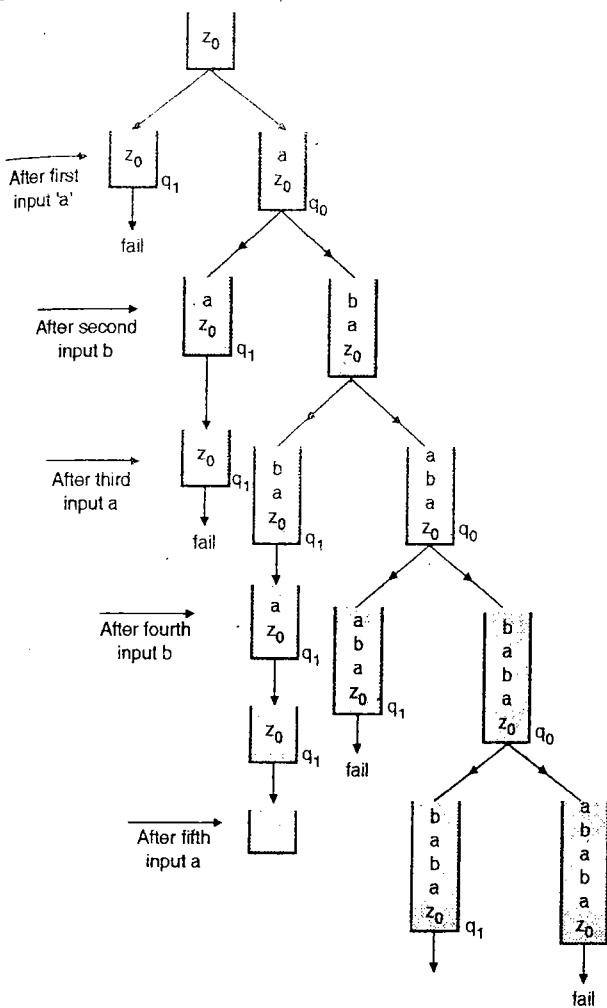


Fig. Ex. 6.5.1 : Processing of string by the PDA. String is taken as "ababa"

The transition table for the PDA is given below,

$$\delta(q_0, a, \epsilon) \Rightarrow \{(q_1, \epsilon), (q_0, a)\}$$

ϵ indicates that irrespective of the current stack symbol, perform the transition.

$$\delta(q_0, b, \epsilon) \Rightarrow \{(q_1, \epsilon), (q_0, b)\}$$

$$\delta(q_1, a, a) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, b, b) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) \Rightarrow \{(q_1, \epsilon)\} \text{ [Accept through an empty stack]}$$

Where, The set of states $Q = \{q_0, q_1\}$

The set input alphabet $\Sigma = \{a, b\}$

The set of stack symbols $\Gamma = \{a, b, z_0\}$

Starting state = q_0

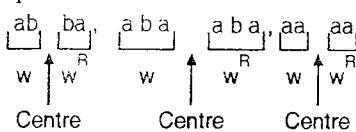
Initial stack symbol = z_0

Example 6.5.2

Design a PDA for detection of even palindrome over $\{a, b\}$.

Solution :

An even palindrome will be of the form ww^R



If the length of w is n then a palindrome of even length is :

First n characters are equal to the last n characters in the reverse order.

- The character immediately before the middle position will be identical to the character immediately after the middle position.

Algorithm : There is no way of finding the middle position by a PDA; therefore the middle position is fixed non-deterministically.

1. First n characters are pushed onto the stack. n is non-deterministic.
2. The n characters on the stack are matched with the last n characters of the input string.
3. n is decided non-deterministically. Every character out of first n characters, whose previous character is same as itself should be considered for two cases :

- (a) It is first character of the second half.

- Pop the current stack symbol using the transitions :

$$\delta(q_0, a, a) \Rightarrow (q_1, \epsilon)$$

$$\delta(q_0, b, b) \Rightarrow (q_1, \epsilon)$$

must be identical

- (b) It belongs to first half.

- Push the current input

$$\delta(q_0, a, \epsilon) \Rightarrow (q_0, a)$$

$$\delta(q_0, b, \epsilon) \Rightarrow (q_0, b)$$

4. n is decided non-deterministically. Every character out of first n characters, whose previous character is not same as itself should be pushed onto the stack.

- Push the current symbol using the transitions :

$$\delta(q_0, a, b) \Rightarrow (q_0, ab)$$

$$\delta(q_0, b, a) \Rightarrow (q_0, ba)$$

The transition table for the PDA is given below :

$$\delta(q_0, a, z_0) \Rightarrow \{(q_0, az_0)\}$$

$$\delta(q_0, b, z_0) \Rightarrow \{(q_0, bz_0)\}$$

$$\delta(q_0, a, a) \Rightarrow \{(q_0, aa), (q_1, \epsilon)\}$$

$$\delta(q_0, a, b) \Rightarrow \{(q_0, ab)\}$$

$$\delta(q_0, b, a) \Rightarrow \{(q_0, ba)\}$$

$$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, \epsilon)\}$$



$$\delta(q_1, a, a) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, b, b) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) \Rightarrow \{(q_1, \epsilon)\} \text{ [Accept through an empty stack]}$$

Where, the set of states $Q = \{q_0, q_1\}$

the set of input symbols $\Sigma = \{a, b\}$

the set of stack symbols $\Gamma = \{a, b, z_0\}$

Starting state = q_0

Initial stack symbol = z_0

Example 6.5.3

Design a PDA for detection of palindromes over $\{a, b\}$.

Solution : A palindrome will be of the form :

1. ww^R — even palindrome

2. waw^R

3. wbw^R — odd palindrome

If the length of w is n then a palindrome is :

First n characters are equal to the last n characters in the reverse order with the middle character as :

(1) a [For odd palindrome]

(2) b [For odd palindrome]

(3) ϵ [For even palindrome]

The transition table for the PDA is given below :

$$\delta(q_0, a, z_0) \Rightarrow \{(q_1, z_0), (q_0, az_0)\}$$

$$\delta(q_0, b, z_0) \Rightarrow \{(q_1, z_0), (q_0, bz_0)\}$$

$$\delta(q_0, a, a) \Rightarrow \{(q_0, aa), (q_1, a), (q_1, \epsilon)\}$$

$$\delta(q_0, a, b) \Rightarrow \{(q_0, ab), (q_1, b)\}$$

$$\delta(q_0, b, a) \Rightarrow \{(q_0, ba), (q_1, a)\}$$

$$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, b), (q_1, \epsilon)\}$$

$$\delta(q_1, a, a) \Rightarrow \{(q_1, \epsilon)\}$$

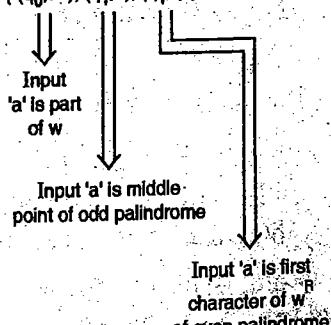
$$\delta(q_1, b, b) \Rightarrow \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) \Rightarrow \{(q_1, \epsilon)\}$$

[Accept through an empty stack].

Details of important transitions

The transaction, $\delta(q_0, a, a) \Rightarrow \{(q_0, aa), (q_1, a), (q_1, \epsilon)\}$



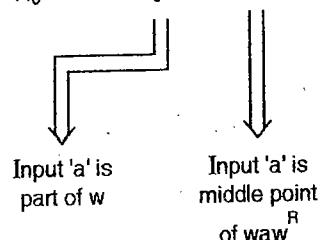
The transition rule for $\delta(q_0, a, a)$, must consider the three cases :

1. Input 'a' is part of w of the palindrome.

2. Input 'a' is middle character of w^R

3. Input 'a' is the first character of w^R .

The transaction, $\delta(q_0, a, b) \Rightarrow \{(q_0, ab), (q_1, b)\}$



Example 6.5.4

Construct PDA for the language

$$L = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$$

Solution :

Algorithm

Machine has to non-deterministically figure out which of the two conditions will be satisfied by the string :

1. $i \neq j$, by entering the state q_1 on first input.

2. $j \neq k$, by entering the state q_2 on first input.

$$\delta(q_0, a, z_0) = \{(q_1, aa), (q_2, z_0)\}$$

In state q_1

1. Initial a's will be pushed.

2. For every input symbol b, an 'a' should be erased from the stack.

3. If a's and b's are not equal i.e. either some a's are left on the stack or some b's are still in the input then the string should be accepted.

In state q_2

1. Initial a's will be skipped.

2. For every input symbol b, a 'b' should be pushed.

3. For every input symbol c, a 'ab' should be erased from the stack.

4. If b's and c's are not equal i.e. either some b's are left on the stack or some c's are still in the input then the string should be accepted.

Transitions for PDA :

$$\delta(q_0, a, z_0) = \{(q_1, aa), (q_2, z_0)\}$$

$$\delta(q_1, a, a) = \{(q_1, aa)\}$$

$$\delta(q_1, b, a) = \{(q_3, \epsilon)\}$$

$$\delta(q_3, b, z_0) = \{(q_f, z_0)\}$$

$$\delta(q_3, c, a) = \{(q_f, \epsilon)\}$$

$$\delta(q_3, b, a) = \{(q_3, \epsilon)\}$$



$$\begin{aligned}
 \delta(q_2, a, z_0) &= \{(q_2, \epsilon)\} \\
 \delta(q_2, b, z_0) &= \{(q_4, bz_0)\} \\
 \delta(q_4, b, b) &= \{(q_4, bb)\} \\
 \delta(q_4, c, b) &= \{(q_4, \epsilon)\} \\
 \delta(q_4, \epsilon, b) &= \{(q_f, \epsilon)\} \\
 \delta(q_f, b, \epsilon) &= \{(q_f, \epsilon)\} \\
 \delta(q_f, c, \epsilon) &= \{(q_f, \epsilon)\} \\
 \delta(q_f, \epsilon, \epsilon) &= \{(q_f, \epsilon)\}
 \end{aligned}$$

6.6 Push Down Automata and Context Free Language

The class of languages accepted by pushdown automata is exactly the class of context-free languages. The following three classes of languages are same :

1. Context Free Language defined by CFG.
2. Languages accepted by PDA by final state.
3. Languages accepted by PDA by empty stack.

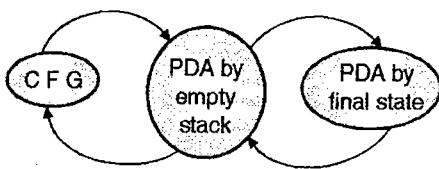


Fig. 6.6.1 : Equivalence of PDA and CFG

- It is possible to find a PDA for a CFG
- It is possible to find a CFG for a PDA.

6.6.1 Construction of PDA from CFG

From a given CFG $G = (V, T, P, S)$, we can construct a PDA, M that simulates the leftmost derivation of G.

The PDA accepting $L(G)$ by empty stack is given by :

$$\begin{aligned}
 M &= (\{q\}, T, V \cup T, \delta, q, S, \phi) \\
 &\quad [M \text{ is a PDA for } L(G)]
 \end{aligned}$$

Where δ is defined by :

1. For each variable $A \in V$, include a transition,
 $\delta(q, \epsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$
2. For each terminal $a \in T$, include a transition
 $\delta(q, a, a) \Rightarrow \{(q, \epsilon)\}$

Example 6.6.1

Find a PDA for the given grammar
 $S \rightarrow 0S1 \mid 00 \mid 11$

Solution : The equivalent PDA, M is given by :

$$M = (\{q\}, \{0, 1\}, \{0, 1, S\}, \delta, q, S, \phi),$$

where δ is given by :

$$\begin{aligned}
 \delta(q, \epsilon, S) &= \{(q, 0S1), (q, 00), (q, 11)\} \\
 \delta(q, 0, 0) &= \{(q, \epsilon)\} \\
 \delta(q, 1, 1) &= \{(q, \epsilon)\}
 \end{aligned}$$

Example 6.6.2

Convert the grammar

$$S \rightarrow 0S1 \mid A$$

$$A \rightarrow 1A0 \mid S \mid \epsilon$$

to PDA that accepts the same language by empty stack.

Solution :

Step 1 : for each variable $A \in V$, include a transition
 $\delta(q, \epsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$

$$\begin{aligned}
 \therefore \delta(q, \epsilon, S) &\Rightarrow \{(q, 0S1), (q, A)\} \\
 \delta(q, \epsilon, A) &\Rightarrow \{(q, 1A0), (q, S), (q, \epsilon)\}
 \end{aligned}$$

Step 2 : For each terminal $a \in T$, include a transition
 $\delta(q, a, a) \Rightarrow (q, \epsilon)$

$$\begin{aligned}
 \therefore \delta(q, 0, 0) &= \{(q, \epsilon)\} \\
 \delta(q, 1, 1) &= \{(q, \epsilon)\}
 \end{aligned}$$

Therefore, the PDA is given by :

$$M = (\{q\}, \{0, 1\}, \{S, A, 0, 1\}, \delta, q, S, \phi)$$

where δ is :

$$\begin{aligned}
 \delta(q, \epsilon, S) &= \{(q, 0S1), (q, A)\} \\
 \delta(q, \epsilon, A) &= \{(q, 1A0), (q, S), (q, \epsilon)\} \\
 \delta(q, 0, 0) &= \{(q, \epsilon)\} \\
 \delta(q, 1, 1) &= \{(q, \epsilon)\}
 \end{aligned}$$

Example 6.6.3 :

Let G be the grammar given by

$$S \rightarrow aABB \mid aAA, A \rightarrow aBB \mid a, B \rightarrow bBB \mid A.$$

Construct NPDA that accepts the language generated by this grammar.

Solution : The equivalent PDA, M is given by :

$$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$$

where δ is given by :

$$\delta(q, \epsilon, S) \Rightarrow \{(q, aABB), (q, aAA)\}$$

$$\delta(q, \epsilon, A) \Rightarrow \{(q, aBB), (q, a)\}$$

For each production in given grammar

$$\delta(q, \epsilon, B) \Rightarrow \{(q, bBB), (q, A)\}$$

$$\delta(q, a, a) \Rightarrow (q, \epsilon)$$

$$\delta(q, b, b) \Rightarrow (q, \epsilon)$$

For each terminal in T.

Example 6.6.4 SPPU - May 16, 4 Marks

Construct a PDA equivalent to the following CFG.
 $S \rightarrow 0BB$


 $B \rightarrow 0S \mid 1S \mid 0$
Test if 010^4 is in the language

Solution : The equivalent PDA, M is given by

$M = (\{q\}, \{0, 1\}, \{0, 1, S, B\}, \delta, q, S, \phi),$

where δ is given by

$\delta(q, \epsilon, S) \Rightarrow ((q, 0BB))$

$\delta(q, \epsilon, B) \Rightarrow ((q, 0S), (q, 1S), (q, 0))$ For each production in the given grammar

$\delta(q, 0, 0) \Rightarrow ((q, \epsilon))$

$\delta(q, 1, 1) \Rightarrow ((q, \epsilon))$ For each terminal.

Acceptance of 010^4 by M

$\delta(q, \epsilon, S) = (q, 0BB)$

$\delta(q, 010000, S) \xrightarrow{\quad} (q, 010000, 0BB)$

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

$\xrightarrow{\quad} (q, 10000, BB)$

$\delta(q, \epsilon, B) = (q, 1S)$

$\xrightarrow{\quad} (q, 10000, 1SB)$

$\delta(q, 1, 1) = (q, \epsilon)$

$\xrightarrow{\quad} (q, 0000, SB)$

$\delta(q, \epsilon, S) = (q, 0BB)$

$\xrightarrow{\quad} (q, 0000, 0BBB)$

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

$\xrightarrow{\quad} (q, 000, BBB)$

$\delta(q, \epsilon, B) = (q, 0)$

$\xrightarrow{\quad} (q, 000, 0BB)$

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

$\xrightarrow{\quad} (q, 00, BB)$

$\delta(q, \epsilon, B) = (q, 0)$

$\xrightarrow{\quad} (q, 00, 0B)$

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

$\xrightarrow{\quad} (q, 0, B)$

$\delta(q, \epsilon, B) = (q, 0)$

$\xrightarrow{\quad} (q, 0, 0)$

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

$\xrightarrow{\quad} (q, \epsilon, \epsilon)$

Thus the string 010^4 is accepted by M using an empty stack.

$\therefore 010^4 \in L$

Example 6.6.5

Design a PDA to recognize the language generated by the following grammar :

$S \rightarrow S + S \mid S * S \mid 4 \mid 2$

Show the acceptance of the input string : $2 + 2 * 4$ by this PDA.

Solution :

The equivalent PDA, M is given by :

$M = (\{q\}, \{+, *\}, \{4, 2\}, \{+, *, 4, 2, S\}, \delta, q, S, \phi)$

Where δ is given by :

$\delta(q, \epsilon, S) \Rightarrow \{(q, S + S), (q, S * S), (q, 4), (q, 2)\}$

For every production in G

$\delta(q, +, +) = \{(q, \epsilon)\}$

$\delta(q, *, *) = \{(q, \epsilon)\}$

$\delta(q, 2, 2) = \{(q, \epsilon)\}$

$\delta(q, 4, 4) = \{(q, \epsilon)\}$

for every terminal in T.

Acceptance of $2 + 2 * 4$ by this PDA

$\delta(q, \epsilon, S) \Rightarrow (q, S + S)$

$\delta(q, 2 + 2 * 4, S) \xrightarrow{\quad} (q, 2 + 2 * 4, S + S)$

$\delta(q, \epsilon, S) \Rightarrow (q, 2)$

$\xrightarrow{\quad} (q, 2 + 2 * 4, 2 + S)$

$\delta(q, 2, 2) \Rightarrow (q, \epsilon)$

$\xrightarrow{\quad} (q, + 2 * 4, + S)$

$\delta(q, +, +) \Rightarrow (q, \epsilon)$

$\xrightarrow{\quad} (q, 2 * 4, S)$

$\delta(q, \epsilon, S) \Rightarrow (q, S * S)$

$\xrightarrow{\quad} (q, 2 * 4, S * S)$

$\delta(q, \epsilon, S) \Rightarrow (q, 2)$

$\xrightarrow{\quad} (q, 2 * 4, 2 * S)$

$\delta(q, 2, 2) \Rightarrow (q, \epsilon)$

$\xrightarrow{\quad} (q, * 4, * S)$

$\delta(q, *, *) \Rightarrow (q, \epsilon)$

$\xrightarrow{\quad} (q, 4, S)$

$\delta(q, \epsilon, S) = (q, 4)$

$\xrightarrow{\quad} (q, 4, 4)$

$\delta(q, 4, 4) = (q, \epsilon)$

$\xrightarrow{\quad} (q, \epsilon, \epsilon)$

Example 6.6.6

Construct PDA for

$S \rightarrow 0AB,$

$A \rightarrow 1A \mid 1,$

$B \rightarrow 0B \mid 1A \mid 0$

Is your PDA a DPDA or NPDA ?

Solution :

The equivalent PDA, M is given by,

$M = (\{q\}, \{0, 1\}, \{0, 1, S, A, B\}, \delta, q, S, \phi)$



where, δ is given by,

$$\delta(q, \epsilon, S) = \{(q, 0AB)\}$$

$$\delta(q, \epsilon, A) = \{(q, 1A), (q, 1)\}$$

$$\delta(q, \epsilon, B) = \{(q, 0B), (q, 1A), (q, 0)\}$$

$$\delta(q, 0, 0) = \{(q, \epsilon)\}$$

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

The PDA is an NPDA.

For each production in the given grammar

For each terminal

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

$$\delta(q_0,), () = (q_0, \epsilon)$$

$$\delta(q_0,], [) = (q_0, \epsilon)$$

$$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$$

- Accepted through an empty stack.

- Z_0 is the initial stack symbol.

Example 6.6.10 [SPPU - May 15, Dec. 16, 8 Marks]

Construct PDA equivalent to the given CFG :

$$S \rightarrow 0A1|0BA, \quad A \rightarrow S01|0, \quad B \rightarrow 1B|1$$

Solution : The PDA, M is given by :

$$M = \{ \{q\}, \{0, 1\}, \{0, 1, S, A, B\}, \delta, q, s, \phi \}$$

Where δ is given by :

$$\delta(q, \epsilon, S) = \{(q, 0A1), (q, 0BA)\}$$

$$\delta(q, \epsilon, A) = \{(q, S01), (q, 0)\}$$

$$\delta(q, \epsilon, B) = \{(q, 1B), (q, 1)\}$$

$$\delta(q, 0, 0) = \{(q, \epsilon)\}$$

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

Example 6.6.11

Show that if L is generated by a CFG then there exists a PDA accepting L.

Solution :

Proof

Let $G = (V, T, P, S)$ be a Context Free Grammar and a PDA, M is constructed as given below :

$$M = \{ \{q\}, T, V \cup T, \delta, q, S, \phi \}$$

where δ is defined by :

1. For each variable $A \in V$, include a transition,
 $\delta(q, \epsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$

2. For each terminal $a \in T$, include a transition

$$\delta(q, a, a) \Rightarrow \{(q, \epsilon)\}$$

- The transitions of M are designed to simulate a leftmost derivation of a string.

1. The transition of the form

$$\delta(q, \epsilon, A) \Rightarrow (q, \alpha)$$

is for expansion of the topmost variable of the stack.

2. The transition of the form

$$\delta(q, a, a) \Rightarrow (q, \epsilon)$$

is for removing terminals from the stack so that a variable is exposed for further expansion.

To prove that a PDA constructed using above rules is equivalent to G.

We can prove that a word $w \in L(M)$ if and only if $w \in L(G)$.

Example 6.6.8

Obtain PDA for the following grammar

$$S \rightarrow aABC, \quad A \rightarrow aB \mid a$$

$$B \rightarrow bA \mid b, \quad C \rightarrow a.$$

Solution : The equivalent PDA, M is given by,

$$M = \{ \{q\}, \{a, b\}, \{a, b, S, A, B, C\}, \delta, q, S, \phi \}$$

Where, δ is given by,

$$\delta(q, \epsilon, S) = \{(q, aABC)\}$$

$$\delta(q, \epsilon, A) = \{(q, aB), (q, a)\}$$

$$\delta(q, \epsilon, B) = \{(q, bA), (q, b)\}$$

$$\delta(q, \epsilon, C) = \{(q, a)\}$$

For each production in the grammar

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

$$\delta(q, b, b) = \{(q, \epsilon)\}$$

For each terminal

Example 6.6.9 [SPPU - May 12, 8 Marks]

Obtain a PDA to accept a string of balanced parentheses. The parentheses to be considered are $(,)$, $[,]$.

Solution : Transitions of PDA

$$\delta(q_0, (, Z_0) = (q_0, (Z_0)$$

$$\delta(q_0, [Z_0) = (q_0, [Z_0)$$

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

$$\delta(q_0, (, [) = (q_0, ([)$$

$$\delta(q_0, [, () = (q_0, [()$$



where, $L(M)$ is the language of PDA and $L(G)$ is the language of the given CFG.

Let us take a word $w \in L(G)$. The word w can be derived using the leftmost derivation.

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = w$$

$$\text{where, } \alpha_i \rightarrow \alpha_{i+1}$$

is obtained by single application of leftmost derivation, using a production in grammar G. We will show that for each α_i there is a unique configuration of PDA. α_n corresponds to the configuration of PDA accepting w .

$$\text{Let } \alpha_i = x_i \beta_i$$

$$\text{Where, } x_i \in T^* \text{ and } \beta_i \in (V \cup T)^*$$

The string α_i corresponds to a unique configuration of PDA given by a pair :

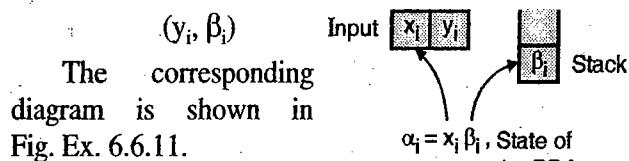
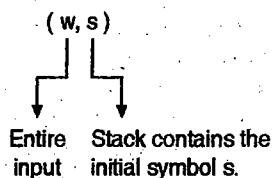


Fig. Ex. 6.6.11 : Unique configuration of PDA for α_i

- x_i portion of the input string w has already been scanned.
- The y_i is the portion of the input string, yet to be scanned.
- The stack contains β_i .
- The theorem can be proved by induction on i of α_i .

Base case : $i = 0$

$\alpha_0 = S$, the configuration of PDA is given by



Induction step

We have to show that the correspondence is preserved for $j = i + 1$ also, if the correspondence for α_i is assumed.

Without the loss of generality, we can assume that the given grammar is in GNF.

If $(i + 1)^{\text{th}}$ input symbol is a_{i+1} and the first variable of β_i is V_i then the top of the stack contains V_i . The variable V_i can be expanded using the production,

$$V_i \Rightarrow a_{i+1} y$$

Similarly, the top symbol of the stack can be replaced with $a_{i+1} y$ and then a_{i+1} can be popped out using the transition

$$\delta(q, a_{i+1}, a_{i+1}) \Rightarrow (q, \epsilon)$$

Thus, there is one-to-one correspondence between the strings α_i and the configuration of PDA. Thus a string $w \in L(G)$ will be accepted by the PDA M.

6.6.2 Construction of CFG from PDA

We can find the Context Free Grammar G for any PDA, M such that

$$L(G) = L(M)$$

i.e., we can construct an equivalent CFG for a PDA.

The variables of the CFG, so constructed will be of the form :

$$[pXq], \text{ where } p, q \in Q \text{ and } X \in \Gamma$$

Let the PDA is given by :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, \phi)$$

Where, z is the initial stack symbol.

Then an equivalent CFG is given by

$$G = (V, \Sigma, P, S) \text{ where}$$

$$V = \{S, [p^x q] \mid p, q \in Q \text{ and } X \in \Gamma\}$$

Example

If $Q = \{q_0, q_1\}$ and $\Gamma = \{a, b, z\}$ then the possible set of variables in the corresponding CFG is given by :

1. S
2. $[q_0^a q_0], [q_0^a q_1], [q_1^a q_0], [q_1^a q_1]$
3. $[q_0^b q_0], [q_0^b q_1], [q_1^b q_0], [q_1^b q_1]$
4. $[q_0^z q_0], [q_0^z q_1], [q_1^z q_0], [q_1^z q_1]$

Set of productions for the equivalent CFG

1. Add the following productions for the start symbol S.

$$S \rightarrow [q_0^z q_i] \text{ for each } q_i \in Q, \text{ where } z \text{ is the start symbol}$$

2. For each transition of the form

$$\delta(q_i, a, B) \Rightarrow (q_j, C)$$

Where, (a) $q_i, q_j \in Q$

(b) a belongs to $(\Sigma \cup \epsilon)$

(c) B and C belong to $(\Gamma \cup \epsilon)$

Then for each $q \in Q$, we add the production :

$$[q_i^B q] \rightarrow a [q_j^C q]$$



3. For each transition of the form

$$\delta(q_i, a, B) \Rightarrow (q_j, C_1 C_2)$$

Where,

$$(a) q_i, q_j \in Q$$

$$(b) a \text{ belongs to } (\Sigma \cup \epsilon)$$

$$(c) B, C_1 \text{ and } C_2 \text{ belongs to } \Gamma$$

then for each $p_1, p_2 \in Q$, we add the production

$$[q_i \ B \ p_1] \rightarrow a [q_j \ C_1 \ p_2] [p_2 \ C_2 \ p_1]$$

Example 6.6.12

Convert PDA to CFG. PDA is given by

$M = (\{p, q\}, \{0, 1\}, \{x, z\}, \delta, q, z)$, transition function δ is defined by :

$$\delta(q, 1, z) \Rightarrow \{(q, xz)\}$$

$$\delta(q, 1, x) \Rightarrow \{(q, xx)\}$$

$$\delta(q, \epsilon, x) \Rightarrow \{(q, \epsilon)\}$$

$$\delta(q, 0, x) \Rightarrow \{(p, x)\}$$

$$\delta(p, 1, x) \Rightarrow \{(p, \epsilon)\}$$

$$\delta(p, 0, z) \Rightarrow \{(q, z)\}$$

Solution :

Step 1 : Add productions for the start symbol.

$$S \rightarrow [q^z q]$$

$$S \rightarrow [q^z p]$$

Step 2 : Add productions for $\delta(q, 1, z) \Rightarrow \{(q, xz)\}$

$$[q^z q] \rightarrow 1 [q^x q] [q^z q]$$

$$[q^z q] \rightarrow 1 [q^x p] [p^z q]$$

$$[q^z p] \rightarrow 1 [q^x q] [q^z p]$$

$$[q^z p] \rightarrow 1 [q^x p] [p^z p]$$

Step 3 : Add productions For $\delta(q, 1, x) \Rightarrow \{(q, xx)\}$

$$[q^x q] \rightarrow 1 [q^x q] [q^x q]$$

$$[q^x q] \rightarrow 1 [q^x p] [p^x q]$$

$$[q^x p] \rightarrow 1 [q^x q] [q^x p]$$

$$[q^x p] \rightarrow 1 [q^x p] [p^x p]$$

Step 4 : Add productions for $\delta(q, \epsilon, x) \Rightarrow \{(q, \epsilon)\}$

$$[q^x q] \rightarrow \epsilon$$

Step 5 : Add productions for $\delta(q, 0, x) \Rightarrow \{(p, x)\}$

$$[q^x q] \rightarrow 0 [p^x q]$$

$$[q^x p] \rightarrow 0 [p^x p]$$

Step 6 : Add productions for $\delta(p, 1, x) \Rightarrow \{(p, \epsilon)\}$

$$[p^x p] \rightarrow 1$$

Step 7 : Add productions for $\delta(p, 0, z) \Rightarrow \{(q, z)\}$

$$[p^z q] \rightarrow 0 [q^z q]$$

$$[p^z p] \rightarrow 0 [q^z p]$$

Step 8 : Renaming of variables :

| Original name | New name |
|---------------|----------|
| $[q^z q]$ | A |
| $[q^z p]$ | B |
| $[p^z q]$ | C |
| $[p^z p]$ | D |
| $[q^x q]$ | E |
| $[q^x p]$ | F |
| $[p^x q]$ | G |
| $[p^x p]$ | H |

The set of productions can be written as :

$$S \rightarrow A \mid B$$

~~$$A \rightarrow 1EA \mid 1FC$$~~

$$B \rightarrow 1EB \mid 1FD$$

$$E \rightarrow IEE \mid 1FG$$

$$F \rightarrow 1EF \mid 1FH$$

$$E \rightarrow \epsilon$$

$$E \rightarrow 0G$$

$$F \rightarrow 0H$$

$$H \rightarrow 1$$

$$C \rightarrow 0A$$

$$D \rightarrow 0B$$

Step 9 : Simplification of grammar

Symbol G does not come on the left side of the production, hence it can be eliminated.

The equivalent set of productions is :

$$S \rightarrow A \mid B$$

~~$$A \rightarrow 1EA \mid 1FC$$~~

$$B \rightarrow 1EB \mid 1FD$$

$$E \rightarrow 1EE \mid \epsilon$$

$$F \rightarrow 1EF \mid 1FH \mid 0H$$

$$H \rightarrow 1$$

$$C \rightarrow 0A$$

$$D \rightarrow 0B$$

Example 6.6.13 | SPPU - May 16, Dec. 16, 8 Marks

Give the CFG generating the language accepted by the following PDA :

$$M = (\{q_0, q_1\}, \{0, 1\}, \{z_0, x\}, \delta, q_0, z_0, \phi)$$

when δ is given below :

$$\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$$

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

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

$$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$$

$$\delta(q_1, 1, x) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, 0, z_0) = \{(q_0, z_0)\}$$

Solution :

Step 1 : Add productions for the start symbol

$$S \rightarrow [q_0 \ x \ z_0]$$

$$S \rightarrow [q_0 \ z_0]$$

Step 2 : Add productions for $\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$

$$[q_0 \ x \ z_0] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ z_0]$$

$$[q_0 \ x \ z_0] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ z_0]$$

$$[q_0 \ z_0] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ z_0]$$

$$[q_0 \ z_0] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ z_0]$$

Step 3 : Add productions for $\delta(q_0, 1, x) \Rightarrow \{(q_0, xx)\}$

$$[q_0 \ x \ z_0] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ z_0]$$

$$[q_0 \ x \ z_0] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ z_0]$$

$$[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ q_1]$$

$$[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ q_1]$$

Step 4 : Add productions for $\delta(q_0, 0, x) \Rightarrow \{(q_1, x)\}$

$$[q_0 \ x \ z_0] \rightarrow 0 [q_1 \ x \ z_0]$$

$$[q_0 \ x \ z_0] \rightarrow 0 [q_1 \ x \ q_1]$$

Step 5 : Add productions for $\delta(q_0, \epsilon, z_0) = \{(q_1, \epsilon)\}$

$$[q_0 \ z_0] \rightarrow \epsilon$$

Step 6 : Add production for $\delta(q_1, 1, x) \Rightarrow \{(q_1, \epsilon)\}$

$$[q_1 \ x \ z_0] \rightarrow 1$$

Step 7 : Add productions for $\delta(q_1, 0, z_0) \Rightarrow \{(q_0, z_0)\}$

$$[q_1 \ z_0] \rightarrow 0 [q_0 \ z_0]$$

$$[q_1 \ z_0] \rightarrow 0 [q_0 \ z_0]$$

$$\delta(q_0, a, a) = \{(q_0, aa)\}$$

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

$$\delta(q_1, b, a) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$$

Obtain CFG equivalent to PDA.

Solution :

Step 1 : Add productions for the start symbol.

$$S \rightarrow [q_0 \ z_0]$$

$$S \rightarrow [q_0 \ q_1]$$

Step 2 : Add productions for $(q_0, a, a) = \{(q_0, aa)\}$

$$[q_0 \ a \ z_0] \rightarrow a [q_0 \ a \ q_0] [q_0 \ a \ q_0]$$

$$[q_0 \ a \ z_0] \rightarrow a [q_0 \ a \ q_1] [q_1 \ a \ q_0]$$

$$[q_0 \ a \ q_1] \rightarrow a [q_0 \ a \ q_0] [q_0 \ a \ q_1]$$

$$[q_0 \ a \ q_1] \rightarrow a [q_0 \ a \ q_1] [q_1 \ a \ q_1]$$

Step 3 : Add productions for $\delta(q_0, b, a) = \{(q_1, \epsilon)\}$

$$[q_0 \ a \ q_1] \rightarrow b$$

Step 4 : Add productions for $\delta(q_1, b, a) = \{(q_1, \epsilon)\}$

$$[q_1 \ a \ q_1] \rightarrow b$$

Step 5 : Add productions for $\delta(q_1, \epsilon, z_0) \rightarrow \{(q_1, \epsilon)\}$

$$[q_1 \ z_0] \rightarrow \epsilon$$

Example 6.6.15 SPPU - Dec. 15, 9 Marks

For the PDA

$$(\{q_0, q_1\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \phi) \text{ where } \delta \text{ is}$$

$$\delta(q_0, \epsilon, z_0) = \{(q_1, \epsilon)\}$$

$$\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$$

$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$

$$\delta(q_0, 1, 0) = \{(q_0, 10)\}$$

$$\delta(q_0, 1, 1) = \{(q_0, 11)\}$$

$$\delta(q_0, 0, 1) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, 0, 1) = \{(q_1, \epsilon)\}$$

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

$$\delta(q_1, \epsilon, z_0) = \{(q_1, \epsilon)\}$$

Obtain CFG accepted by the above PDA and simplify the CFG and describe the language it accepts.

Example 6.6.14

Consider the PDA with the following moves :

$$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$$



Solution :

PDA transition

1. Productions due to start symbol S.

Corresponding productions

$$S \rightarrow [q_0 \ z_0 \ q_0]$$

$$S \rightarrow [q_0 \ z_0 \ q_1]$$

$$[q_0 \ z_0 \ q_1] \rightarrow \epsilon$$

$$[q_0 \ z_0 \ q_0] \rightarrow 0 [q_0 \ 0 \ q_0] [q_0 \ z_0 \ q_0]$$

$$[q_0 \ z_0 \ q_0] \rightarrow 0 [q_0 \ 0 \ q_1] [q_1 \ z_0 \ q_0]$$

$$[q_0 \ z_0 \ q_1] \rightarrow 0 [q_0 \ 0 \ q_0] [q_0 \ z_0 \ q_1]$$

$$[q_0 \ z_0 \ q_1] \rightarrow 0 [q_0 \ 0 \ q_1] [q_1 \ z_0 \ q_1]$$

$$[q_0 \ 0 \ q_0] \rightarrow 0 [q_0 \ 0 \ q_0] [q_0 \ 0 \ q_0]$$

$$[q_0 \ 0 \ q_0] \rightarrow 0 [q_0 \ 0 \ q_1] [q_1 \ 0 \ q_0]$$

$$[q_0 \ 0 \ q_1] \rightarrow 0 [q_0 \ 0 \ q_0] [q_0 \ 0 \ q_1]$$

$$[q_0 \ 0 \ q_1] \rightarrow 0 [q_0 \ 0 \ q_1] [q_1 \ 0 \ q_1]$$

$$[q_0 \ 0 \ q_0] \rightarrow 1 [q_0 \ 1 \ q_0] [q_0 \ 0 \ q_0]$$

$$[q_0 \ 0 \ q_0] \rightarrow 1 [q_0 \ 1 \ q_1] [q_1 \ 0 \ q_0]$$

$$[q_0 \ 0 \ q_1] \rightarrow 1 [q_0 \ 1 \ q_0] [q_0 \ 0 \ q_1]$$

$$[q_0 \ 0 \ q_1] \rightarrow 1 [q_0 \ 1 \ q_1] [q_1 \ 0 \ q_1]$$

$$[q_0 \ 1 \ q_0] \rightarrow 1 [q_0 \ 1 \ q_0] [q_0 \ 1 \ q_0]$$

$$[q_0 \ 1 \ q_0] \rightarrow 1 [q_0 \ 1 \ q_1] [q_1 \ 1 \ q_0]$$

$$[q_0 \ 1 \ q_1] \rightarrow 1 [q_0 \ 1 \ q_0] [q_0 \ 1 \ q_1]$$

$$[q_0 \ 1 \ q_1] \rightarrow 1 [q_0 \ 1 \ q_1] [q_1 \ 1 \ q_1]$$

$$[q_0 \ 1 \ q_1] \rightarrow 0$$

$$[q_1 \ 1 \ q_1] \rightarrow 0$$

$$[q_1 \ 0 \ q_1] \rightarrow 0$$

$$[q_1 \ z_0 \ q_1] \rightarrow \epsilon$$

$\delta(q_0, 0, 0) = (q_0, 00)$

$\delta(q_0, 1, 0) = (q_0, 10)$

$\delta(q_0, 1, 1) = (q_0, 11)$

$\delta(q_0, 0, 1) = (q_1, \epsilon)$

$\delta(q_1, 0, 1) = (q_1, \epsilon)$

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

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Simplification of grammar

We can rename the variables as given below.

$$[q_0 \ z_0 \ q_0] - A, [q_0 \ z_0 \ q_1] - B, [q_1 \ z_0 \ q_0] - C, [q_1 \ z_0 \ q_1] - D$$

$$[q_0 \ z_0 \ q_0] - E, [q_0 \ 0 \ q_1] - F, [q_1 \ 0 \ q_0] - G, [q_1 \ 0 \ q_1] - H$$

$$[q_0 \ 1 \ q_0] - I, [q_0 \ 1 \ q_1] - J, [q_1 \ 1 \ q_0] - K, [q_1 \ 1 \ q_1] - L$$

With the above substitutions, the resulting set of productions can be written as :

$$S \rightarrow A \mid B$$

$$B \rightarrow \epsilon$$

$$A \rightarrow 0EA \mid 0FC$$

$$B \rightarrow 0EB \mid 0FD$$

 $E \rightarrow 0EE \mid 0FG$ $F \rightarrow 0EF \mid 0FH$ $E \rightarrow 1IE \mid 1JG$ $F \rightarrow 1IF \mid 1JH$ $I \rightarrow 1II \mid 1JK$ $J \rightarrow 1IJ \mid 1JL$ $J \rightarrow 0$ $L \rightarrow 0$ $H \rightarrow 0$ $D \rightarrow \epsilon$

1. Removing ϵ -productions :

Nullable set = {D, B, S}

ϵ -productions are removed with resulting set of productions as given below :

 $S \rightarrow A \mid B$ $A \rightarrow 0EA \mid 0FC$ $B \rightarrow 0EB \mid 0FD \mid 0E \mid 0F$ $E \rightarrow 0EE \mid 0FG \mid 1IE \mid 1JG$ $F \rightarrow 0EF \mid 0FH \mid 1IF \mid 1JH$ $H \rightarrow 0$ $I \rightarrow 1II \mid 1JK$ $J \rightarrow 1IJ \mid 1JL \mid 0$ $L \rightarrow 0$

2. Removing non-generating symbols

Set of productions after elimination of non-generating symbols {A, C, D, E, G, I} is given below :

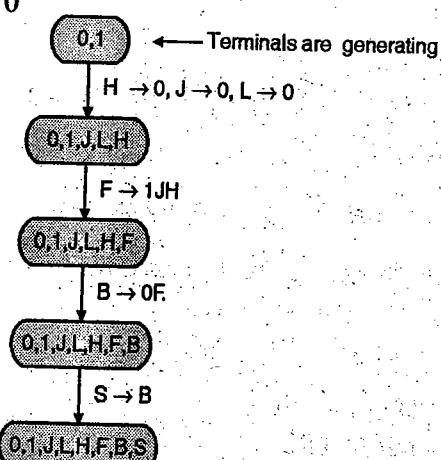
 $S \rightarrow B$ $B \rightarrow 0F$ $F \rightarrow 0FH \mid 1JH$ $H \rightarrow 0$ $J \rightarrow 1JL \mid 0$ $L \rightarrow 0$ 

Fig. Ex. 6.6.15

3. The unit production $S \rightarrow B$ should be removed. The set of productions after elimination of the unit production $S \rightarrow B$ is given below :

 $S \rightarrow 0F$ $F \rightarrow 0FH \mid 1JH$ $H \rightarrow 0$ $J \rightarrow 1JL \mid 0$ $L \rightarrow 0$

Language accepted by the PDA

The language accepted by the PDA is given by :

$$L = \{0^n 1^m 0^{n+m} \mid n, m \geq 1\} \cup \epsilon$$

Example 6.6.16 SPPU - May 15, 8 Marks

Construct a CFG equivalent to PDA.

$$M = (\{q_0, q_1\}, \{0, 1\}, \{B, R\},$$

$\{\delta, q_0, R, \phi\})$ where δ is

$$\delta(q_0, 0, R) = (q_0, BR),$$

$$\delta(q_0, 0, B) = (q_0, BB),$$

$$\delta(q_0, 1, B) = (q_1, B),$$

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

$$\delta(q_1, 0, B) = (q_1, \epsilon),$$

$$\delta(q_1, \epsilon, R) = (q_1, \epsilon)$$

Solution :

| Sr. No. | PDA transition | Corresponding productions in CFG |
|---------|--|--|
| 1. | Production due to start symbol S | $S \rightarrow [q_0 R q_0]$ $S \rightarrow [q_0 R q_1]$ |
| 2. | $\delta(q_0, 0, R) = (q_0, BR)$ | $[q_0 R q_0] \rightarrow 0 [q_0 B q_0] [q_0 R q_0]$ $[q_0 R q_0] \rightarrow 0 [q_0 B q_1] [q_1 R q_0]$ $[q_0 R q_1] \rightarrow 0 [q_0 B q_0] [q_0 R q_1]$ $[q_0 R q_1] \rightarrow 0 [q_0 B q_1] [q_1 R q_1]$ |
| 3. | $\delta(q_0, 0, B) = (q_0, BB)$ | $[q_0 B q_0] \rightarrow 0 [q_0 B q_0] [q_0 B q_0]$ $[q_0 B q_0] \rightarrow 0 [q_0 B q_1] [q_1 B q_0]$ $[q_0 B q_1] \rightarrow 0 [q_0 B q_0] [q_0 B q_1]$ $[q_0 B q_1] \rightarrow 0 [q_0 B q_1] [q_1 B q_1]$ |
| 4. | $\delta(q_0, 1, B) = (q_1, B)$ | $[q_0 B q_0] \rightarrow 1 [q_1 B q_0]$ $[q_0 B q_1] \rightarrow 1 [q_1 B q_1]$ |
| 5. | $\delta(q_1, 1, B) = (q_1, B)$ | $[q_1 B q_0] \rightarrow 1 [q_1 B q_0]$ $[q_1 B q_1] \rightarrow 1 [q_1 B q_1]$ |
| 6. | $\delta(q_1, 0, B) = (q_1, \epsilon)$ | $[q_1 B q_1] \rightarrow \epsilon$ |
| 7. | $\delta(q_1, \epsilon, R) = (q_1, \epsilon)$ | $[q_1 R q_1] \rightarrow \epsilon$ |

**Example 6.6.17**

Construct a PDA accepting $\{a^m b^n a^m \mid m, n > 1\}$ by null store. From the PDA construct the corresponding CFG.

Solution :

Algorithm

- Sequence of a's should be pushed onto the stack in state q_0 .

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

- Sequence of input b's should be skipped. These b's will have no effect on the stack.

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

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

- Initial a's which are on the stack should be matched with the trailing a's in the input. An 'a' should be popped for every 'a' as input till the end of input.

$$\delta(q_1, a, a) = (q_2, \epsilon)$$

$$\delta(q_2, a, a) = (q_2, \epsilon)$$

- Finally, the symbol z_0 should be popped but to make the stack empty.

$$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$$

Transition diagram and the transition table are given below.

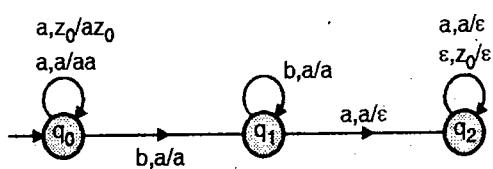


Fig. Ex. 6.6.17 : Transition diagram

Transition table

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

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

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

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

$$\delta(q_1, a, a) = (q_2, \epsilon)$$

$$\delta(q_2, a, a) = (q_2, \epsilon)$$

$$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$$

The PDA $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$

PDA to CFG

Step 1 : Add productions for the start symbol

$$S \rightarrow [q_0, z_0, q_0]$$

$$S \rightarrow [q_0, z_0, q_1]$$

$$S \rightarrow [q_0, z_0, q_2]$$

Step 2 : Add productions for $\delta(q_0, a, z_0) = (q_0, az_0)$

$$[q_0, z_0, q_0] \rightarrow a [q_0, a, q_0] [q_0, z_0, q_0]$$

$$[q_0, z_0, q_0] \rightarrow a [q_0, a, q_1] [q_1, z_0, q_0]$$

$$[q_0, z_0, q_0] \rightarrow a [q_0, a, q_2] [q_2, z_0, q_0]$$

$$[q_0, z_0, q_1] \rightarrow a [q_0, a, q_0] [q_0, z_0, q_1]$$

$$[q_0, z_0, q_1] \rightarrow a [q_0, a, q_2] [q_2, z_0, q_1]$$

$$[q_0, z_0, q_2] \rightarrow a [q_0, a, q_0] [q_0, z_0, q_2]$$

$$[q_0, z_0, q_2] \rightarrow a [q_0, a, q_1] [q_1, z_0, q_2]$$

$$[q_0, z_0, q_2] \rightarrow a [q_0, a, q_2] [q_2, z_0, q_2]$$

Step 3 : Add productions for $\delta(q_0, a, a) = (q_0, aa)$

$$[q_0, a, q_0] \rightarrow a [q_0, a, q_0] [q_0, a, q_0]$$

$$[q_0, a, q_0] \rightarrow a [q_0, a, q_1] [q_1, a, q_0]$$

$$[q_0, a, q_0] \rightarrow a [q_0, a, q_2] [q_2, a, q_0]$$

$$[q_0, a, q_1] \rightarrow a [q_0, a, q_0] [q_0, a, q_1]$$

$$[q_0, a, q_1] \rightarrow a [q_0, a, q_1] [q_1, a, q_1]$$

$$[q_0, a, q_1] \rightarrow a [q_0, a, q_2] [q_2, a, q_1]$$

$$[q_0, a, q_2] \rightarrow a [q_0, a, q_0] [q_0, a, q_2]$$

$$[q_0, a, q_2] \rightarrow a [q_0, a, q_1] [q_1, a, q_2]$$

$$[q_0, a, q_2] \rightarrow a [q_0, a, q_2] [q_2, a, q_2]$$

Step 4 : Add productions for $\delta(q_0, b, a) = (q_1, a)$

$$[q_0, a, q_0] \rightarrow b [q_1, a, q_0]$$

$$[q_0, a, q_1] \rightarrow b [q_1, a, q_1]$$

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

Step 5 : Add productions for $\delta(q_1, b, a) = (q_1, a)$

$$[q_1, a, q_0] \rightarrow b [q_1, a, q_0]$$

$$[q_1, a, q_1] \rightarrow b [q_1, a, q_1]$$

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



Step 6 : Add productions for $\delta(q_1, a, a) = (q_2, \epsilon)$

$$[q_1 \ x \ q_2] \rightarrow a$$

Step 7 : Add productions for $\delta(q_2, a, a) = (q_2, \epsilon)$

$$[q_2 \ x \ q_2] \rightarrow a$$

Step 8 : Add productions for $\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$

$$[q_2 \ z_0 \ q_2] \rightarrow \epsilon$$

Example 6.6.18

Design a PDA and then corresponding CFG for the language that accepts the simple palindrome.

$$L = \{x \in \{a,b\}^* \mid x = x^R\}$$

Solution : Transitions for the PDA are given by :

$$\delta(q_0, a, z) = (q_0, az)$$

$$\delta(q_0, b, z) = (q_0, bz)$$

$$\delta(q_0, c, z) = (q_0, \epsilon)$$

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

$$\delta(q_0, a, b) = (q_0, ab)$$

$$\delta(q_0, b, a) = (q_0, ba)$$

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

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

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

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

$$\delta(q_1, b, b) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z) = (q_1, \epsilon)$$

- z is the initial stack symbol.

- q_0 is the initial state.

- String is accepted through an empty stack.

Productions for the corresponding CFG are given below :

Step 1 : Add productions for the start symbol.

$$S \rightarrow [q_0 \ z \ q_0]$$

$$S \rightarrow [q_0 \ z \ q_1]$$

Step 2 : Add productions for $\delta(q_0, a, z) = (q_0, az)$

$$[q_0 \ z \ q_0] \rightarrow a [q_0 \ a \ q_0] [q_0 \ z \ q_0]$$

$$[q_0 \ z \ q_0] \rightarrow a [q_0 \ a \ q_1] [q_1 \ z \ q_0]$$

$$[q_0 \ z \ q_1] \rightarrow a [q_0 \ a \ q_0] [q_0 \ z \ q_1]$$

$$[q_0 \ z \ q_1] \rightarrow a [q_0 \ a \ q_1] [q_1 \ z \ q_1]$$

Step 3 : Add productions for $\delta(q_0, c, z) = (q_0, \epsilon)$

$$[q_0 \ z \ q_0] \rightarrow c$$

Step 4 : Add productions for $\delta(q_0, a, a) = (q_0, aa)$

$$[q_0 \ a \ q_0] \rightarrow a [q_0 \ a \ q_0] [q_0 \ a \ q_0]$$

$$[q_0 \ a \ q_0] \rightarrow a [q_0 \ a \ q_1] [q_1 \ a \ q_0]$$

$$[q_0 \ a \ q_1] \rightarrow a [q_0 \ a \ q_0] [q_0 \ a \ q_1]$$

$$[q_0 \ a \ q_1] \rightarrow a [q_0 \ a \ q_1] [q_1 \ a \ q_1]$$

Step 5 : Add productions for $\delta(q_0, b, b) = (q_0, bb)$

$$[q_0 \ b \ q_0] \rightarrow b [q_0 \ b \ q_0] [q_0 \ b \ q_0]$$

$$[q_0 \ b \ q_0] \rightarrow b [q_0 \ b \ q_1] [q_1 \ b \ q_0]$$

$$[q_0 \ b \ q_1] \rightarrow b [q_0 \ b \ q_0] [q_0 \ b \ q_1]$$

$$[q_0 \ b \ q_1] \rightarrow b [q_0 \ b \ q_1] [q_1 \ b \ q_1]$$

Step 6 : Add productions for $\delta(q_0, b, a) = (q_0, ba)$

$$[q_0 \ a \ q_0] \rightarrow b [q_0 \ b \ q_0] [q_0 \ a \ q_0]$$

$$[q_0 \ a \ q_0] \rightarrow b [q_0 \ b \ q_1] [q_1 \ a \ q_0]$$

$$[q_0 \ a \ q_1] \rightarrow b [q_0 \ b \ q_0] [q_0 \ a \ q_1]$$

$$[q_0 \ a \ q_1] \rightarrow b [q_0 \ b \ q_1] [q_1 \ a \ q_1]$$

Step 7 : Add productions for $\delta(q_0, b, b) = (q_0, bb)$

$$[q_0 \ b \ q_0] \rightarrow b [q_0 \ b \ q_0] [q_0 \ b \ q_0]$$

$$[q_0 \ b \ q_0] \rightarrow b [q_0 \ b \ q_1] [q_1 \ b \ q_0]$$

$$[q_0 \ b \ q_1] \rightarrow b [q_0 \ b \ q_0] [q_0 \ b \ q_1]$$

$$[q_0 \ b \ q_1] \rightarrow b [q_0 \ b \ q_1] [q_1 \ b \ q_1]$$

Step 8 : Add productions for $\delta(q_0, c, a) = (q_1, a)$

$$[q_0 \ a \ q_0] \rightarrow c [q_1 \ a \ q_0]$$

$$[q_0 \ a \ q_1] \rightarrow c [q_1 \ a \ q_1]$$

Step 9 : Add productions for $\delta(q_0, c, b) = (q_1, b)$

$$[q_0 \ b \ q_0] \rightarrow c [q_1 \ b \ q_0]$$

$$[q_0 \ b \ q_1] \rightarrow c [q_1 \ b \ q_1]$$

Step 10 : Add productions for $\delta(q_1, a, a) = (q_1, \epsilon)$

$$[q_1 \ a \ q_1] \rightarrow a$$



Step 11 : Add productions for $\delta(q_1, b, b) = (q_1, \epsilon)$

$$[q_1 \ x \ q_1] \rightarrow b$$

Step 12 : Add production for $\delta(q_1, \epsilon, z) = (q_1, \epsilon)$

$$[q_1 \ x \ q_1] \rightarrow \epsilon$$

Step 13 : Add productions for $\delta(q_0, b, z) = (q_0, bz)$

$$[q_0 \ x \ q_0] \rightarrow b[q_0 \ x \ q_0] [q_0 \ x \ q_0]$$

$$[q_0 \ x \ q_0] \rightarrow b[q_0 \ x \ q_1] [q_1 \ x \ q_0]$$

$$[q_0 \ x \ q_1] \rightarrow b[q_0 \ x \ q_0] [q_0 \ x \ q_1]$$

$$[q_0 \ x \ q_1] \rightarrow b[q_0 \ x \ q_1] [q_1 \ x \ q_1]$$

Example 6.6.19 SPPU - May 12, 8 Marks

Obtain a PDA to accept the language

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

Solution :

Algorithm

- For every 'a' in input string, 2 X's are pushed.
- For every 'b' in input string, 1 X is popped.

$$\delta(q_0, a, Z_0) = (q_0, XXZ_0)$$

$$\delta(q_0, a, X) = (q_0, XXX)$$

$$\delta(q_0, b, X) = (q_1, \epsilon)$$

$$\delta(q_1, b, X) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon),$$

accept through an empty stack.

Z_0 – initial stack symbol

Example 6.6.20

For the PDA

$(\{q_0, q_1\}, \{0, 1\}, \{z_0, x\}, \delta, q_0, z_0, \phi)$ where δ is

$$\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$$

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

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

$$\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$$

$$\delta(q_1, 1, x) = \{(q_1, \epsilon)\}$$

$$\delta(q_1, 0, z_0) = \{(q_0, z_0)\}$$

Obtain CFG accepted by the above PDA and simplify the CFG and describe the language it accepts.

Solution :

Sr. No. PDA transition Corresponding productions

$$1. \text{ Productions due to start symbol } S. \quad S \xrightarrow{z_0} [q_0 \ x \ q_0]$$

$$S \xrightarrow{z_0} [q_0 \ x \ q_1]$$

| Sr. No. | PDA transition | Corresponding productions |
|---------|--|---|
| 2. | $\delta(q_0, 1, z_0) = (q_0, xz_0)$ | $[q_0 \ x \ q_0] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ q_0]$ |
| | | $[q_0 \ x \ q_0] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ q_0]$ |
| | | $[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ q_1]$ |
| | | $[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ q_0]$ |
| 3. | $\delta(q_0, 1, x) = (q_0, xx)$ | $[q_0 \ x \ q_0] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ q_0]$ |
| | | $[q_0 \ x \ q_0] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ q_0]$ |
| | | $[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_0] [q_0 \ x \ q_1]$ |
| | | $[q_0 \ x \ q_1] \rightarrow 1 [q_0 \ x \ q_1] [q_1 \ x \ q_1]$ |
| | $\delta(q_0, 0, x) = (q_1, x)$ | $[q_0 \ x \ q_0] \rightarrow 0 [q_1 \ x \ q_0]$ |
| | | $[q_0 \ x \ q_1] \rightarrow 0 [q_1 \ x \ q_1]$ |
| | $\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$ | $[q_0 \ x \ q_0] \rightarrow \epsilon$ |
| | $\delta(q_1, 1, x) = (q_1, \epsilon)$ | $[q_1 \ x \ q_1] \rightarrow 1$ |
| | $\delta(q_1, 0, z_0) = (q_1, z_0)$ | $[q_1 \ x \ q_0] \rightarrow 0 [q_0 \ x \ q_0]$ |
| | | $[q_1 \ x \ q_1] \rightarrow 0 [q_0 \ x \ q_1]$ |

Simplification of grammar

We can rename the variables as given below.

$$[q_0 \ x \ q_0] - A, [q_0 \ x \ q_1] - B, [q_1 \ x \ q_0] - C, [q_1 \ x \ q_1] - D$$

$$[q_0 \ x \ q_0] - E, [q_0 \ x \ q_1] - F, [q_1 \ x \ q_0] - G, [q_1 \ x \ q_1] - H$$

With the above substitutions, the resulting set of productions can be written as :

$$S \rightarrow A \mid B$$

$$A \rightarrow 1EA \mid 1FC$$

$$B \rightarrow 1EB \mid 1FD$$

$$E \rightarrow 1EE \mid 1FG$$

$$F \rightarrow 1EF \mid 1FH$$

$$E \rightarrow 0G$$

$$F \rightarrow 0H$$

$$A \rightarrow \epsilon$$

$$H \rightarrow 1$$

$$C \rightarrow 0A$$

$$D \rightarrow 0B$$

1. Removing ϵ -production.

$$\text{Nullable set} = \{S, A\}$$

ϵ -productions are removed with resulting set of productions as given below :

$$S \rightarrow A \mid B$$

$$A \rightarrow 1EA \mid 1FC \mid 1E$$

$$B \rightarrow 1EB \mid 1FD$$

- $C \rightarrow 0A10$
 $D \rightarrow 0B$
 $E \rightarrow 1EE \mid 1FG \mid 0G$
 $F \rightarrow 1EH \mid 1FH \mid 0H$
 $H \rightarrow 0$
2. Removing non-generating symbols

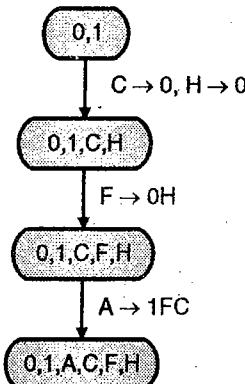


Fig. Ex. 6.6.20

Following symbols are non-generating

$$= (B, D, E, G)$$

Set of productions after elimination of non-generating symbols :

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow 1FC \\ C &\rightarrow 0A10 \\ F &\rightarrow 1FH \mid 0H \\ H &\rightarrow 0 \end{aligned}$$

3. Unit production $S \rightarrow A$ is removed, the resulting set of productions :

$$\begin{aligned} S &\rightarrow 1FC \\ A &\rightarrow 1FC \\ C &\rightarrow 0A10 \\ F &\rightarrow 1FH \mid 0H \\ H &\rightarrow 0 \end{aligned}$$

The language accepted by the PDA is given by :

$$L = \{1^n 0^m 1^n \mid n \geq 1\} \cup \epsilon$$

Example 6.6.21

Show that if a language L is accepted by a PDA then there exists a CFG generating L .

Solution :

Proof

For simplicity, we will consider a normalized PDA with following properties :

1. There is a single final state q_f
2. It empties the stack before accepting

3. Each transition either pushes a symbol onto the stack or performs a **pop** operation, but not both.

A transition,

$$\delta(q_i, x, b) \Rightarrow (q_j, c) \quad [\text{pop } b \text{ and push } c]$$

can be replaced by couple of transitions.

1. $\delta(q_i, x, b) \Rightarrow \delta(q_{temp}, \epsilon)$
2. $\delta(q_{temp}, \epsilon, \epsilon) \Rightarrow \delta(q_j, c)$

A new state q_{temp} has been introduced.

Similarly, a transition that neither pushes nor pops anything can be replaced with two transitions :

1. Transition pushing a dummy stack symbol
2. Transition popping the dummy stack symbol.

Let us consider a normalized PDA,

$$N = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F),$$

where $F = \{q_f\}$

- A word w will be accepted by N if it starts in q_0 and ends up in q_f with an empty stack.
- The language $L_{p, q}$, where $p, q \in Q$, is defined as consisting of those strings that start at state p with an empty stack and ends up in q with an empty stack.

Example 6.6.22

Describe the language $L(M)$ in English for push down automation.

$$M = (K, \Sigma, \Gamma, \Delta, S, F)$$

where $K = \{S, F\}$, $\Gamma = \{F\}$, $\Sigma = \{a, b\}$, $T = \{a\}$

$$\Delta = \{((S, a, \epsilon), (S, a)), ((S, b, \epsilon), (S, a)), ((S, a, \epsilon), (F, \epsilon)), ((F, a, a), (F, \epsilon)), ((F, b, a), (F, \epsilon))\}$$

Solution :

- It recognizes a string of the form $(a + b)^m a (a + b)^n \mid n \leq m$ and $n, m \geq 0$
- The middle character 'a' is determined non-deterministically.
- For every symbol from the first $(a + b)^m$ of $(a + b)^m a (a + b)^n$ an 'a' is pushed onto the stack using the given two moves.
 - $((S, a, \epsilon), (S, a)), ((S, b, \epsilon), (S, a))$
- The middle a of the input string $(a + b)^m a (a + b)^n$ takes the PDA to state F. 'a' is fixed non-deterministically.
- For every symbol from $(a + b)^n$ of $(a + b)^m a (a + b)^n$ an a is erased from the stack in state F.



Syllabus Topic : Deterministic PDA Vs Nondeterministic PDA

6.7 Deterministic Push Down Automata (DPDA)

SPPU - May 12

University Question

Q. Explain with example DPDA. (May 2012, 4 Marks)

In a DPDA there is only one move in every situation. A DPDA is less powerful than NPDA. Every context free language cannot be accepted by a DPDA. For example, a string of the form ww^R can not be processed by a DPDA. The class of a language a DPDA can accept lies in between a regular language and CFL.

A DPDA is defined as :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F), \text{ where}$$

$\delta(q, a, x)$ has one move for any $q \in Q, X \in \Gamma$ and $a \in \Sigma$.

6.7.1 Regular Language and DPDA

We can always design a DPDA for a regular language. A DPDA can be designed for regular language in two steps :

Step 1 : Construct an equivalent DFA for the given regular language.

Step 2 : For every transition $\delta(q_i, a) = q_j$ in FA (where $q_i, q_j \in Q$ and $a \in \Sigma$), we can write an equivalent transition for DPDA.

$$\delta(q_i, a, z_0) = \{(q_j, z_0)\}.$$

The move for PDA involves neither a Push nor a Pop operation.

Example 6.7.1

Design a DPDA for a binary number divisible by 3.

Solution :

Step 1 : Construction of DFA.

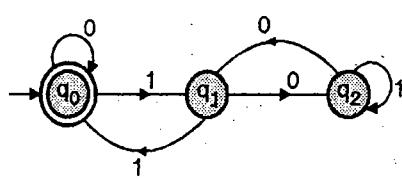


Fig. Ex. 6.7.1 : Construction of DFA

Step 2 : The DPDA is given by :

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$$

where the transition function δ is :

$$\delta(q_0, 0, z_0) = (q_0, z_0)$$

$$\begin{aligned}\delta(q_0, 1, z_0) &= (q_1, z_0) \\ \delta(q_1, 0, z_0) &= (q_2, z_0) \\ \delta(q_1, 1, z_0) &= (q_0, z_0) \\ \delta(q_2, 0, z_0) &= (q_1, z_0) \\ \delta(q_2, 1, z_0) &= (q_2, z_0)\end{aligned}$$

Example 6.7.2

Show that if L is accepted by a PDA in which no symbols are removed from the stack, then L is regular.

Solution :

Every regular language is accepted by some FA. Every transition of an FA can be converted into a PDA move without any push or pop operation.

Let there be a transition $\delta(q_i, a) = q_j$ in the FA. Where,

$$q_i, q_j \in Q \quad \text{and} \quad a \in \Sigma.$$

The FA move $\delta(q_i, a) = q_j$ can be converted into an equivalent PDA move as given below :

$$\delta(q_i, a, z_0) = \{(q_j, z_0)\}$$

This move for the PDA involves neither a Push nor a Pop operation.

Example 6.7.3

Prove "Let L be a language accepted by deterministic PDA, then the complement of L , can also be accepted by a DPDA".

Solution :

Let the DPDA for the given language L is

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

We can construct M' from M , such that M' accepts L' .

$$M' = (Q, \Sigma, \Gamma, \delta, q_0, z_0, Q-F)$$

i.e. an accepting state in M becomes a non-accepting state in M' and a non-accepting state in M becomes an accepting state in M' .

Proof that M' accepts L'

Case I : Let us take a string $\omega \in L$.

On application of ω , the machine M will reach a final state.

$$(q_0, \omega, z_0) \xrightarrow[M]{*} (p, -), \text{ where } p \in F.$$

Since the state of $p \notin Q-F$, it will not be accepted by M' .

Case II : Let us take a string $\omega \in \Sigma^*$ but $\omega \notin L$.

i.e. $\omega \in L'$.

On application of ω , the machine M will reach a non-final state.



$$(q_0, \omega, z_0) \xrightarrow[M]{*} (r, -), \text{ where } r \notin F.$$

Since, the state $r \in Q - F$, it will be accepted by M' .

Example 6.7.4 | SPPU - Dec. 12, 4 Marks

Enlist the difference between PDM and FSM.

Solution : Difference

1. PDM is more powerful than FSM.
2. PDM has additional memory in the form of a stack.
3. PDM can handle CFL but FSM can not handle CFL.

6.8 Application of PDA

SPPU - Dec. 12, Dec. 14, May 16

University Question

Q. What are the applications of PDA ?

(Dec. 2012, Dec. 2014, May 2016, 4 Marks)

PDA is a machine for CFL. A string belonging to a CFL can be recognized by a PDA. PDA is extensively used for parsing. PDA is an abstract machine; it can also be used for giving proofs of lemma on CFL.

Syllabus Topic : Pumping Lemma for CFL.

6.9 Pumping Lemma for CFG

SPPU - May 12, Dec. 15

University Question

Q. State and prove Pumping lemma theorem for Context-Free Languages.

(May 2012, Dec. 2015, 8 Marks)

Let G be a context free grammar. Then there exists a constant n such that any string $w \in L(G)$ with $|w| \geq n$ can be rewritten as $w = uvxyz$, subject to the following conditions :

1. $|vxy| \leq n$, the middle portion is less than n .
2. $vy \neq \epsilon$, strings v and y will be pumped.
3. For all $i \geq 0$, $uv^i xy^i z$ is in L . The two strings v and y can be pumped zero or more times.

Proof

Let us assume that the grammar

G is given by (V, T, P, S) .

$\phi(G)$ denotes that largest number of symbols on the right-hand side of a production in P .

In pumping lemma, it is a requirement that the constant n should satisfy the following condition :

$$n \geq \phi(G)^{1^{V-T}}$$

Let us take a string $w \in L(G)$, such that $|w| \geq n$. Let us construct a parse tree T with root as S . The parse tree T generates w with smallest number of leaves.

The tree T will have a path length of at least $|V - T| + 1$. This path will have $|V - T| + 2$ nodes with the last node labelled as terminal and remaining non-terminals.

Fig. 6.9.1 shows paths in detail.

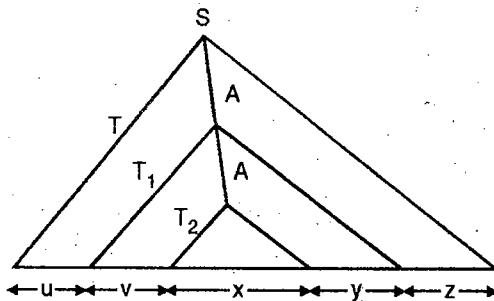


Fig. 6.9.1 : Paths in the parse tree

- x is generated by T_2
- v is generated by T_1
- u is generated by T

T_1 excluding T_2 can be repeated any number of times.

This will yield a string of the form $uv^i xy^i z$ where $i \geq 0$.

Example 6.9.1 | SPPU - May 12, 8 Marks

Prove that the language

$L = \{a^p \mid p \text{ is a prime}\}$ is not context free language.

Solution :

1. Let us assume that L is a CFL.
2. Let n be the natural number for L , as per the pumping lemma.
3. Let p be a prime number greater than n . Then $z = a^p \in L$. We can write $z = uvxyz$.
4. By pumping lemma $uv^0 xy^0 z = uxz \in L$. Therefore, $|uxz|$ is a prime number.

Let us assume that $|uxz| = q$.

Now, let us consider a string $uv^q xy^q z$,

The length of $uv^q xy^q z$ is given by :

$|uv^q xy^q z| = q + q(|v| + |y|)$, which is not a prime with q is a factor.

Thus $uv^q xy^q z \notin L$. This is a contradiction.

Therefore, L is not a context free language.

Example 6.9.2 | SPPU - Dec. 12, 8 Marks

Prove that $L = \{abc \mid i \geq 1\}$ is not a CFL.

Solution :

1. Let us assume that L is CFL.
2. Let us pick up a word $w = a^n b^n c^n$ where the constant n is given as per the pumping lemma.



3. w is rewritten as $uvxyz$.
where $|vxy| \leq n$ and $v \cdot y \neq \epsilon$ i.e., both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then $uv^i xy^i z$ is in $L(G)$ for each $i = 0, 1, 2, \dots$

There are two cases :

Case I : vy contains all three symbols a, b and c .

If vy contains all three symbols a, b and c then either v or y contains two symbols.

The exact ordering of a, b and c will be broken in $uv^2 xy^2 z$ and hence

$$uv^2 xy^2 z \notin L(G)$$

Case II : If vy does not contain three symbols a, b and c then $uv^2 xy^2 z$ will have unequal number of a 's, b 's and c 's and hence $uv^2 xy^2 z \notin L(G)$.

Hence, it is proved by contradiction.

Example 6.9.3

Prove that $L = \{a^i b^j c^l | j \geq i\}$ is not a CFL.

Solution :

1. Let us assume that L is CFL.
2. Let us pick up a word $\omega = a^n b^n c^n$, where n is a constant given as per the pumping lemma.
3. ω is rewritten as $uvxyz$ where, $|vxy| \leq n$ and $v \cdot y \neq \epsilon$, both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then $uv^i xy^i z$ is in $L(G)$ for each $i = 0, 1, 2, \dots$

There are two cases :

Case I : vy contains all three symbols a, b and c . If vy contains all three symbols a, b , and c then either v or y contains two symbols. The exact ordering of a, b and c will be broken in $uv^2 xy^2 z$ and hence $uv^2 xy^2 z \notin L(G)$.

Case II : If vy does not contain three symbols a, b and c then $uv^2 xy^2 z$ will have either :

- Unequal number of a and b
- Count of either a or b can be increased from the count of c .

Hence, proved by contradiction.

Example 6.9.4

$A = \{a^{n^2} | n \geq 1\}$ is context free. If so, enumerate some members of the equivalent CFL.

Solution :

$$L = \{a^n | n \geq 1\}$$
 is not a context free language

Proof that L is not a CFL

It can be proved using a contradiction. Let us assume that L is a context free language.

1. Let n be the constant as per the pumping lemma.
2. Let us choose a word $w = a^n$
3. w is rewritten as $uvxyz$.
Where $|vxy| \leq n$ and $v \cdot y \neq \epsilon$, both v and y are not null.

$$\begin{aligned}|uvxyz| &= |a^n| = n^2 \\ |uv^2 xy^2 z| &= |uvxyz| + |vy| > n^2 \\ \text{and } |uv^2 xy^2 z| &\leq n^2 + n \quad [\text{as } |vy| \leq n] \\ &\leq n^2 + 2n + 1 \\ &\leq (n+1)^2\end{aligned}$$

$|uv^2 xy^2 z|$ is a square for every i , $uv^i xy^i z \in L$. But there is no square between n^2 and $(n+1)^2$. This is a contradiction. Therefore, L is not a CFL.

Example 6.9.5

Prove that the following language is not a context free language.

$$L = \{a^n b^m c^p | 0 \leq n < m < p\}$$

Solution :

1. Let us assume that $L = \{a^n b^m c^p | 0 \leq n < m < p\}$ is CFL.
2. Let us pick up a word $\omega = a^n b^{n+1} c^{n+2}$, where the constant n is given as per the pumping lemma.
3. ω is rewritten as $uvxyz$, where $|vxy| \leq n$ and $v \cdot y \neq \epsilon$, both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then $uv^i xy^i z$ is in $L(G)$ for each $i = 0, 1, 2, \dots$

There are two cases :

Case I : vy contains all three symbols a, b and c . If vy contains all three symbols a, b and c , then either v or y contains two symbols the exact ordering of a, b and c will be broken in $uv^2 xy^2 z$ and hence $uv^2 xy^2 z \notin L(G)$.

Case II : If vy does not contain three symbols a, b and c then the count of the missing symbols/symbol will be less than or equal to the other symbols/symbol in $uv^2 xy^2 z$. Similarly, the count of the missing symbols/symbol will be more than or equal to the other symbol/symbols in $uv^0 xy^0 z$. Thus the sequence $n < m < p$ in $a^n b^m c^p$ can be made to violate either for $i = 0$ or $i = 2$ in $uv^i xy^i z$.

Hence, it is proved by contradiction.

Example 6.9.6

Prove that the language

$$L = \{\omega\omega\omega \text{ is in } (0+1)^*\}$$
 is not a CFL.

**Solution :**

1. Let us assume that $L = \{\omega\omega\omega \text{ is in } (0+1)^*\}$ is a CFL.
2. Let us pick up a word $\omega = 0^n 1^n 0^n 1^n$, where the constant n is given as per the pumping lemma.
3. ω is rewritten as $uvxyz$, when $|vxy| \leq n$ and $v, y \neq \epsilon$.
4. From pumping lemma, if $uvxyz \in L$, then $uv^i xy^i z$ is in L for each $i = 0, 1, 2, \dots$
5. vxy must be in one of the following forms :
 $0^j 1^j, 0^j 1^k, 1^j 0^k$

Case I : vxy is of the form 0^j - Two sets of 0's will be unequal in $uv^0 xy^0 z$.

Case II : vxy is of the form 1^j - two sets of 1's will be unequal in $uv^0 xy^0 z$.

Case III : vxy is of the form $0^j 1^k$ or $1^j 0^k$ - Two set of 0's or two sets of 1's will be unequal in $uv^0 xy^0 z$.

Thus the string $uv^0 xy^0 z$ does not belong to L. Hence, it is proved by contradiction.

Syllabus Topic : Closure Properties of CFLs**6.10 Properties of Context-free Languages**

In this section, we will consider some general properties of CFL. These properties can be classified in two groups :

1. Closure properties
2. Algorithmic properties

6.10.1 Closure Properties

SPPU - Dec. 13

University Question

- Q. State and explain any four closed properties of CFLs. (Dec. 2013, 8 Marks)

A context free language is closed under following operations :

1. Union
 2. Concatenation
 3. Kleene star
- Context free language is not closed under intersection.
 - The intersection of a context-free language with a regular language is a context free language.
 - The CFL is not closed under complementation.
 - The CFL is closed under reversal.

6.10.1.1 CFL is Closed under Union

Theorem : If L_1 and L_2 are context-free languages, then $L_1 \cup L_2$ is a context free language.

Proof : Let L_1 be a CFL. It is generated by a context free grammar $G_1 = (V_1, T_1, P_1, S_1)$.

Similarly, L_2 is another CFL generated by a context-free grammar $G_2 = (V_2, T_2, P_2, S_2)$.

We can combine the two grammars G_1 and G_2 into one grammar G that will generate the union of the two languages.

- A new start symbol S is added to G.
- Two new productions are added to G.

$$S \rightarrow S_1$$

$$S \rightarrow S_2$$

The grammar G can be written as :

$$G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$$

S can generate a string of terminals either by selecting start symbol S_1 of G_1 or start symbol S_2 of G_2 . Thus, S can generate a string from L_1 or from L_2 .

$$\therefore L(G) = L_1 \cup L_2$$

6.10.1.2 CFL is Closed under Concatenation

Theorem : If L_1 and L_2 are context-free languages, then $L_1 L_2$ is a context-free language.

Proof :

Let L_1 be a CFL with the grammar

$$G_1 = (V_1, T_1, P_1, S_1)$$

Let L_2 be a CFL with the grammar

$$G_2 = (V_2, T_2, P_2, S_2)$$

A new language L is constructed by combining the two grammars G_1 and G_2 into one grammar G that will generate the concatenation of the two languages.

- A new start symbol S is added to G.
- A new production is added to G.

$$S \rightarrow S_1 S_2$$

The start symbol S will generate a string w of the form :

$$w = w_1 w_2, \text{ where } w_1 \in L_1 \text{ and } w_2 \in L_2$$

The grammar G can be written as :

$$G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S)$$



6.10.1.3 CFL is Closed under Kleene Star

Theorem : If L is a context-free language, then L^* is a context-free language.

Proof :

Let L_1 be a CFL with the grammar

$$G_1 = (V_1, T_1, P_1, S_1)$$

A new language L is constructed from L_1 , which is L_1^* .

$$\text{i.e. } L = L_1^*$$

- A new start symbol S is added to the grammar G of L.
- Two new productions are added to G.

$$S \rightarrow SS_1$$

$$S \rightarrow \epsilon$$

The production $S \rightarrow SS_1 \mid \epsilon$ will generate a string w^* where $w \in L_1$.

The grammar G can be written as :

$$G = (V_1, T_1, P_1 \cup \{S \rightarrow SS_1 \mid \epsilon\}, S)$$

6.10.1.4 CFL is not Closed under Intersection

Theorem

Context-free languages are not closed under intersection.

Proof : Let us consider two context-free languages L_1 and L_2 .

Where,

$$L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$$

$$L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$$

The language L_1 is a CFL with set of productions given below :

$$S \rightarrow AB$$

$$A \rightarrow aAb \mid \epsilon$$

$$B \rightarrow cB \mid \epsilon$$

The language L_2 is a CFL with set of productions given below :

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bBc \mid \epsilon$$

- A string $w_1 \in L_1$ contains equal number of a's and b's.
- A string $w_2 \in L_2$ contains equal number of b's and c's.

A string $w \in L_1 \cap L_2$ will contain equal number of a's and b's and equal number of b's and c's.

Thus, $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$. From pumping lemma for CFL, a string of the form $a^n b^n c^n$ can not be generated by a CFG.

Therefore, the class of context-free languages is not closed under intersection.

6.10.1.5 CFL is not Closed under Complementation

Theorem

The set of context-free languages is not closed under complementation.

Proof

This theorem can be proved through contradiction.

Let us assume that CFL is closed under complementation.

If L_1 is context-free then L_1' is also context-free.

If L_2 is context-free then L_2' is also context-free.

Now, $L_1 \cap L_2$ can be written as $(L_1' \cup L_2')$, which should also be a context-free.

Since, $L_1 \cap L_2$ is not guaranteed to be context-free, our assumption that CFL is closed under complementation is wrong.

6.10.1.6 Intersection of CFL and RL

Theorem

If L is a CFL and R is a regular language, then $R \cap L$ is a CFL.

Proof

Let us assume that L is accepted by a PDA

$$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, z_1, F_1)$$

and R is accepted by a FA

$$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$$

We can combine M_1 and M_2 into a single PDA $M = (Q, \Sigma, \Gamma, \delta, q, z, F)$. The PDA M will accept a string w if it is accepted by the PDA M_1 and FA M_2 both executing in parallel.

The construction of M is given below :

$Q = Q_1 \times Q_2$, the Cartesian product of states of M_1 and M_2

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$\Gamma = \Gamma_1$$

$$q = (q_1, q_2)$$

$$z = z_1$$

$$F = F_1 \times F_2$$

The transition function δ is defined as :

$$\delta((q_1, q_2), u, \beta) = ((p_1, p_2), y) \quad [\text{transition for } M]$$

if and only if

$$\delta_1(q_1, u, \beta) = (p_1, y) \quad [\text{transition for } M_1]$$

$$\text{and } (q_2, u) \xrightarrow{*} (p_2, \epsilon)$$

- When M passes from state (q_1, q_2) to state (p_1, p_2) , M_1 passes from state q_1 to p_1 .
- Since, M_2 will read one symbol at a time, it requires $|u|$ steps to reach the state p_2 from q_2 .

Thus M is a PDA for intersection of $L(M_1)$ and $L(M_2)$.

$$L(M) = L(M_1) \cap L(M_2)$$

6.10.1.7 CFL is Closed under Reversal

Theorem : If L is a context-free language, then so is L^R .

Proof : Let us assume that $L = L(G)$ for some context-free grammar $G = (V, T, P, S)$

A grammar generating reverse L is given by

$$G^R = (V, T, P^R, S)$$

P^R can be obtained from P by reversing the right hand side of the production.

If $A \rightarrow \alpha$ is a production in P then

$$A \rightarrow \alpha^R \text{ is a production in } P^R.$$

6.10.2 Algorithmic Properties

There are algorithms for testing :

1. Whether the language generated by a CFG is empty i.e. Is $L(G) = \emptyset$?
2. Given a grammar G and a string w , is $w \in L(G)$?

The method of finding whether a string w belongs to $L(G)$ is known as parsing. There are two types of parsing :

1. Top-down parsing
2. Bottom-up parsing

In top-down parsing, we try to derive w , starting from the start symbol S .

In bottom up parsing, we try to reduce the word w into the start symbol S .

Syllabus Topic : Post Machine - Definition and Construction

6.11 Post Machine

SPPU - Dec. 12, May 16

University Questions

Q. Explain the concept of Post machine.
(Dec. 2012, 4 Marks)

Q. Write short note on post machine with example.
(May 2016, 6 Marks)

Informally, a post machine can be viewed as finite automata with a queue. An added queue provides memory and increases the capability of the machine.

- A post machine is deterministic
- It uses a queue instead of a stack.
- Input is always assumed to be loaded in the queue.
- A special symbol z_0 (say) is the marker and it is stored at the rear end of the queue along with the input.
- Items can be added only to the rear of the queue, and deleted only from the front.
- A single move in post machine depends on :
 1. State of the machine.
 2. Symbol at the front of the queue.
- A move in post machine has three components :
 1. The next state.
 2. Whether or not to remove the current symbol from the front of the queue.
 3. What to add to the rear of the queue, including the null.
- A post machine is more powerful than PDA as a stack can be simulated using a queue but a queue can not be simulated using a stack. A model of post machine is shown in Fig. 6.11.1.

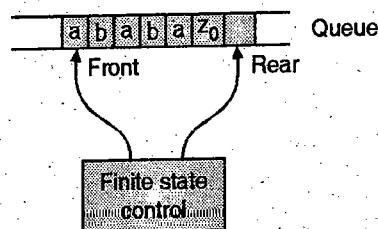


Fig. 6.11.1 : Model of post machine

6.11.1 Definition of Post Machine

SPPU - Dec. 13, Dec. 14, May 15, Dec. 15

University Questions

Q. Define post machine. Explain its elements.
(Dec. 2013, Dec. 2014, 4 Marks)

Q. Define Post machine with suitable example.
(May 2015, Dec. 2015, 3 Marks)

A post machine is defined as 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where, Q = The set of states

Σ = input alphabet

Γ = queue symbols; $\Sigma \subseteq \Gamma$

δ = The transition function

$q_0 = q_0 \in Q$ is the initial state



z_0 = A special symbol, such that $z_0 \notin \Sigma$.

It is used to mark the rear end of the input string initially stored in the queue. An empty queue contains the only symbol z_0 .

$F = F \subseteq Q$ is the set of final states

The transition function in detail

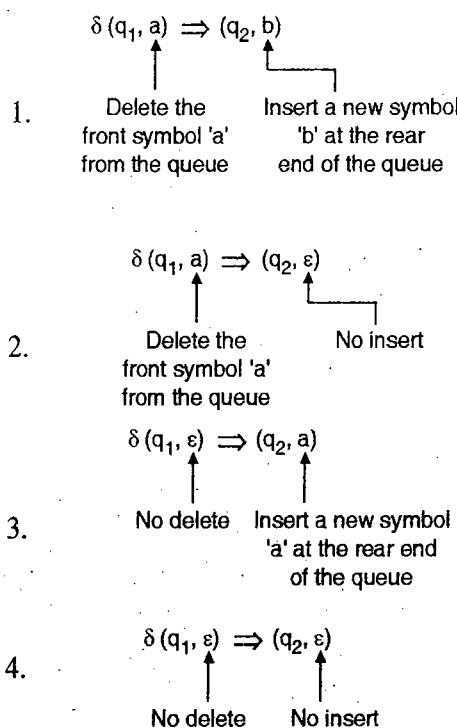
A transition in post machine is based on :

1. Current state
2. Front symbol of the queue.

A transition in post machine means :

1. A new state, it could be any state belonging to Q .
2. An insert without delete operation.
3. In insert with delete operation.
4. A delete without insert operation.

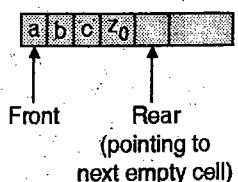
In general, we can have following types of transitions :



Queue can act like a stack

A queue can be made to behave like a stack by simulating `insert_at_the_front` operation using queue operations. A stack allows both deletion and insertion, only at the front end.

Suppose, there is a queue with the following data in it,



A symbol x can be inserted at the front using the following operations :

| Sr. No. | Operation | Queue |
|---------|---|-----------------------------|
| 1. | Initially | a b c z_0 Front Rear |
| 2. | Insert a new data x using the transition $\delta(q_0, \epsilon) \Rightarrow (q_0, x)$ | a b c $z_0 x$ Front Rear |
| 3. | Delete 'a' and then insert 'a' using the transition $\delta(q_0, a) \Rightarrow (q_0, a)$ | b c $z_0 x a$ Front Rear |
| 4. | Delete 'b' and then insert 'b' using the transition $\delta(q_0, b) \Rightarrow (q_0, b)$ | c $z_0 x a b$ Front Rear |
| 5. | Delete 'c' and then insert 'c' using the transition $\delta(q_0, c) \Rightarrow (q_0, c)$ | $z_0 x a b c$ Front Rear |
| 6. | Delete ' z_0 ' and then insert ' z_0 ' and also enter a final state (halting state) $\delta(q_0, z_0) \Rightarrow (q_f, z_0)$ | x a b c z_0 Front Rear |

Example 6.11.1

Construct a post machine for the language

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

Solution : Let us try to understand the design process with the help of the input string $a^3 b^3$.

There are three a's and three b's. Three a's will be matched with three b's in three cycles. Each cycle will involve :

1. Deletion of left most 'a'.
2. Deletion of left most 'b'.

These cycles of computation are shown in Fig. Ex. 6.11.1(a).

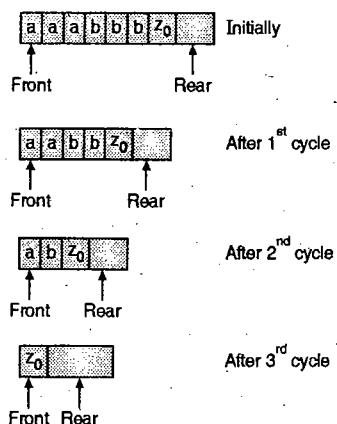


Fig. Ex. 6.11.1(a) : Cycles of computation for $a^3 b^3$

The post machine M is given by :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

| | |
|--------------------------------|--|
| $Q = \{ q_0, q_1, q_2, q_3 \}$ | q_0 is the initial state. z_0 is the special queue symbol. |
| $\Sigma = \{ a, b \}$ | $F = \{ q_3 \}$ |
| $\Gamma = \{ a, b, z_0 \}$ | δ = Transition function is given in Fig. Ex. 6.11.1(b), Fig. Ex. 6.11.1(c). |

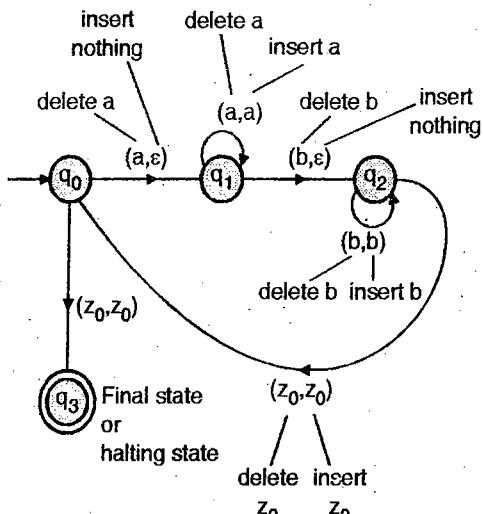


Fig. Ex. 6.11.1(b) : Transition diagram

| | a | b | Z_0 | \leftarrow | Front symbol (deleted) |
|-------------------|-----------------|-----------------|------------|--------------|------------------------|
| $\rightarrow q_0$ | q_1, ϵ | - | q_3, Z_0 | | |
| q_1 | q_1, a | q_2, ϵ | - | | Symbol inserted |
| q_2 | - | q_2, b | q_0, Z_0 | | |
| q_3 | q_3 | q_3 | q_3 | \leftarrow | Final state |

Fig. Ex. 6.11.1(c) : Transition table

- Transition from q_0 to q_1 is for deletion of the first 'a'.
- Transition from q_1 to q_2 is for deletion of the first 'b'.
- Failure state is not shown as it is a standard practice.

Example 6.11.2

Design a post machine which accepts the strings with an equal number of a's and b's.

Solution : Let us try to understand the design process with the help of the input string abbaba. There are three a's and three b's. Three cycles will be needed. In each cycle :

- If the front character is 'a' then it will delete the front 'a' and the first 'b'.

- If the front character is 'b' then it will delete front 'b' and the first 'a' from the queue.

These cycles of computations are shown in Fig. Ex. 6.11.2.

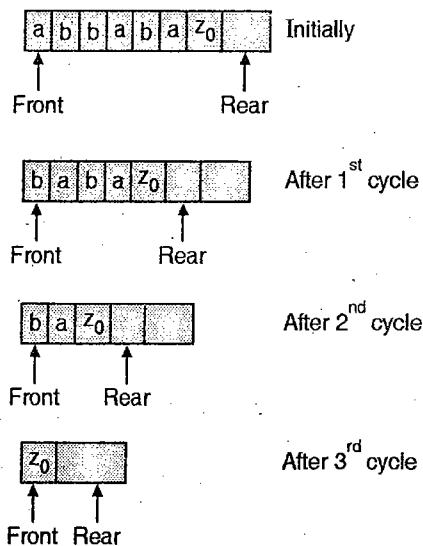


Fig. Ex. 6.11.2 : Cycles of computation for abbaba

The post machine M is given by :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

$$Q = \{ q_0, q_1, q_2, q_3, q_4 \}$$

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

$$\Gamma = \{ a, b, z_0 \}$$

$$q_0 = \text{initial state}$$

$$z_0 = \text{special queue symbol}$$

$$\delta = \text{Transition function is given in Fig. Ex. 6.11.2(a).}$$

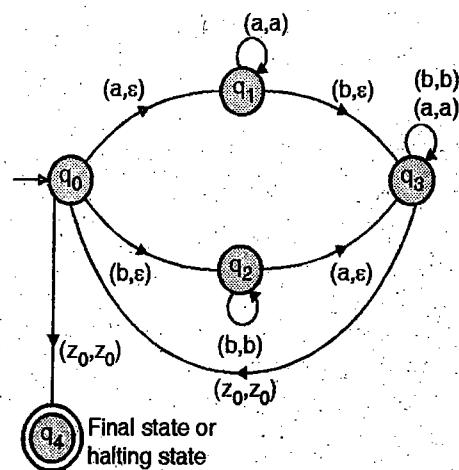


Fig. Ex. 6.11.2(a) : Transition diagram

- If the front character is 'a', machine moves through $q_0 \rightarrow q_1 \rightarrow q_3$. It deletes the front 'a', locates the first 'b' and deletes 'b'.
- If the front character is 'b', machine moves through $q_0 \rightarrow q_2 \rightarrow q_3$. It deletes the front 'b' locates the first 'a' and deletes 'a'.



- In the state q_3 , symbols from the queue are deleted and then inserted back until the deleted symbol is z_0 , restarting a new cycle.

Example 6.11.3 SPPU - Dec. 15, 9 Marks

Design a post machine to accept the language

$$L = \{a^n b^{n+1} \mid n \geq 0\}$$

Solution :

Post machine for $\{a^n b^{n+1} \mid n \geq 0\}$

The post machine M is given by :

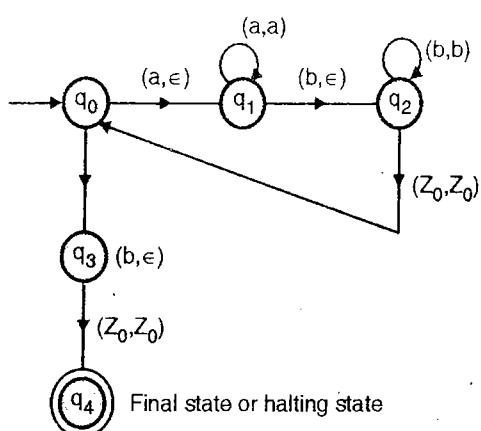


Fig. Ex. 6.11.3

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

z_0 is the special queue symbol

q_0 = initial state

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

$$\Gamma = \{a, b, z_0\}$$

- The loop from $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$ is for deletion of first 'a' and the first 'b' in the string the loop will be iterated n time.
- After all the a's and equal number of b's are deleted, the machine goes through the states q_3 and q_4 .

Example 6.11.4

Design the post machine which accepts the string over $\{a, b\}$ having odd length and the element at the centre as a.

Solution :

A string with centre character a can be represented as $(a+b)^n a (a+b)^n \mid n \geq 0$

- The algorithm involves deletion of two characters in each cycle :
 1. Character at the beginning of the queue.
 2. Character from the end of the queue.
- After n-cycles, the queue will be left with the only character 'a'.

- It is difficult to locate the last character of a queue. But it can still be deleted using the concept of look ahead i.e.

1. Remember the previous character. The state q_3 in Fig. Ex. 6.11.4 stands for the previous character as 'b' and the state q_4 stands for the previous character as 'a'.
2. Delete the next character, insert the previous character and then make the next character as previous character. This concept is handled in state q_3 and state q_4 .

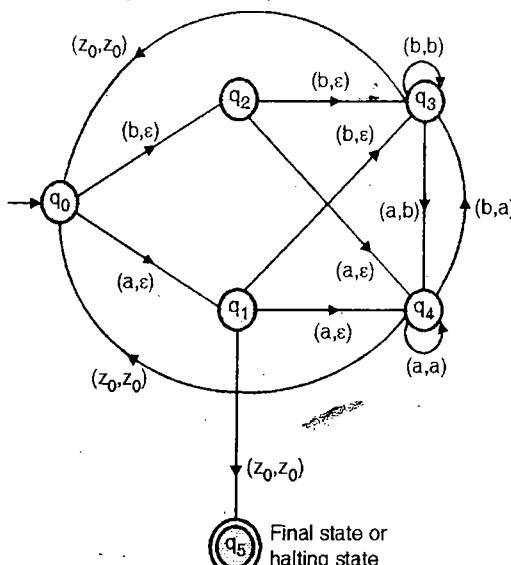


Fig. Ex. 6.11.4 : State transition diagram

3. A transition from q_3 to q_0 or q_4 to q_0 is for deletion of last character (by not inserting the last character).

The post machine M is given by :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

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

$$\Gamma = \{a, b, z_0\}$$

δ = Transition function is shown in Fig. Ex. 6.11.4.

q_0 = is the initial state

z_0 is the special queue symbol

$$F = \{q_5\}$$

The working of post machine is being simulated for the input abaab.

| Step No. | Contents of queue | State of the machine |
|----------|-------------------|----------------------|
| 1. | abaab z_0 | q_0 (initially) |
| 2. | baab z_0 | q_1 |



| Step No. | Contents of queue | State of the machine |
|----------|--------------------|-------------------------------------|
| 3. | aabz ₀ | q ₃ |
| 4. | abz ₀ b | q ₄ |
| 5. | bz ₀ ba | q ₄ |
| 6. | z ₀ baa | q ₃ |
| 7. | baaz ₀ | q ₀ |
| 8. | aaz ₀ | q ₂ |
| 9. | az ₀ | q ₄ |
| 10. | z ₀ a | q ₄ |
| 11. | az ₀ | q ₀ |
| 12. | z ₀ | q ₁ |
| 13. | z ₀ | q ₅ ← string is accepted |

Example 6.11.5

Design post machine to check well formedness of parentheses.

Solution :

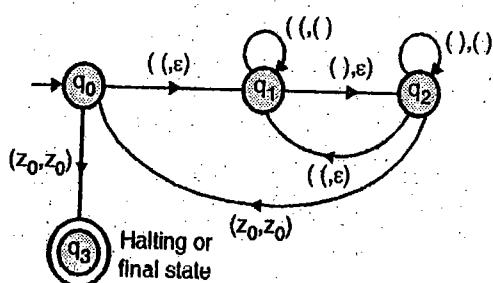


Fig. Ex. 6.11.5

- The algorithm involves deletion of a pair of opening and closing brackets in each cycle.
- The opening bracket '(' is deleted in state q_0 . The machine makes a move to state q_1 .
- All opening brackets before the nearest closing bracket are deleted and then inserted back in the queue in state q_1 .
- The nearest closing bracket ')' is deleted in state q_1 . The machine makes a move to state q_2 .
- The working of the post machine is being simulated for the input $((())()$.

| Step No. | Contents of the queue | State of the machine |
|----------|------------------------|--------------------------------|
| 3. |)())()z ₀ (| q ₁ |
| 4. | ()()()z ₀ (| q ₂ |
| 5. |)())z ₀ (| q ₁ |
| 6. |)()z ₀ (| q ₂ |
| 7. | ()z ₀ (| q ₂ |
| 8. |)z ₀ () | q ₁ |
| 9. | z ₀ () | q ₂ |
| 10. | ()z ₀ | q ₀ |
| 11. |)z ₀ | q ₁ |
| 12. | z ₀ | q ₂ |
| 13. | z ₀ | q ₀ |
| 14. | z ₀ | q ₃ ← Halting state |

The post machine $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

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

$\Gamma = \{(), z_0\}$,

δ = transition function specified through transition diagram.

q_0 = initial state,

z_0 = special queue symbol

$F = \{q_3\}$

Example 6.11.6 SPPU - Dec. 16, 6 Marks

Design PM for $L = \{a^n b^n c^n \mid n \geq 0\}$

Can you design a NPDA for the same ? Justify.

Solution :

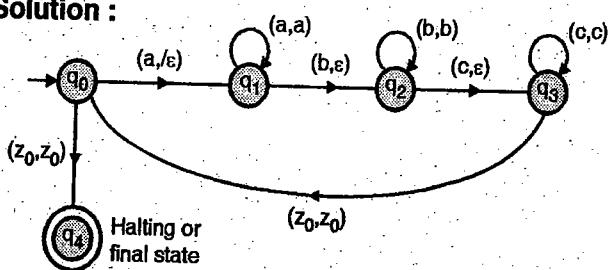


Fig. Ex. 6.11.6

The post machine $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_4\})$

δ is specified through the transition diagram.

The post machine will require n cycles to verify a string $a^n b^n c^n$.

| Step No. | Contents of the queue | State of the machine |
|----------|-----------------------|----------------------|
| 1. |)())()z ₀ | q ₀ |
| 2. |)()()z ₀ | q ₁ |



- The each cycle :
 1. Left most a is deleted
 2. Left most b is deleted
 3. Left most c is deleted
- After n-cycles, the queue will contain only one symbol z_0 .
- NPDA can not be designed. The language is not CFL.

Example 6.11.7

Draw a post machine that accepts even and odd palindrome.

Solution :

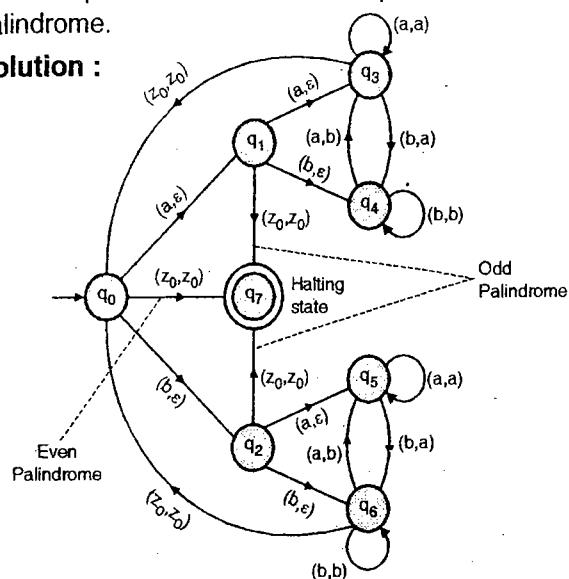


Fig. Ex. 6.11.7 : Transition Diagram

The post machine $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{\text{a, b}\}, \{\text{a, b, } z_0\}, \delta, q_0, z_0 \setminus \{q_7\})$

The transition function δ is specified through the transition diagram.

- If ω represents a palindrome of length > 1 then
 1. First and the last characters will be same.
 2. After deletion of 1st and the last characters from ω , ω will remain a palindrome.

Let us try to understand the design process with the help of the input ababa.

The machine will require 2 cycles :

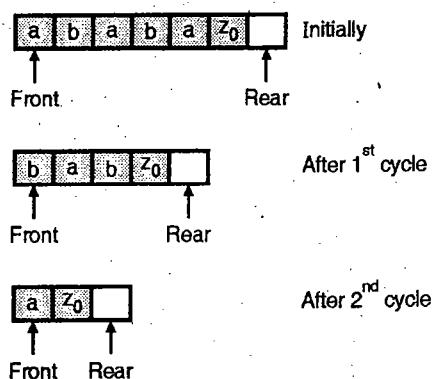


Fig. Ex. 6.11.7(a) : Machine cycles

- If the front character is 'a' then machine goes to state q_1 to ensure that the last character is $a \cdot *$
- If the front character is 'b' then machine goes to state q_2 to ensure that the last character is $b \cdot *$
- Machine uses the concept of look-ahead to locate the last character.
- Look-ahead is implemented by deleting the next character and inserting the previous character.

Example 6.11.8 SPPU - May 15, 6 Marks

Construct a post machine to accept the language $\{a^n b^{n+1} / \text{where } n >= 1\}$.

Solution : The post machine M is given by :

$$\begin{aligned} M &= (Q, \Sigma, \Gamma, \delta, q_0, z_0, F) \\ Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \\ z_0 &= z_0 \text{ is the special queue symbol} \\ q_0 &= \text{initial state} \\ \Sigma &= \{a, b\}, F = \{q_7\} \\ \Gamma &= \{a, b, z_0\} \end{aligned}$$

The transition diagram is as given Fig. Ex. 6.11.8 :

- States from q_0 to q_2 are for deletion of first 'a' and first 'b' in the string. It is necessary as the string is of the $a^n b^{n+1} \mid n \geq 1$.
- The cycle $q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_3$ deletes first 'a' and the first 'b' in the stack.
- After all the a's and equal number of b's are deleted, the machine goes through the states q_6 and q_7 .

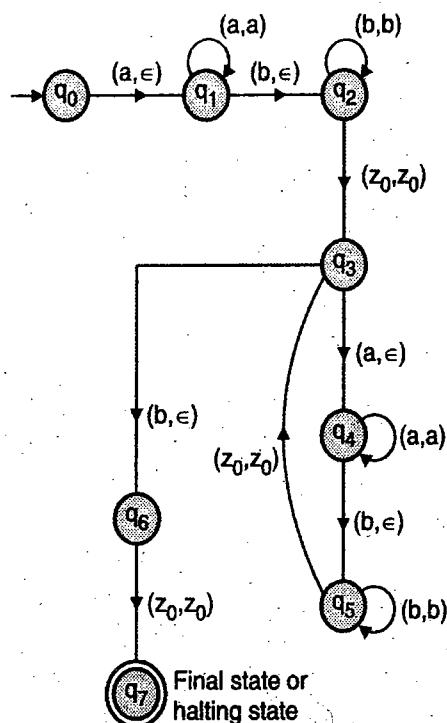


Fig. Ex. 6.11.8.

**Example 6.11.9 SPPU - May 15, 4 Marks**

Compare PDA and FA.

Solution :

| Sr. No. | | FA | PDA |
|---------|------------|--|---|
| 1. | Power | FA is less powerful than PDA | PDA is more powerful than FA. |
| 2. | Language | FA accepts regular language | PDA accepts context free language. |
| 3. | Memory | No memory | Memory in the form of a stack |
| 4. | Transition | Transition on the basis of current state and the input symbol. | Transition on the basis of current state, input symbol and top of the stack symbol. |
| 5. | Acceptance | Through final state. | Through empty stack or final state. |

Example 6.11.10 SPPU - Dec. 15, Dec. 16, 4 Marks

Compare PDA and post machine.

Solution :

| Sr. No. | PDA | Post machine |
|---------|---------------------------------|---------------------------------------|
| 1. | Memory in the form of stock | Memory in the form of queue. |
| 2. | PDA can not modify input string | Post machine can modify its own input |

| Sr. No. | PDA | Post machine |
|---------|--|---|
| 3. | PDA is less powerful than post machine | Post machine is more powerful than PDA and in fact it is as powerful as a turing machine. |
| 4. | CFL is the language of PDA | The language of post machine recursive language |

6.12 Power of Various Machines**SPPU - Dec. 14****University Question**

Q. Show that the post machine is more powerful than PDA. **(Dec. 2014, 8 Marks)**

- A post machine can simulate functioning of a turing machine. Hence a turing machine and a post machine have equal power.
- A post machine is more powerful than a PDA. A string of the form $a^n b^n c^n$ can be handled by a post machine but it can not be handled by a PDA.
- An NPDA is more powerful than a DPDA. A string of the form $\omega\omega^R$ can be handled by an NPDA but it cannot be handled by a DPDA.
- DPDA is more powerful than FA. A string of the form $a^n b^n$ can be handled by a DPDA but it cannot be handled by FA.



Turing Machines

Syllabus

Formal definition of a Turing machine, Recursive Languages and Recursively Enumerable Languages, Design of Turing machines, Variants of Turing Machines : Multi-tape Turing machines, Universal Turing Machine, Nondeterministic Turing machines. Comparisons of all automata.

7.1 Introduction to Turing Machine

SPPU - Dec. 13

University Question

Q. Compare pushdown automata and Turing machine.
(Dec. 2013, 2 Marks)

Turing machine is an example of computing machine. So far we have discussed three types of machines :

1. Finite state machine
2. Pushdown machine
3. Post machine

These machines have no control over the input and they can not modify their own inputs. Turing machine is a writing machine, it can modify its own input symbols. Turing machine is more powerful than a pushdown machine. Power of various machines is shown below :

$$\text{FA} \leq \text{DPDA} \leq \text{NPDA} \leq \text{Post machine} = \text{Turing machine}$$

Turing machine is capable of performing computations on inputs and producing a new result.

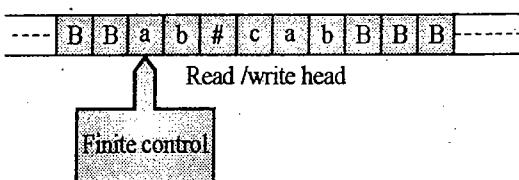


Fig. 7.1.1 : An example of a Turing Machine

An abstract model of a turing machine is shown in Fig. 7.1.1.

- Input to a turing machine is provided through a long tape.
- Turing machine is provided with a read / write head.
- The tape is divided into squares, each square holds a single symbol.
- Blank squares hold a special character 'B'.

- The head is capable of performing three operations :

1. Reading a symbol being scanned.
2. Modifying a symbol being scanned.
3. Shifting either to previous square (L) or next square (R).

Example 7.1.1

Design a turing machine to perform the following computation :

Initially the tape contains two finite blocks of 1's separated by finite block of blanks. The machine should delete the block of blanks between two blocks of 1's.

B | B | 1 | 1 | 1 | 1 | B | B | B | 1 | 1 | 1 | B

(a) Input tape

B | B | 1 | 1 | 1 | 1 | 1 | 1 | 1 | B

(b) Output tape

Fig. Ex. 7.1.1 : Input, Output tapes

Solution :

Let us call the first block of 1's as ω_1 and the second block of 1's as ω_2 .

There are three blanks between ω_1 and ω_2 . These three blanks will be deleted in three cycle.

Each cycle will consist of following steps :

Step 1 : Leftmost 1 of ω_1 is erased.

Step 2 : Head moves to first 'B' between ω_1 and ω_2 and replaces it by 1.

Step 3 : If some B's are yet to be deleted then head comes back to first 1 of ω_1 .

These cycles of computation are shown in Fig. Ex. 7.1.1(c).

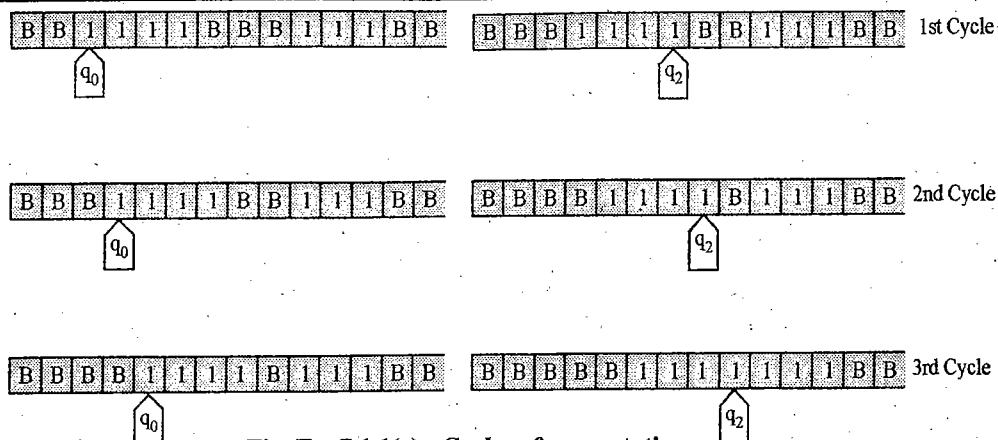


Fig. Ex. 7.1.1(c) : Cycles of computation

The transition behaviour of the turing machine is shown in Fig. Ex. 7.1.1(d).

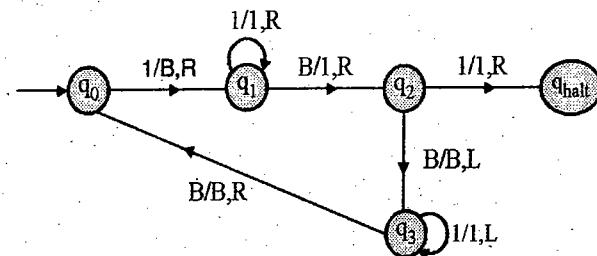


Fig. Ex. 7.1.1(d) : Transition diagram

Arc between q_0 and q_1 is marked as $(1/B, R)$. It implies that the square being scanned contains 1 and it will be replaced by B, then the head will move to right cell.

Legend:
 Input symbol $1/B, R$
 replacement symbol head moves to right cell
 $B/B, L$

Meaning of various transitions and their effect on input tape

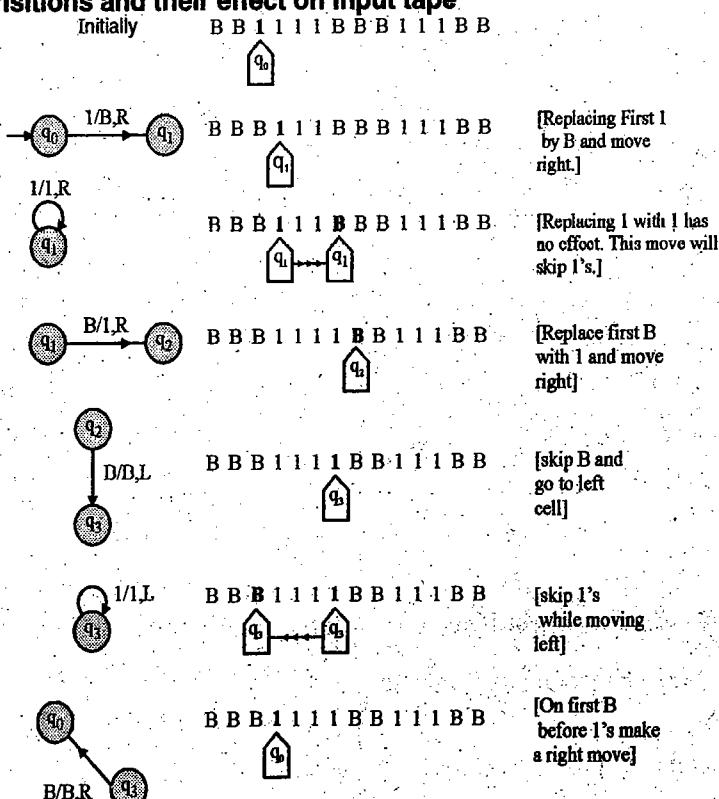


Fig. Ex. 7.1.1(e) : Transitions in detail for first cycle



7.1.1 Limitation of Turing Machine

Turing machine is widely accepted as a general model of computation. Besides accepting languages, they can compute functions and carry out any conceivable algorithmic procedure. The Turing machine model can accommodate the idea of a stored-program computer, so that the concept of "universal" machine executes any algorithm by giving it an input that includes an encoding of the algorithm.

- For a class of problem (Turing acceptable), the Turing machine can loop forever.
- There are unsolvable problems, which can be solved by a TM.

Syllabus Topic : Formal Definition of a Turing Machine, Design of Turing Machines

7.2 The Formal Definition of Turing Machine

A Turing machine M is a 7-tuple given by :

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

where

1. Q is finite set of states.
2. Σ is finite set of input alphabet not containing B.
3. Γ is a finite set of tape symbols. Tape symbols include B.
4. $q_0 \in Q$ is the initial symbol.
5. $B \in \Gamma$ is a special symbol representing an empty cell.
6. $F \subseteq Q$ is the set of final states, final states are also known as halting states.
7. The transition function δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, N\}$

A transition in turing machine is written as,

$\delta(q_0, a) = (q_1, b, R)$, which implies, when in state q_0 and scanning symbol a, the machine will enter state q_1 , it will rewrite a as b and move to the right cell.

A transition $\delta(q_0, a) = (q_1, a, R)$, implies that the machine will enter state q_1 , it will not change the symbol being scanned and move to the right cell.

Movement of Read / Write head is given L, R or N

L → Move to left cell

R → Move to right cell

N → Remain in the current cell
(No movement)

Example 7.2.1

Design a turing machine that erases all non-blank symbols on the tape, where the sequence of non-blank symbols does not contain any blank symbol B in-between.

Solution : The turing machine M is given by :

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

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

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

$$\Gamma = \{a, b, B\},$$

q_0 = is the initial state,

B = is blank symbol.

F = $\{q_1\}$ is a final state or 'Halt state'.

The transition function δ is defined below :

$$\delta(q_0, a) = (q_0, B, R) \quad [\text{erase 'a' and move right}]$$

$$\delta(q_0, b) = (q_0, B, R) \quad [\text{erase 'b' and move right}]$$

$$\delta(q_0, B) = (q_1, B, N) \quad [\text{stop in state } q_1]$$

The transition function can also be given in a tabular form as shown in Fig. Ex. 7.2.1(a).

| | a | b | B | |
|-------------------|---------------|---------------|---------------|----------------------------|
| $\rightarrow q_0$ | (q_0, B, R) | (q_0, B, R) | (q_1, B, N) | |
| q_1^* | q_1 | q_1 | q_1 | \leftarrow Halting state |

Fig. Ex. 7.2.1(a) : Transition table

The transition function can also be given as transition diagram as shown in Fig. Ex. 7.2.1(b).



Fig. Ex. 7.2.1(b) : Transition diagram



7.2.1 A String Accepted by TM

A string α over Σ is said to be accepted by a turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ if when the string α is placed on the tape and head is positioned on leftmost cell containing a non-blank cell, machine is started in state q_0 , then after a finite number of moves the machine is in a "Halt-state" $\in F$.

$$(q_0, \alpha) \xrightarrow[M]{*} \text{Halt-state}$$

A string is rejected by a TM if the machine enters a state $q \notin F$ and scans a symbol x for which $\delta(q, x)$ is not defined.

Example 7.2.2 SPPU - Dec. 14, 10 Marks

Design a TM which accepts all strings of the form $a^n b^n$ for $n \geq 1$ and rejects all other strings.

Solution :

Let us try to understand the design process with the help of the input string $a^3 b^3$.

There are three a's and three b's. Three a's will be matched with three b's in three cycles.

Each cycle will consist of following steps :

Step 1 : Leftmost a is changed to x.

Step 2 : Rightmost b is changed to y.

Step 3 : Head comes back to first a.

These cycles of computations are shown in Fig. Ex. 7.2.2(a).

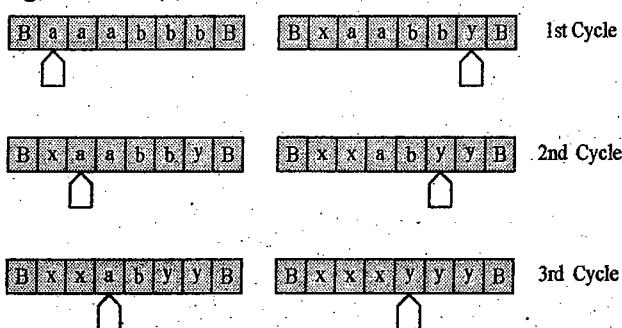


Fig. Ex. 7.2.2(a) : Cycles of computation for $a^3 b^3$

The turing machine M is given by :

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

Where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$$\Gamma = \{a, b, B\}$$

q_0 = initial state

B = blank symbol.

F = $\{q_4\}$ is a final state or 'Halt-state'

The transition function δ is given in Fig. Ex. 7.2.2(b).

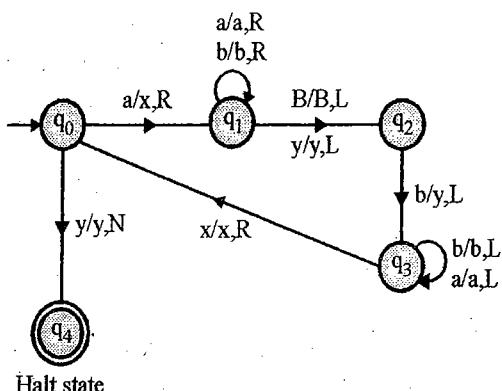


Fig. Ex. 7.2.2(b) : Transition diagram

| | a | b | x | y | B | |
|-------------------|---------------|---------------|---------------|---------------|---------------|-----------------|
| $\rightarrow q_0$ | (q_1, x, R) | - | - | (q_4, y, N) | - | |
| q_1 | (q_1, a, R) | (q_1, b, R) | - | (q_2, y, L) | (q_2, B, L) | |
| q_2 | - | (q_3, y, L) | - | - | - | |
| q_3 | (q_3, a, L) | (q_3, b, L) | (q_0, x, R) | - | - | |
| q_4^* | q_4 | q_4 | q_4 | q_4 | q_4 | ← Halting state |

Fig. Ex. 7.2.2(c) : Transition table

Meaning of Various States

q_0 – Leftmost a is replaced by x.

q_1 – State q_1 locates the last b by skipping a's and b's and taking a left turn on first B or y.

q_2 – Rightmost b is replaced by y.

q_3 – State q_3 locates the first a by skipping a's and b's while moving left and taking a right turn on first x from right side and entering state q_0 .

q_4 – It is halt state for a valid string.

Acceptance : In case of a valid string, the head will be exposed to y in state q_0 .

Rejections

1. An input symbol 'b' in state q_0 implies that number of b's are more than a's.
2. An input symbol 'a' in state q_2 implies that number of a's are more than b's.

**Example 7.2.3**

Draw a transition diagram for a turing machine accepting the following language. $L = \{ a^i b^j \mid i < j \}$

Solution : Solution to this example follows from Example 7.2.2. In Fig. Ex. 7.2.2(b), if the machine finds a string of only b's in state q_0 then number of a's is less than number of b's in $a^i b^j$.

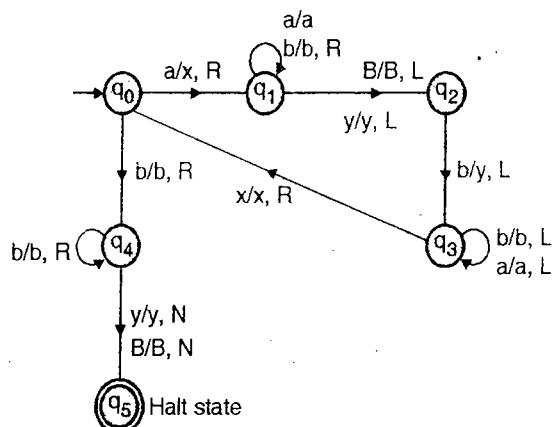


Fig. Ex. 7.2.3

7.2.2 Instantaneous Descriptions for Turing Machines

Processing of a string by a turing machine can be shown using the instantaneous description - An instantaneous description of a turing machine include :

1. The input string at any point of time
2. Position of head
3. State of the machine

The string $a_1 a_2 \dots a_{i-1} [q] a_i a_{i+1} \dots a_n$ gives the snapshot of the machine in which :

1. q is the state of the turing machine.
2. The head is scanning the symbol a_i .

The representation of machine with head scanning the symbol a_i and the machine in state q is shown in Fig. 7.2.1.

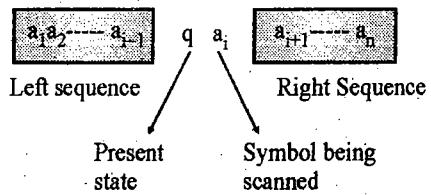


Fig. 7.2.1 : Instantaneous description of TM

Example 7.2.4

A turing machine is represented in Fig. Ex. 7.2.4 show the processing sequence for the input string aaabbb.

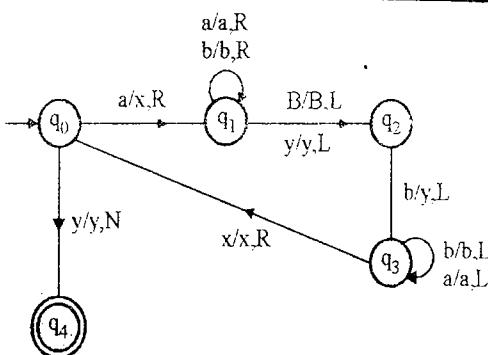


Fig. Ex. 7.2.4 : TM under consideration

Solution :

$\begin{array}{c} a \ a \ a \ b \ b \ b \ B \vdash x \ a \ a b \ b \ b \ B \vdash x \ a \ a b \ b \ b \ B \\ \uparrow \qquad \uparrow \qquad \uparrow \\ q_0 \qquad q_1 \qquad q_1 \end{array}$

 $\vdash x \ a \ a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_1 \qquad q_1 \qquad q_1$

$\vdash x \ a \ a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_1 \qquad q_2 \qquad q_3$

$\vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_3 \qquad q_3 \qquad q_3$

$\vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B \vdash x \ a a b \ b \ b \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_3 \qquad q_0 \qquad q_0$

$\vdash x x \ a b b \ y \ B \vdash x x a b b y \ B \vdash x x a b b y \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_1 \qquad q_1 \qquad q_1$

$\vdash x x a b b y \ B \vdash x x a b b y \ B \vdash x x a b b y \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_1 \qquad q_2 \qquad q_3$

$\vdash x x a b b y \ B \vdash x x a b b y \ B \vdash x x a b b y \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_3 \qquad q_3 \qquad q_0$

$\vdash x x x b y y \ B \vdash x x x b y y \ B \vdash x x x b y y \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_1 \qquad q_1 \qquad q_2$

$\vdash x x x y y y \ B \vdash x x x y y y \ B \vdash x x x y y y \ B$
 $\uparrow \qquad \uparrow \qquad \uparrow$
 $\qquad q_3 \qquad q_0 \qquad q_4$

[Accept]

Example 7.2.5

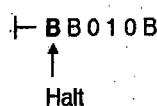
- Let T be the Turing Machine defined by the 5-tuples,
- $(S_0, 0, S_1, L)$
 - $(S_0, 1, S_0, L)$
 - $(S_0, B, B, Halt, L)$
 - $(S_1, 0, 1, S_0, L)$
 - $(S_1, 1, 1, S_0, R)$

For each of the following initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

- 1) 110B 2) 0011B 3) 0101B

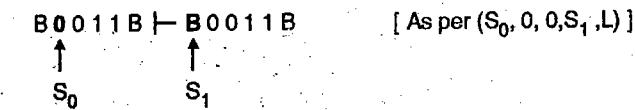
Solution :

1) 110B



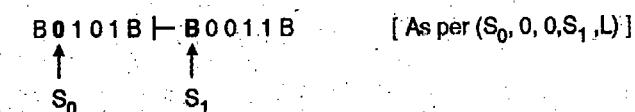
[If there is less than 2 blanks on the left then the tape head halt with a crash]

2) 0011B



The machine will fail at this point. There is no transition for input B in state S_1 .

3) 0101B



This machine will fail at this point. There is no transition for input B in state S_1 .

Example 7.2.6

Design a right shifting turing machine over alphabet $\{0,1\}$

Solution :

A right shifting turing machine will shift the input string, right by 1 place.

Algorithm

Right shifting should start from the rightmost character. Each character is shifted right starting from right end and working towards left end.

Cycles of computation for an input string 1010 are shown in Fig. Ex. 7.2.6 :

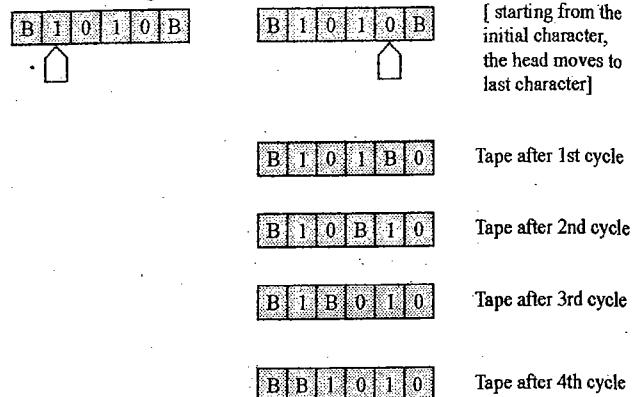


Fig. Ex. 7.2.6 : Cycles of computation

Turing machine is shown in Fig. Ex. 7.2.6(a).

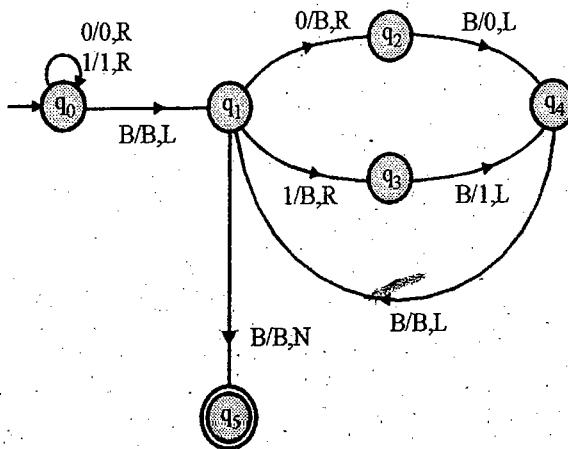


Fig. Ex. 7.2.6(a) : Transition diagram

| | 0 | 1 | B |
|-------------------|---------------|---------------|---------------|
| $\rightarrow q_0$ | $(q_0, 0, R)$ | $(q_0, 1, R)$ | (q_1, B, L) |
| q_1 | (q_2, B, R) | (q_3, B, R) | (q_5, B, N) |
| q_2 | - | - | $(q_4, 0, L)$ |
| q_3 | - | - | $(q_4, 1, L)$ |
| q_4 | - | - | (q_1, B, L) |
| q_5^* | q_5 | q_5 | q_5 |

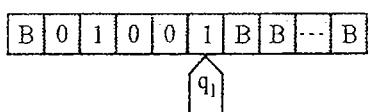
Fig. Ex. 7.2.6(b) : Transition table

Meaning of various states

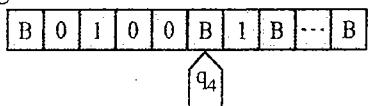
- Initial state q_0 is being used to skip the string of 0's and 1's so that the head can be positioned on last character.



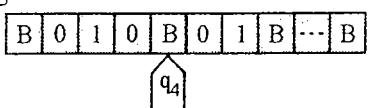
- On seeing a B (blank), the head takes a left turn and positions itself on the last character. Machine enters state q_1 .



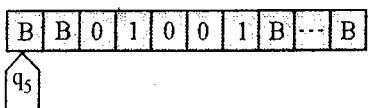
- A path through $q_1 \rightarrow q_3 \rightarrow q_4$ is for right shifting 1.



- A path through $q_1 \rightarrow q_2 \rightarrow q_4$ is for right shifting 0.



- Machine halts in state q_5 .



Example 7.2.7

Design a TM to make a copy of a string over $\{0,1\}$

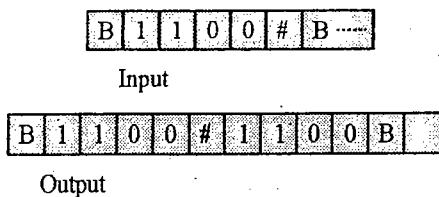


Fig. Ex. 7.2.7

Solution :

- Two copies of the strings are separated by #.
- Machine is described in Fig. Ex. 7.2.7(a).

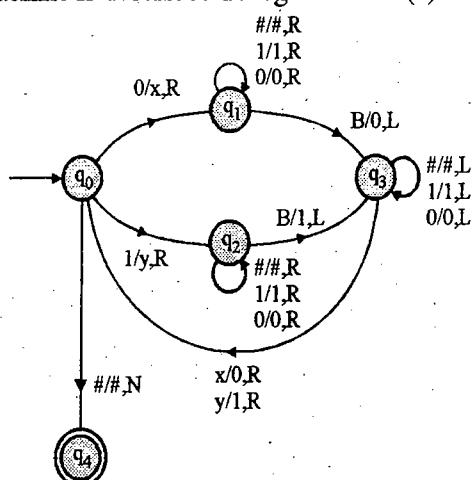


Fig. Ex. 7.2.7(a) : Transition diagram

| | 0 | 1 | # | B | x | y |
|------------------------------------|-----------------|-----------------|------------------|-----------------|-----------------|-----------------|
| $\rightarrow q_0$ | (q_1, x, R) | (q_2, y, R) | ($q_4, \#, N$) | - | - | - |
| q_1 | ($q_1, 0, R$) | ($q_1, 1, R$) | ($q_1, \#, R$) | ($q_3, 0, L$) | - | - |
| q_2 | ($q_2, 0, R$) | ($q_2, 1, R$) | ($q_2, \#, R$) | ($q_3, 1, L$) | - | - |
| q_3 | ($q_3, 0, L$) | ($q_3, 1, L$) | ($q_3, \#, L$) | - | ($q_0, 0, R$) | ($q_0, 1, R$) |
| Halting state $\rightarrow q_4$ | q_4 | q_4 | q_4 | q_4 | q_4 | q_4 |

Fig. Ex. 7.2.7(b) : Transition table

- The turing machine M is given by :

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

Where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$$\Gamma = \{0, 1, x, y, \#, B\}$$

q_0 = initial state

B = blank symbol

$$F = \{q_4\}$$

- Working of machine for an input 011 is shown in Fig. Ex. 7.2.7(c) :

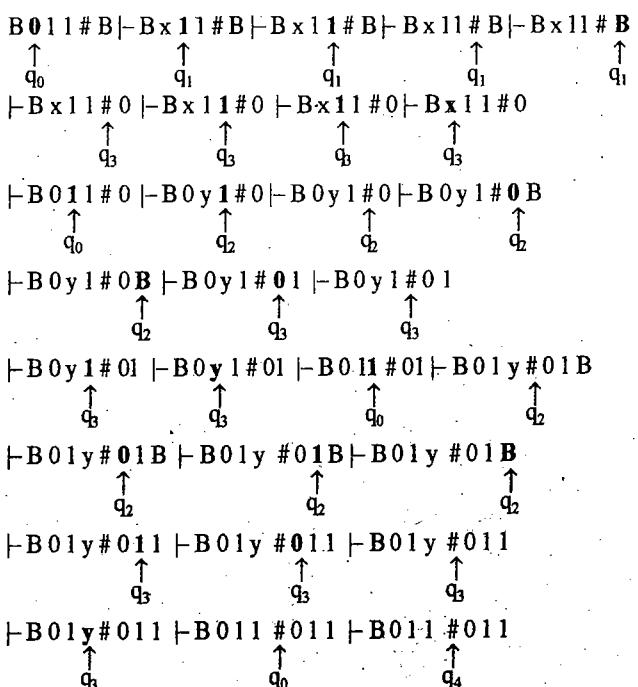


Fig. Ex. 7.2.7(c)

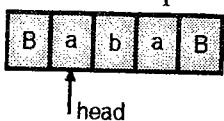
Example 7.2.8

Design TM to reverse a string $\Sigma = \{a, b\}$

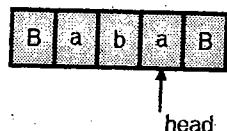


Solution : The working of the TM is being explained with the help of the string aba.

Initial configuration of the tape

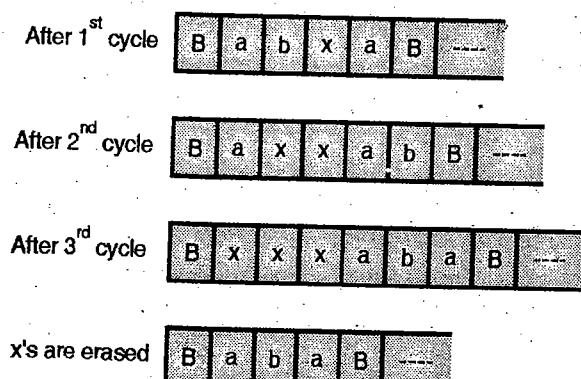


Step 1 : Head is moved to the rightmost symbol



Step 2 : We create another copy of aba in the reverse order. It will require 3 cycles as the length of aba is 3. In each cycle, the input symbol 'a' and 'b' are renamed as x.

Step 3 :



The transition diagram of the TM is given in Fig. Ex. 7.2.8.

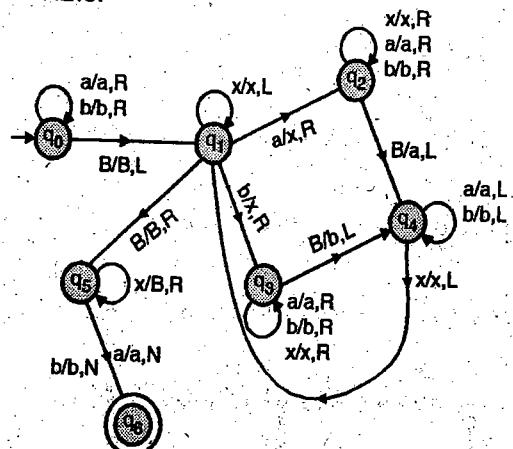


Fig. Ex. 7.2.8

Simulation of machine for input abb is shown in Fig. Ex. 7.2.8(a).

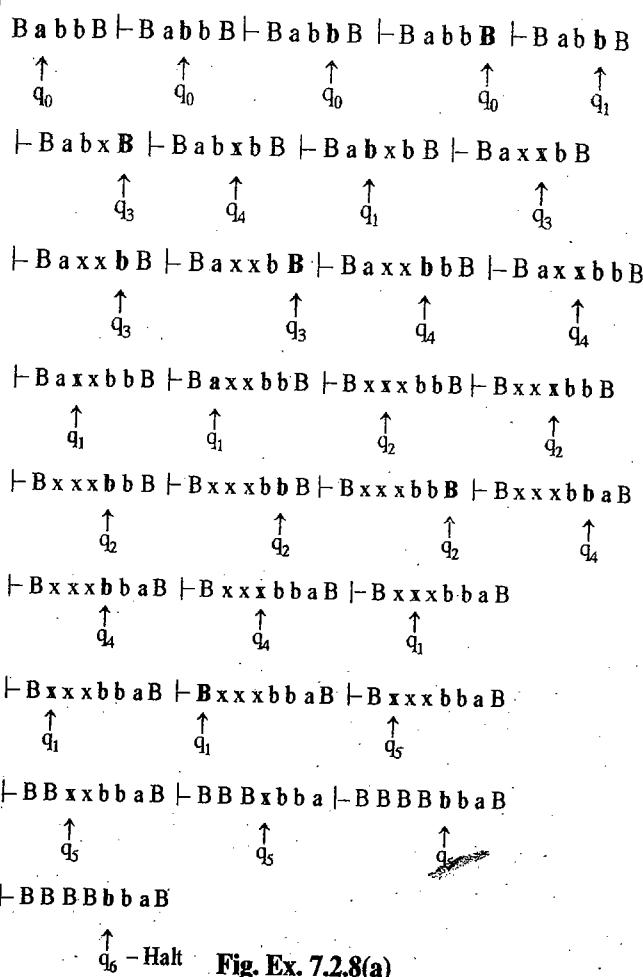


Fig. Ex. 7.2.8(a)

Example 7.2.9 SPPU - May 15, 8 Marks

Design a turing machine to check whether a string over {a,b} contains equal number of a's and b's.

Solution : Algorithm

1. Locate first a or first b.
2. If it is 'a' then locate 'b' rewrite them as x.
3. If it is 'b' then locate 'a' rewrite them as x.
4. Repeat steps from 1 to 3 till every a or b is re-written as x.

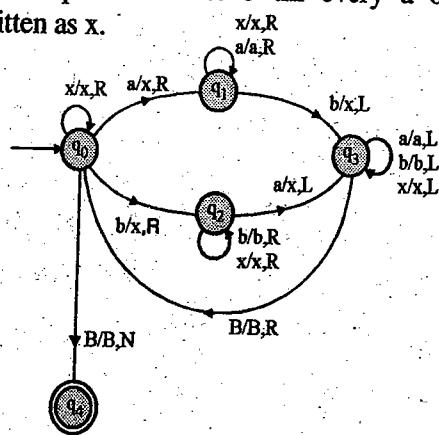


Fig. Ex. 7.2.9(a) : State transition diagram



| | a | b | X | B |
|-------------------|---------------|---------------|---------------|----------------------------|
| $\rightarrow q_0$ | (q_1, X, R) | (q_2, X, R) | (q_0, X, R) | (q_4, B, N) |
| q_1 | (q_1, a, R) | (q_3, X, L) | (q_1, X, R) | - |
| q_2 | (q_3, X, L) | (q_2, b, R) | (q_2, X, R) | - |
| q_3 | (q_3, a, L) | (q_3, b, L) | (q_3, X, L) | (q_0, B, R) |
| q_4^* | q_4 | q_4 | q_4 | \leftarrow Halting state |

Fig. Ex. 7.2.9(b) : Transition table

The turing machine M is given by :

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

Where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$$\Gamma = \{a, b, X, B\}$$

q_0 = initial state

B = blank symbol

$$F = \{q_4\}$$

Working of machine for an input abba is shown in Fig. Ex. 7.2.9(c) :

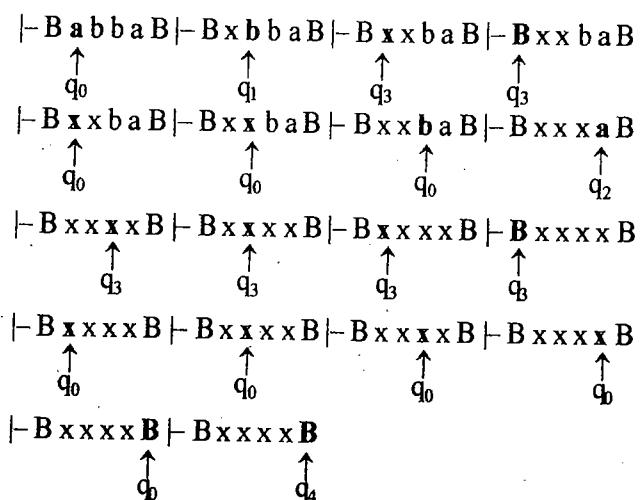


Fig. Ex. 7.2.9(c)

Example 7.2.10 :

Construct a TM for checking well formedness of parentheses.

Solution : In each cycle, the left-most ')' is written as X, then the head moves left to locate the nearer '(' and it is changed to X. The cycles of computation are shown in Fig. Ex. 7.2.10(a).

Input string is assumed to be ((0))0.

Cycle No.

Tape

| | |
|---------|--------------|
| Initial | B ((0))0 B |
| 1. | B (XX())0 B |
| 2. | B (XXXX)() B |
| 3. | B XXXXXX() B |
| 4. | B XXXXXXXX B |

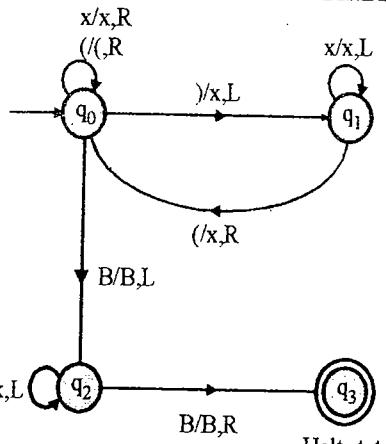


Fig. Ex. 7.2.10(a) : State transition diagram

| | (|) | X | B |
|---------|---------------|---------------|---------------|---------------|
| q_0 | $(q_0, (, R)$ | (q_1, X, L) | (q_0, X, R) | (q_2, B, L) |
| q_1 | (q_0, X, R) | - | (q_1, X, L) | - |
| q_2 | - | - | (q_2, X, L) | (q_3, B, R) |
| q_3^* | q_3 | q_3 | q_3 | q_3 |

Halting state

Fig. Ex. 7.2.10(b) : State transition table

The Turing machine M is given by :

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

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

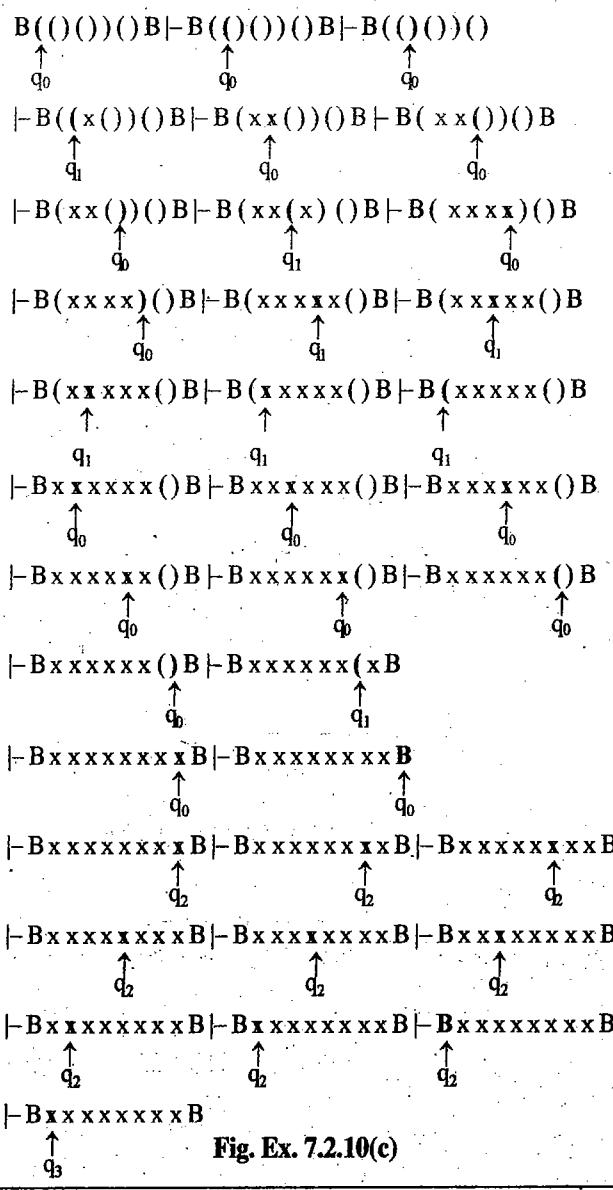
$$\Sigma = \{(),\}, \quad \Gamma = \{(), X, B\}$$

δ is given in Fig. Ex. 7.2.10(a) or Ex. 7.2.10(b)

q_0 = initial state, B = blank symbol

F = {q3}, halting state

Making of the machine for input (00)0 is given in Fig. Ex. 7.2.10(c) :

**Example 7.2.11** SPPU - Dec. 14, 6 Marks

Design a TM that recognizes a string containing aba as a substring.

Solution : This problem can be solved by a DFA. A problem that can be solved by a DFA can also be solved by a Turing machine.

- The DFA is shown in Fig. Ex. 7.2.11(a).
- The equivalent TM is shown in Fig. Ex. 7.2.11(b).
- The head scans the input from left to right without modifying it.

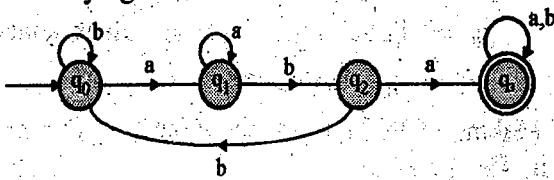


Fig. Ex. 7.2.11(a) : DFA for substring aba

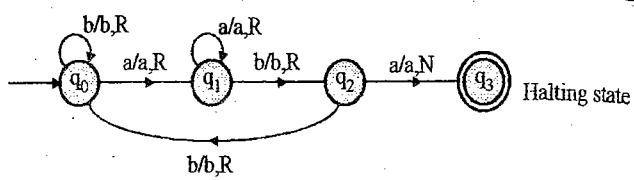


Fig. Ex. 7.2.11(b) : TM obtained from the DFA of Fig. Ex. 7.2.11(a)

The Turing machine M is given by :

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

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

$$\Sigma = \{a, b\}, \quad \Gamma = \{a, b, B\}$$

δ is given in Fig. Ex. 7.2.11(b)

q_0 = initial state, B = blank symbol,

$F = \{q_3\}$, Halting state

Example 7.2.12

Design a TM that replaces every occurrence of abb by baa.

Solution :

- The state q_2 implies that the preceding two characters are ab.
- In input b in state q_2 , completes the sequence abb.
- abb is changed to baa by a transition through $q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5$.

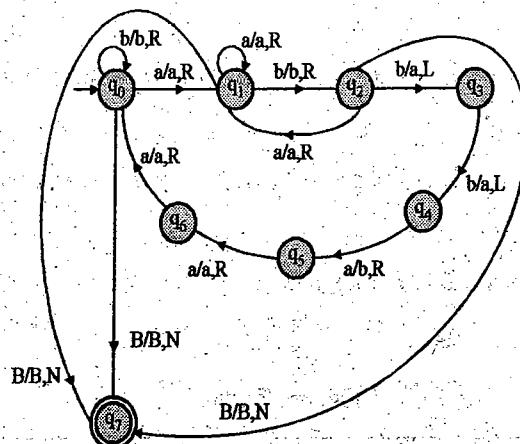


Fig. Ex. 7.2.12(a) : Transition diagram

- The head is positioned after the converted baa, by a transition through $q_5 \rightarrow q_6 \rightarrow q_0$.
- Above cycle is repeated to replace every occurrence of abb by baa.



| | a | b | B |
|-------------------|-----------------|-----------------|----------------------------|
| $\rightarrow q_0$ | (q_1, a, R) | (q_0, b, R) | (q_7, B, N) |
| q_1 | (q_1, a, R) | (q_2, b, R) | (q_7, B, N) |
| q_2 | (q_1, a, R) | (q_3, a, L) | (q_7, B, N) |
| q_3 | - | (q_4, a, L) | - |
| q_4 | (q_5, b, R) | - | - |
| q_5 | (q_6, a, R) | - | - |
| q_6 | (q_6, a, R) | - | - |
| q_7^* | q_7 | q_7 | \leftarrow Halting state |

Fig. Ex. 7.2.12(b) : State transition table

The Turing machine M is given by

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

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

$$\Sigma = \{a, b\}, \quad \Gamma = \{a, b, B\}$$

δ is given in Fig. Ex. 7.2.12(a) or (b)

q_0 = initial state, B = blank symbol

$F = \{q_7\}$, Halting state

Working of the machine for input bbabbaabbab is shown in Fig. Ex. 7.2.12(c) :

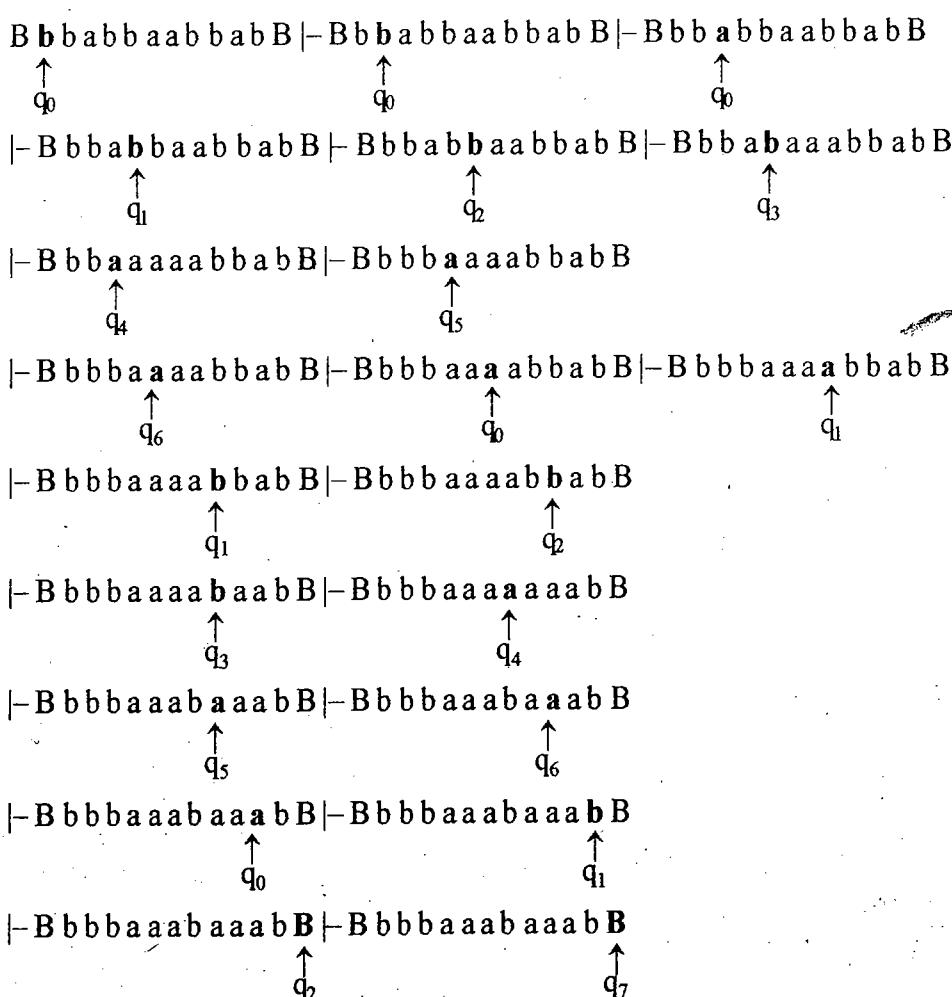


Fig. Ex. 7.2.12(c)

Example 7.2.13

Construct Turing Machine that recognizes the language :

$$L = \{x \in \{0,1\}^* \mid x \text{ ends in } 00\}$$

Solution : The given language L can have an equivalent DFA. Therefore, we will first design a DFA and then convert this DFA into a TM.

Step 1 : DFA for the given language L.

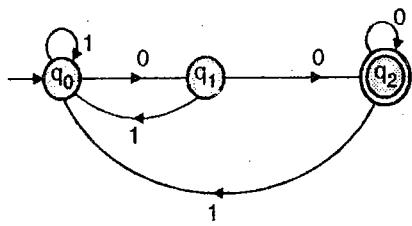


Fig. Ex. 7.2.13(a) : DFA for L

Step 2 : TM from the DFA.

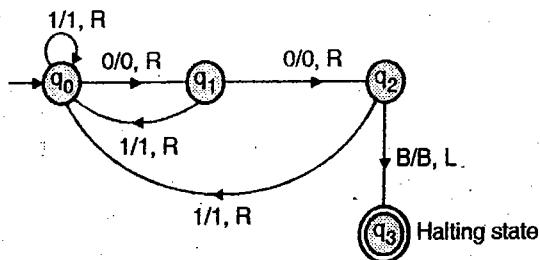


Fig. Ex. 7.2.13(b) : TM from DFA

- An input symbol B in either q_0 or q_1 will imply rejection.

The turing machine M is given by :

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

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

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

$$\Gamma = \{0, 1, B\}$$

δ = Transition function is given in step 2

q_0 = initial state

B = Blank symbol for the tape

$$F = \{q_3\}, \text{ halting state}$$

Example 7.2.14 [SPPU - Dec. 16, 8 Marks]

Design a turing machine to replace string 110 by 101 in binary input string.

Solution : The turing machine will look for every occurrence of the string 110.

- State q_2 is for previous two symbols as 11.
- Next symbol as 0 in state q_2 , will initiate the replacement process to replace 110 by 101.

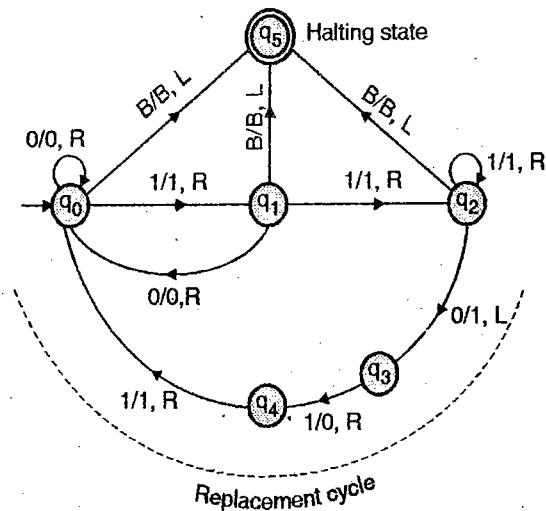


Fig. Ex. 7.2.14

The turing machine M is given by :

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

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

$$\Sigma = \{0, 1\}, \quad \Gamma = \{0, 1, B\}$$

δ = Transition function is shown using the transition diagram

B = Blank symbol for the tape

$$F = \{q_5\}, \text{ halting state}$$

Working of the machine for input 0101101 is shown in Fig. Ex. 7.2.14(a) :

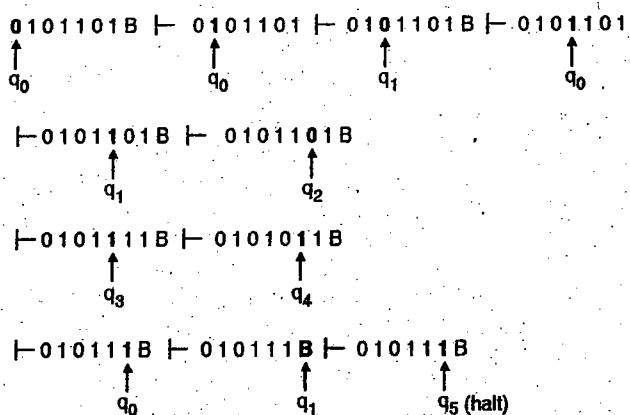


Fig. Ex. 7.2.14(a)

Example 7.2.15

Design and write out in full a turing machine that scans to the right until it finds two consecutive a's and then halts over the language of {a,b}.

Solution :

The solution can be given in two steps :

1. Design a DFA to recognize strings with substring aa.
2. Designing a TM from the DFA.

Step 1 : DFA for strings having aa in it.

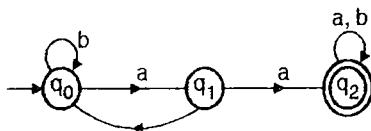


Fig. Ex. 7.2.15(a)

Step 2 : TM from the DFA

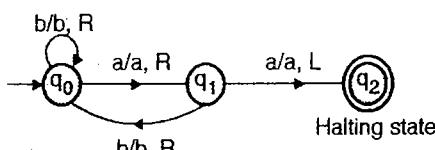


Fig. Ex. 7.2.15(b)

Example 7.2.16 SPPU - Dec. 15, 4 Marks

Design a turing machine that accepts the language of all strings which contain 'aba' as a substring.

Solution : This language can be accepted by a DFA. Therefore, we will first design a DFA and then convert the DFA into a TM.

Step 1 : DFA for the given language L

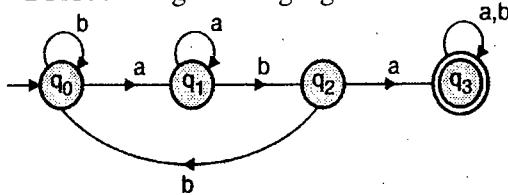


Fig. Ex. 7.2.16 (a)

Step 2 : TM from the DFA

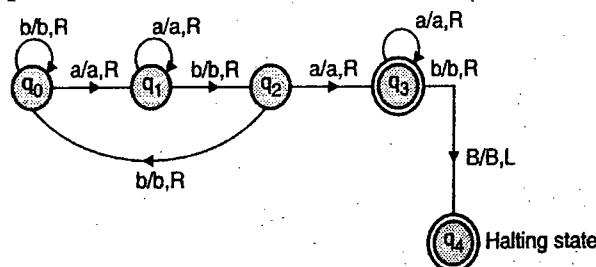


Fig. Ex. 7.2.16 (b)

The turing machine M is given by :

$$M = (Q, \Sigma, \Gamma, q_0, B, F)$$

Where, $Q = \{q_0, q_1, q_2, q_3, q_4\}$

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

$$\Gamma = \{a, b, B\}$$

δ = Transition function is given in the form of transition graph in step 2.

q_0 = initial state

B = Blank symbol for the tape

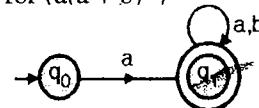
$$F = \{q_4\}, \text{ halting state.}$$

Example 7.2.17

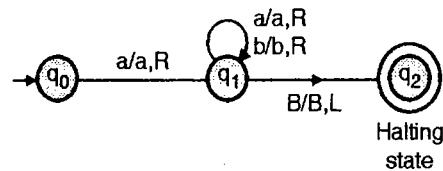
Design a Turing machine with no more than three states that accepts the language $(a(a+b)^*)$ assume that $\Sigma = \{a, b\}$.

Solution : This problem can be solved by a DFA. A problem that can be solved by a DFA can also be solved by a TM.

Step 1 : DFA for $(a(a+b)^*)$



Step 2 : Equivalent TM from the DFA.



The Turing machine

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_2\})$$

Example 7.2.18

Design a TM which recognizes words of the form $a^n b^n c^n$ | $n \geq 1$.

Solution :

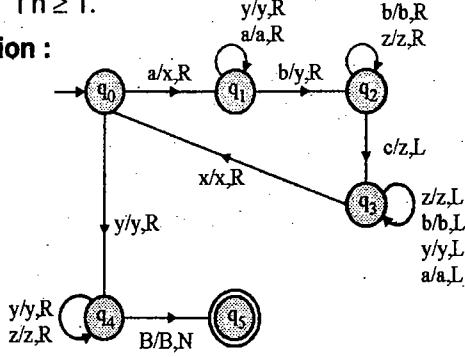


Fig. Ex. 7.2.18(a) : Transition diagram



| | a | b | c | x | y | z | B |
|-------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| $\rightarrow q_0$ | (q_1, x, R) | - | - | - | (q_4, y, R) | - | - |
| q_1 | (q_1, a, R) | (q_2, y, R) | - | - | (q_1, y, R) | - | - |
| q_2 | - | (q_2, b, R) | (q_3, z, R) | - | - | (q_2, z, R) | - |
| q_3 | (q_3, a, L) | (q_3, b, L) | - | (q_0, x, R) | (q_3, y, L) | (q_3, z, L) | - |
| q_4 | - | - | - | - | (q_4, y, R) | (q_4, z, R) | (q_5, B, N) |
| q_5^* | q_5 |

Hating
state

Fig. Ex. 7.2.18(b) : Transition table

The Turing machine M is given by :

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

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

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

$$\Gamma = \{a, b, c, x, y, z, B\}$$

δ = The transition is given

Fig. Ex. 7.2.18(a) or Ex. 7.2.18(b)

q_0 = Initial state

B = Blank symbol

F = $\{q_5\}$, Halting state

Algorithm

For a string $a^n b^n c^n$, the TM will need n cycles. In each cycle :

1. Leftmost a is written as x
2. Leftmost b is written as y
3. Leftmost c is written as z

At the end of n cycles, the tape should contain only x's, y's and z's.

Working of the TM for input $a^3 b^3 c^3$ is shown in Fig. Ex. 7.2.18(c).

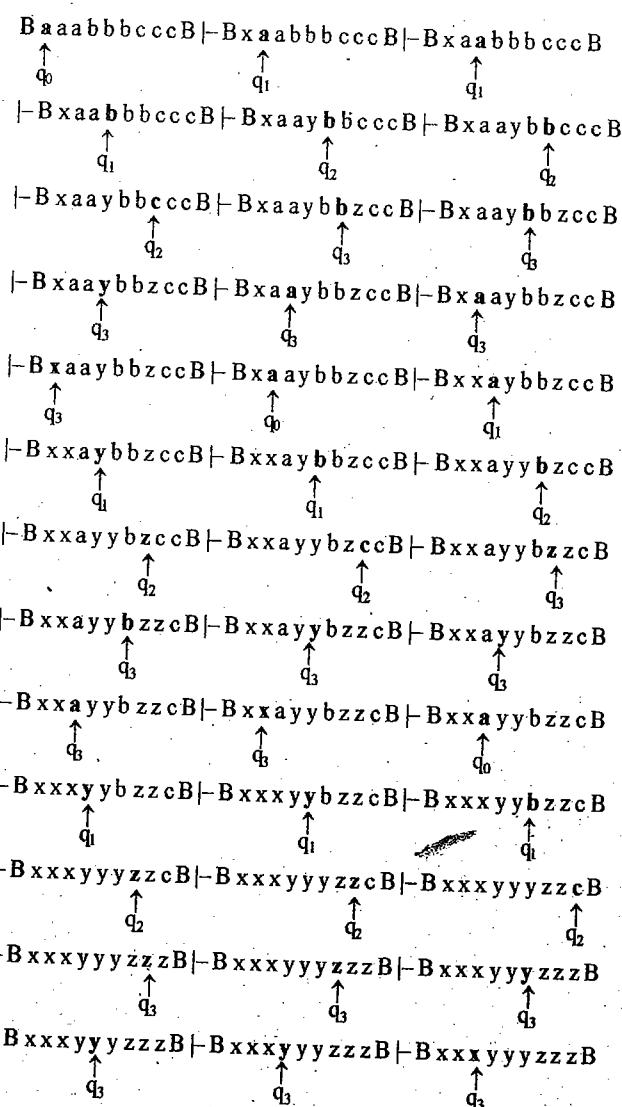


Fig. Ex. 7.2.18(c)

Example 7.2.19 SPPU - Dec. 13, 8 Marks

Design a TM which recognizes palindromes over alphabet {a,b}

Solution : A palindrome can have one of the following forms :

1. $\omega\omega^R$
2. $\omega a \omega^R$
3. $\omega b \omega^R$

Where ω is a string over {a,b} with $|\omega| \geq 0$

Algorithm

1. Algorithm requires n cycles, where $|\omega| = n$.
2. In each cycle, first character is matched with the last character and both are erased.

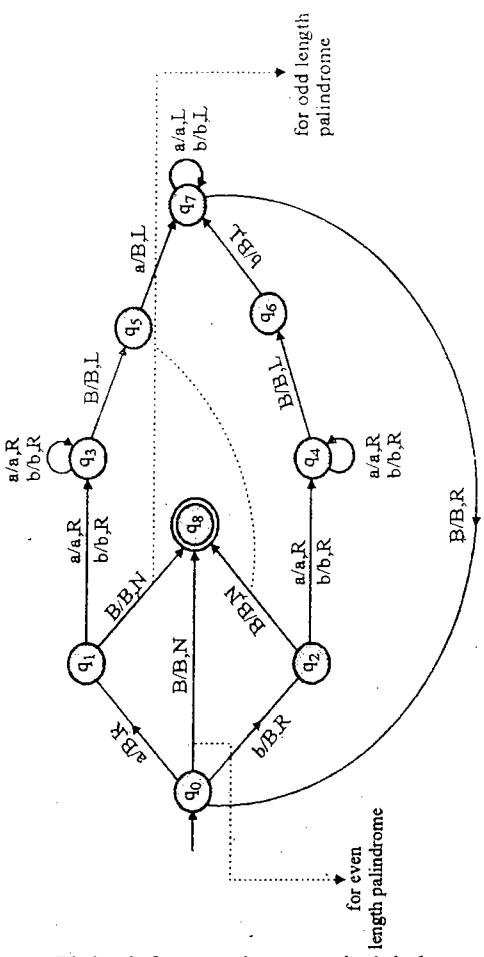


Fig. Ex. 7.2.19(a) : Transition diagram

- If the leftmost character is 'a' the machine takes a path through $q_0 \rightarrow q_1 \rightarrow q_3 \rightarrow q_5 \rightarrow q_7$, looking for last character as 'a'.
- If the leftmost character is 'b', the machine takes a path through $q_0 \rightarrow q_2 \rightarrow q_4 \rightarrow q_6 \rightarrow q_7$, looking for last character as 'b'.

The Turing machine M is given by :

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

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

$$\Sigma = \{a, b\}, \quad \Gamma = \{a, b, B\}$$

The transition function δ is given in Fig. Ex. 7.2.19(a)

q_0 = initial state

B = blank symbol

F = $\{q_8\}$, halting state

Working of TM for input abbabba is shown in Fig. Ex. 7.2.19(b).

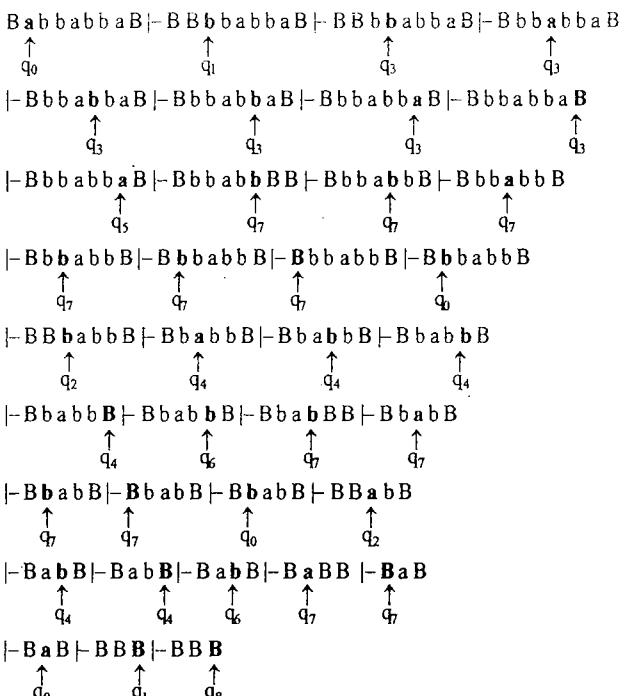


Fig. Ex. 7.2.19(b)

Example 7.2.20

Draw a transition diagram for a Turning machine accepting the following language.

L = The language of all non-palindromes over {a,b}

Solution : The solution follows from example 7.2.19.

In Fig. Ex. 7.2.19(a), if the tape does not contain 'a' in the state q_5 or the tape does not contain 'b' in state q_6 then it is a non-palindrome.

The required TM is given below :

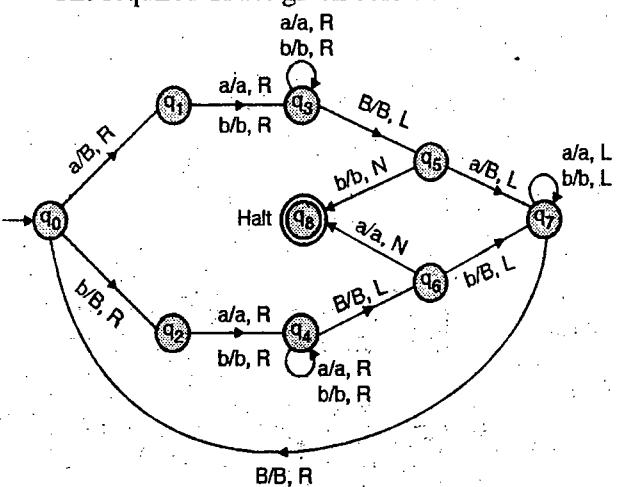


Fig. Ex. 7.2.20 : TM

**Example 7.2.21**

Design a TM which recognizes words of the form $\omega\omega\omega$ over alphabet {a,b}.

Solution :

Step 1: Three ω 's of $\omega\omega\omega$ can be separated as explained below :

- 1.1 Advance two places from the left. While advancing, each 'a' is written as x and 'b' is written as y.**

Initial

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | a | b | b | a | b | b | a | b | b | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 1st cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | x | y | b | a | b | b | a | b | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 2nd cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | x | y | y | x | b | b | a | 2 | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 3rd cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | x | y | y | x | y | y | 1 | 2 | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

- 1.2 Locate the last alphabet of $\omega\omega\omega$ and change it to 1 if it is a, otherwise change it to 2.**

- 1.3 Step number 1.1 over 1.2 is performed several times.**

Situation of tape for $\omega = abb$ is shown below.

- Step 2 :** First ω of $\omega\omega\omega$ is matched with last ω . While matching, the first ω is erased and the last ω is converted to a string of {a,b}.

Situation of the tape is shown below :

Initial

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | x | y | y | x | y | y | 1 | 2 | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 1st cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | B | y | y | x | y | y | a | 2 | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 2nd cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | B | B | y | x | y | y | a | b | 2 | B |
|---|---|---|---|---|---|---|---|---|---|---|

After 3rd cycle

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| B | B | B | B | x | y | y | a | b | b | B |
|---|---|---|---|---|---|---|---|---|---|---|

- Step 3 :** After step 2, the tape contains $\omega\omega$, where the first ω is a string over {x,y} . In this step first ω is matched with 2nd ω .

Situation of the tape is shown below :

Initial

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | x | y | y | a | b | b | B |
|---|---|---|---|---|---|---|---|

After 1st cycle

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | B | y | y | 1 | b | b | B |
|---|---|---|---|---|---|---|---|

After 2nd cycle

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | B | B | y | 1 | 2 | b | B |
|---|---|---|---|---|---|---|---|

After 3rd cycle

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | B | B | B | 1 | 2 | 2 | B |
|---|---|---|---|---|---|---|---|

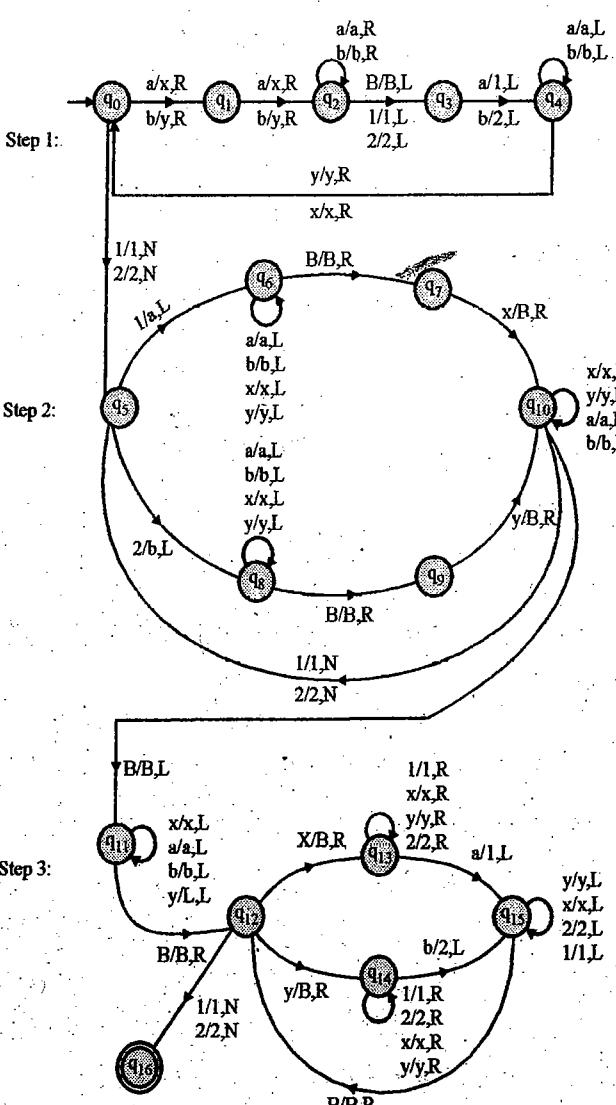


Fig. Ex. 7.2.21 : Transition diagram



7.3 Turing Machines as Computer of Functions

A turing machine can be used for computation. It can perform several operations including :

1. Addition 2. Subtraction
3. Multiplication 4. Division

Computation of a function f can be represented as

$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

A function f , $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is said to be turing computable if there is a turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ such that :

$$(q_0, B\omega B) \xrightarrow{M} (q_f, B\mu B)$$

Where $\omega \in \Sigma_1^*$ and $\mu \in \Sigma_2^*$ satisfying

$$f(\omega) = \mu \text{ and } q_f \in F$$

Representation of a number

The unary number system is often used while computing a function using a Turing machine. The unary system uses only one symbol. This symbol is represented as 0. Representation of some decimal numbers in unary system is given below :

Decimal Number Unary Number

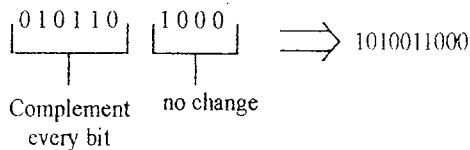
| | |
|---|----------------------------------|
| 0 | B [Represented by blank string] |
| 1 | 0 |
| 2 | 00 |
| 5 | 00000 |
| n | 0 n times 0, or 0^n |

Example 7.3.1 [SPPU - Dec. 15, May 16, 8 Marks]

Design a TM to find 2's complement of a binary machine.

Solution :

2's complement of a binary number can be found by not changing bits from right end till the first '1' and then complementing remaining bits. For example, the 2's complement of a binary number 0101101000 is calculated as given below :



Algorithm

1. Locate the last bit (right most).
2. Move towards left till the first 1.
3. Complement remaining bits, while moving towards left.

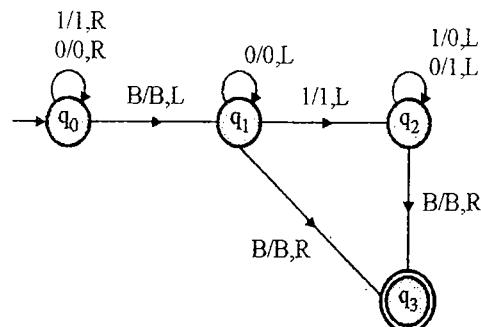


Fig. Ex. 7.3.1(a) : Transition diagram

| | 0 | 1 | B |
|-------------------|---------------|---------------|---------------------------------------|
| $\rightarrow q_0$ | $(q_0, 0, R)$ | $(q_0, 1, R)$ | (q_1, B, L) |
| q_1 | $(q_1, 0, L)$ | $(q_2, 1, L)$ | (q_3, B, R) |
| q_2 | $(q_2, 1, L)$ | $(q_2, 0, L)$ | (q_3, B, R) |
| q_3^* | q_3 | q_3 | $q_3 \leftarrow \text{Halting state}$ |

Fig. Ex. 7.3.1(b) : Transition table

The Turing machine M is given by :

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

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

$$\Sigma = \{0, 1\}, \quad \Gamma = \{0, 1, B\}$$

The transition function δ is given in

Fig. Ex. 7.3.1(a) or (b),

q_0 = Initial state, B = Blank symbol

$F = \{q_3\}$, Halting state

Working of TM for input 0101101000 is shown in Fig. Ex. 7.3.1(c) :

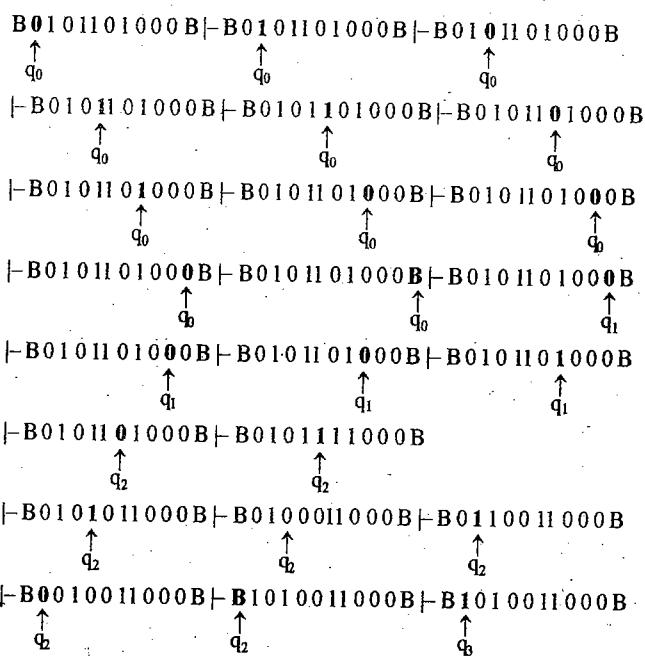
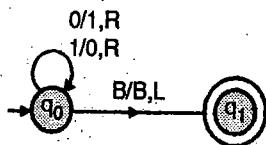


Fig. Ex. 7.3.1(c)

Example 7.3.2

Design a TM to find the 1's complement of a given binary input.

Solution : 1's complement of a binary number can be computed by complementing every bit of the given binary number.



(a) Transition diagram

| | 0 | 1 | B |
|---------|---------------|---------------|---------------------------------------|
| q_0 | $(q_0, 1, R)$ | $(q_0, 0, R)$ | (q_1, B, L) |
| q_1^* | q_1 | q_1 | $q_1 \leftarrow \text{Halting state}$ |

(b) Transition table

Fig. Ex. 7.3.2

The Turing machine $M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_1\})$

The transition function δ is given through transition diagram/table.

Example 7.3.3 SPPU - Dec. 13, 8 Marks

Design a TM to compute proper subtraction of two unary numbers. The proper subtraction function f is defined as follows : $f(m, n) = \begin{cases} m - n & \text{if } m > n \\ 0 & \text{otherwise} \end{cases}$

Solution : The working of the TM is being explained with subtraction of 3 from 5.

In unary system, 5 is represented as 00000.

In unary system, 3 is represented as 000.

In unary system, 0 is represented by a blank tape.

Subtraction will require several cycle. In each cycle :

1. Leftmost 0 is erased.
2. Rightmost 0 is erased.

Situation of tape after each cycle is shown below :

Initial

After 1st cycle

After 2nd cycle

After 3rd cycle

Transition diagram and transition table are given in Fig. Ex. 7.3.3(a) and (b).

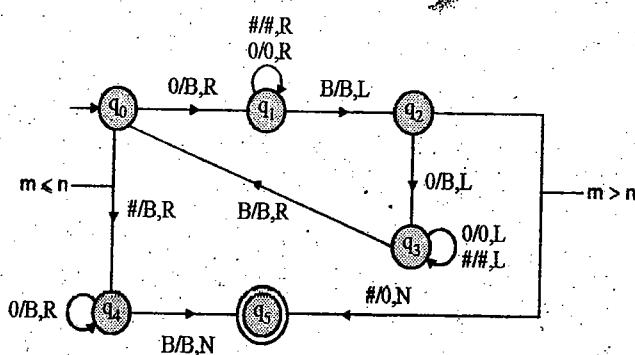


Fig. Ex. 7.3.3(a) : Transition diagram

| | 0 | # | B |
|-------------------|---------------|----------------|---------------------------------------|
| $\rightarrow q_0$ | (q_1, B, R) | (q_4, B, R) | - |
| q_1 | $(q_1, 0, R)$ | $(q_1, \#, R)$ | (q_2, B, L) |
| q_2 | (q_3, B, L) | $(q_5, 0, N)$ | - |
| q_3 | $(q_3, 0, L)$ | $(q_3, \#, L)$ | (q_0, B, R) |
| q_4 | (q_4, B, R) | - | (q_5, B, N) |
| q_5^* | q_5 | q_5 | $q_5 \leftarrow \text{Halting state}$ |

Fig. Ex. 7.3.3(b) : Transition table

The Turing machine M is given by :

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



where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$$\Sigma = \{0, 1, \#\}, \Gamma = (0, 1, \#, B)$$

The transition function δ is given in Fig. Ex. 7.3.3(a) and (b)

q_0 = initial state, B = blank symbol

$F = \{q_5\}$, Halting state

The working of TM is being simulated for 5-3 is shown in Fig. Ex. 7.3.3(c) :

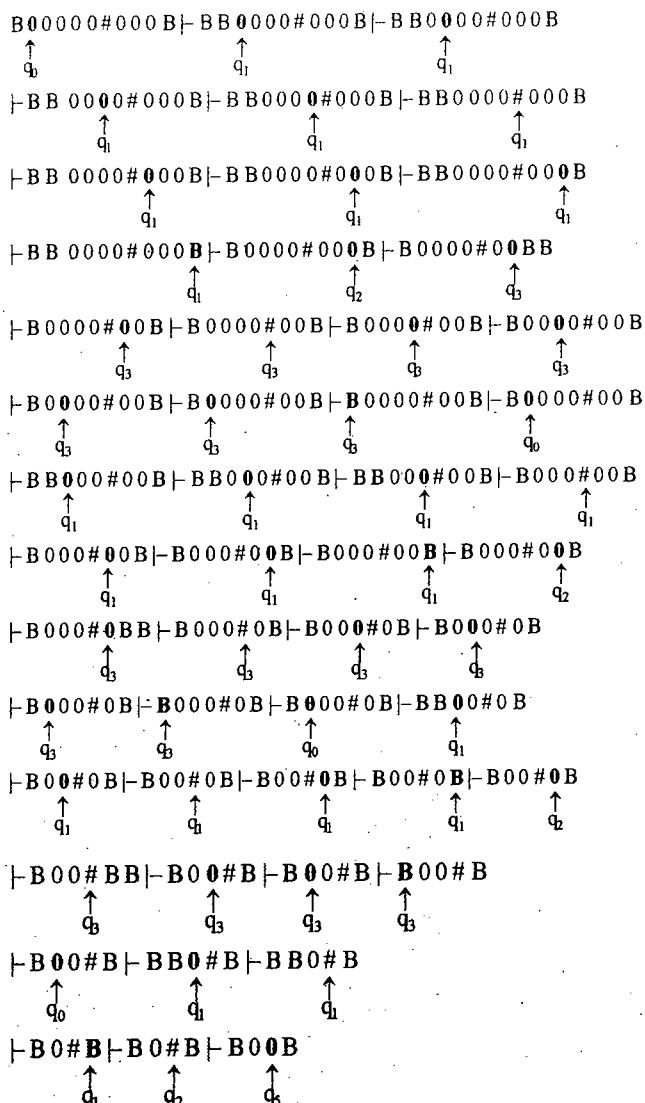


Fig. Ex. 7.3.3(c)

Example 7.3.4

Design TM to compare two numbers, which will produce the output L if first number is less than the second number, output G if first is greater than the second number and E otherwise.

Solution :

This machine is an extension of machine for proper subtraction given in Example 7.3.3.

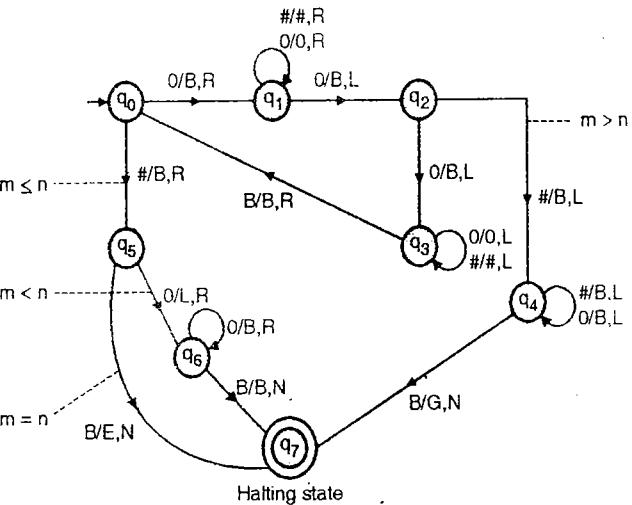


Fig. Ex. 7.3.4

- An input # in state q_2 means $m > n$. Everything is erased from the tape and an alphabet G is written on the tape.
- A # followed by 0 in state q_0 means $m < n$. L is written on the tape.
- A # followed by B (blank) in state q_0 means $m = n$. E is written on the tape.

The Turing machine

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{0, \#\}, \{0, \#, B, G, L, E\}, \delta, q_0, B, \{q_7\})$$

The transition function δ is specified using the transition diagram.

Example 7.3.5 SPPU - May 15, 12 Marks

Design a TM that computes the function

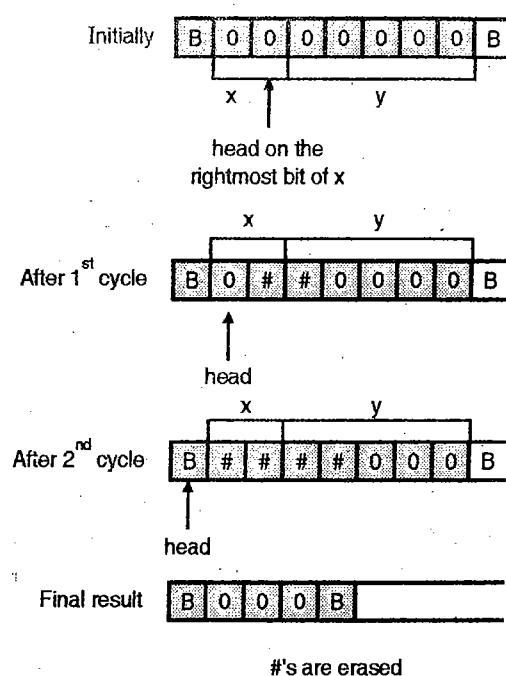
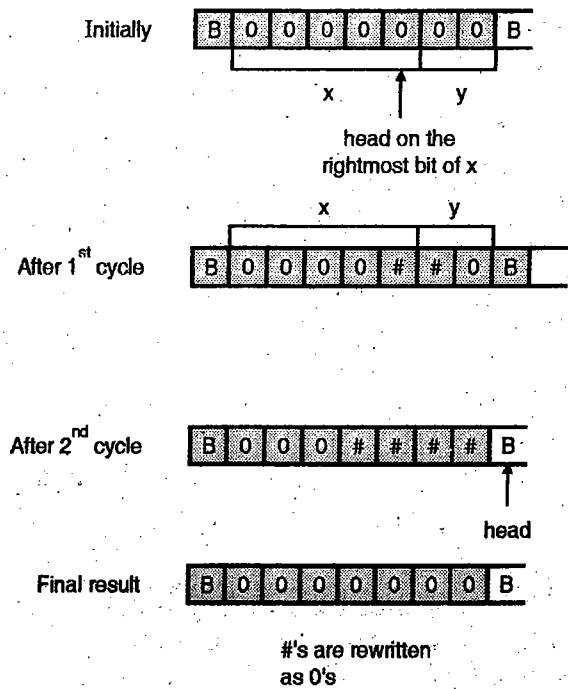
$$g(x, y) = x + y, \text{ if } x \geq y = y - x, \text{ if } y > x$$

Simulate working for (i) $x = 2, y = 2$ (ii) $x = 0, y = 2$.

Solution :

Working of the machine is being explained with the help of two examples :

1. $y > x$ [$y = 5, x = 2$]
2. $y \leq x$ [$y = 2, x = 5$]

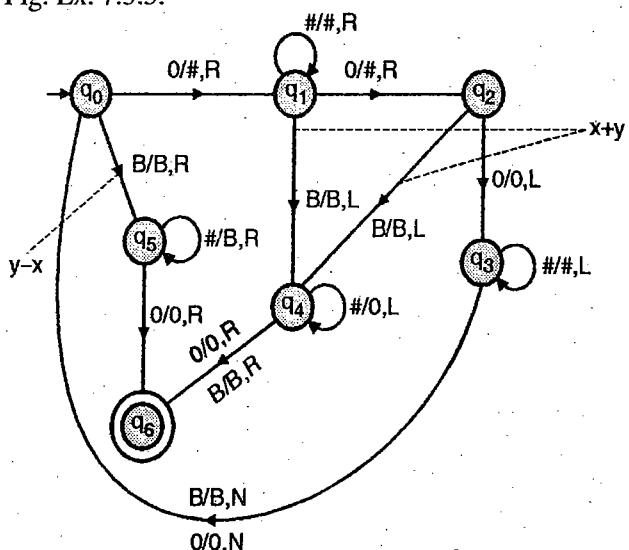
Example 1 ($y=5, x=2$)**Example 2 ($y=2, x=5$)**

- Initially, head is positioned on the rightmost bit of x.
- In each cycle :
 1. Nearest (from head) zero of x is written as #.
 2. Nearest zero of y is written as #.

- If the head hits the B (blank) on the left side before B (blank) on the right side then #'s are erased to perform $y - x$.

- If the head hits the B(blank) on the right side before B (blank) on the left side then #'s are re-written as 0's to perform $x + y$.

The transition diagram of the TM is shown in Fig. Ex. 7.3.5.

**Fig. Ex. 7.3.5**

The Turing machine $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, \#, B\}, \delta, q_0, B, \{q_6\})$

Where δ is specified through the transition diagram.

Simulation of machine

- (i) $x = 2, y = 2$

|— B 0 0 0 0 B |— B 0 # 0 0 B |— B 0 # # 0 B |— B 0 # # 0 B
 ↑ ↑ ↑ ↑
 q₀ q₁ q₂ q₃
 |— B 0 # # 0 B |— B # # 0 B |— B # # 0 B
 ↑ ↑ ↑
 q₃ q₀ q₀
 |— B # # # 0 B |— B # # # 0 B |— B # # # 0 B |— B # # # # B
 ↑ ↑ ↑ ↑
 q₁ q₁ q₁ q₂
 |— B # # # # B |— B # # # 0 B |— B # # 0 0 B |— B # 0 0 0 B
 ↑ ↑ ↑ ↑
 q₄ q₄ q₄ q₄
 |— B 0 0 0 0 B |— B 0 0 0 0 B
 ↑ ↑
 q₄ q₆ —halt



(ii) $x = 0, y = 2$

$x = 0$, will be represented by blanks and $y = 2$ will be represented by two 0's.

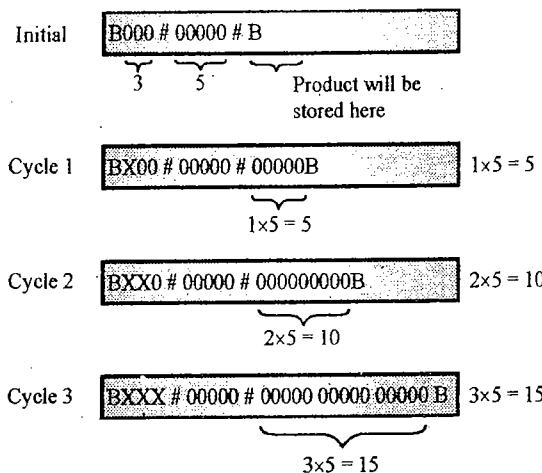
B 0 0 B \vdash B 0 0 B \vdash B 0 0 B
 \uparrow \uparrow \uparrow
 q_0 q_5 q_6 - halt

Example 7.3.6 SPPU - May 12, 10 Marks

Design a TM to compute multiplication of two unary numbers.

Solution : Multiplication algorithm is being explained with the help of an example.

3×5 will require three cycles.



- To calculate 3×5 , three times, 5 zero's are appended.
- Unary representation of 3 is 000.
- Unary representation of 5 is 00000.
- 3, 5 and the result, are separated by #.
- Inside each major cycles (three cycles for 3), there will be a number of minor cycles (5 minor cycles for 5) to append 0's one at a time.

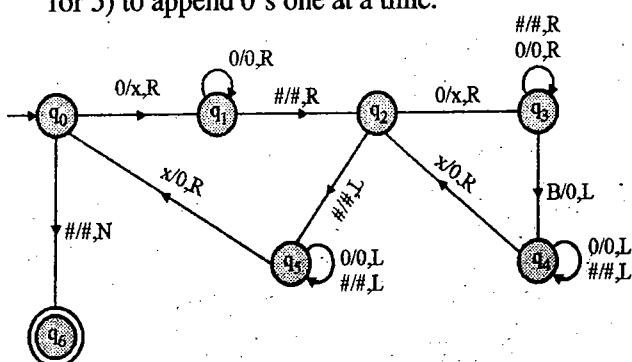


Fig. Ex. 7.3.6 : Transition diagram for TM of example 7.3.6

- Let us assume that the two numbers to be multiplied are x_1 and x_2 .

x_1 is represented by ω_1 , where ω_1 is a string of 0's.

x_2 is represented by ω_2 , where ω_2 is a string of 0's.

$x_1 * x_2$ is represented by ω_3 , where ω_3 is a string of 0's.

separates ω_1 and ω_2 , ω_2 and ω_3 .

- In the TM shown in Fig. Ex. 7.3.6, there are two cycles.

- The cycle $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_5 \rightarrow q_0$ appends ω_2 to ω_3 for every zero in ω_1 , with the help of cycle $q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_2$

Working of TM for 2×2 is shown in Fig. Ex. 7.3.6(a) :

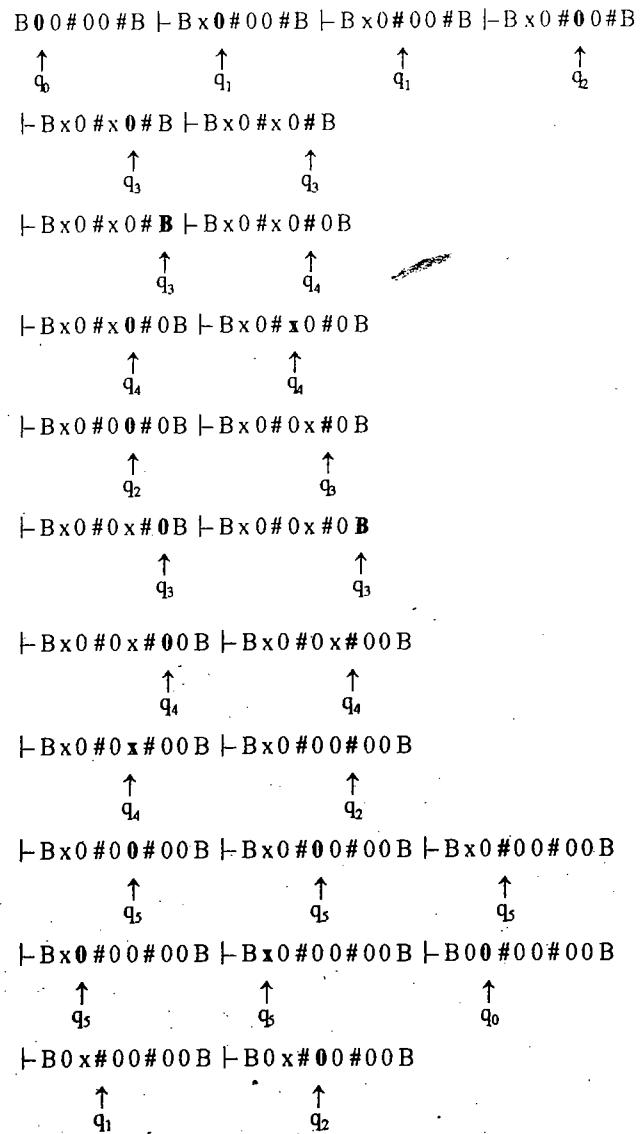


Fig. Ex. 7.3.6(a) Contd...

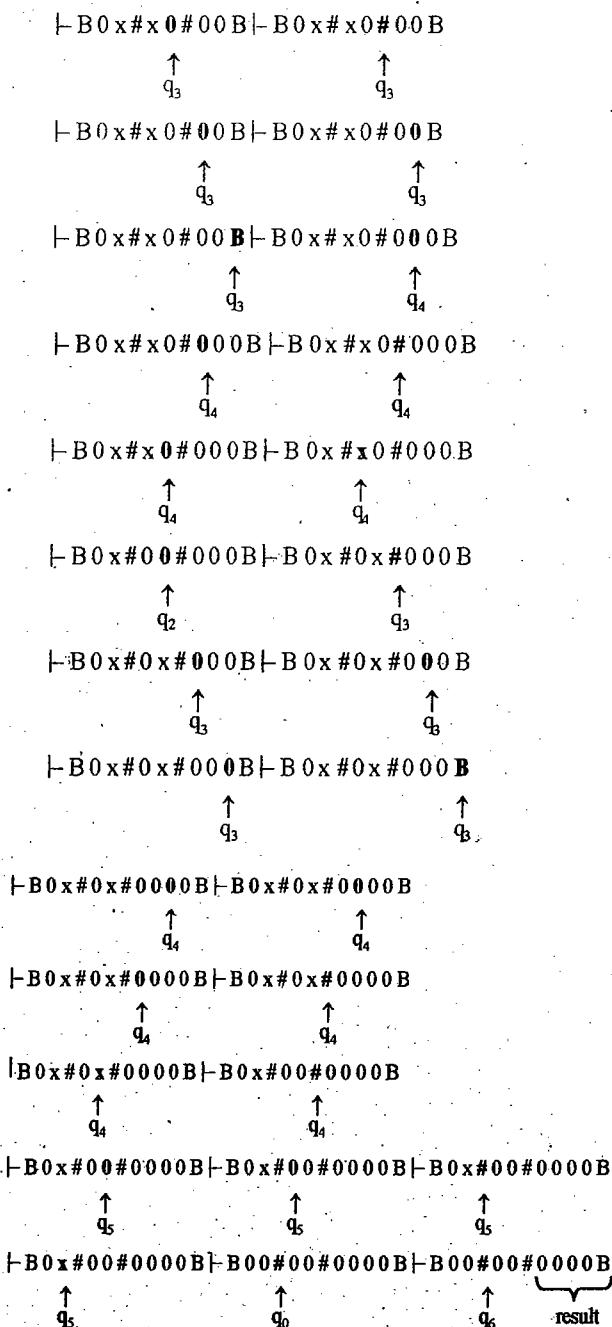


Fig. Ex. 7.3.6(a)

Example 7.3.7Design TM for n^2 where ' n ' is integer and $n \geq 0$.Solution : Computation of n^2 is same as multiplication of n with n . Computation of n^2 is being explained with the help of an example.**Computation of 3^2 :**

3 is represented as 000.

Initially the tape will contain $|B 0 0 0 \# B|$, we make a copy of the string $|B 0 0 0 \# 0 0 0 B|$.

Now, the multiplication of 3 with 3 will require three cycles as shown in Fig. Ex. 7.3.7(a).

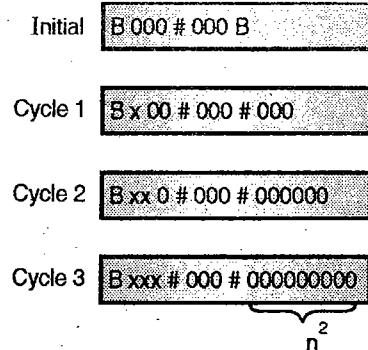


Fig. Ex. 7.3.7(a)

The transition diagram of the required TM is shown in Fig. Ex. 7.3.7(b).

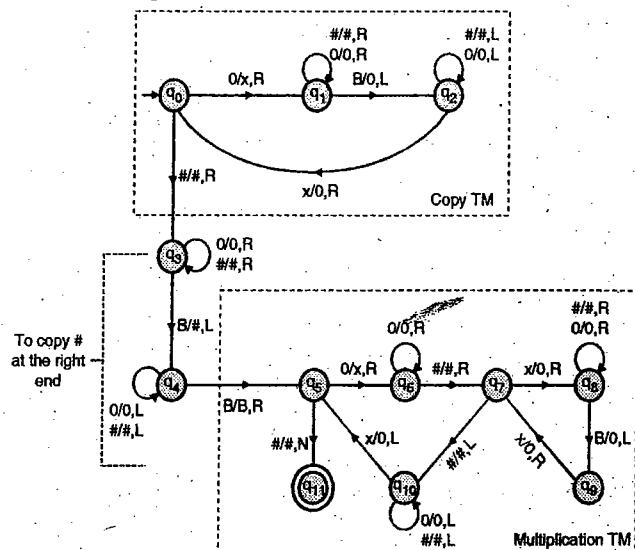


Fig. Ex. 7.3.7(b)

- TM for multiplication is given in Example 7.3.6.
- TM for copy operation is given in Example 7.2.7.

Example 7.3.8

Design a TM to compute remainder and quotient when a unary number is divided by another unary number.

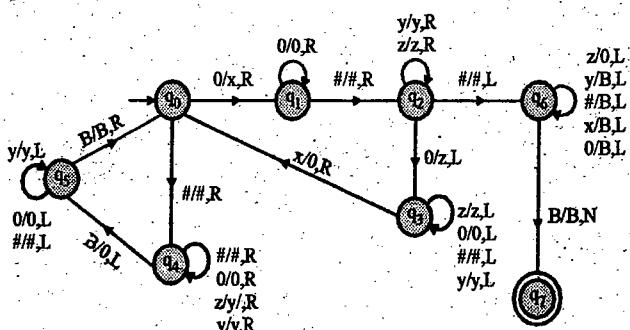
Solution :

Fig. Ex. 7.3.8 : Transition diagrams



- Division can be performed through repeated subtractions.
- To perform $x \div y$, both x and y are represented in unary system. y is divisor and x is dividend.
- y is repeatedly subtracted from x , as long as $x \geq y$ each time, y is subtracted from x , quotient is incremented by 1.
- Initial situation of tape for $5 \div 2$ is shown in Fig. Ex. 7.3.8(a).

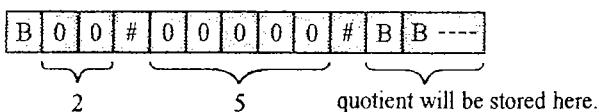


Fig. Ex. 7.3.8(a)

Working of machine for $5 \div 2$ is shown in Fig. Ex. 7.3.8(b).

- If y cannot be subtracted from x then z 's become remainder.

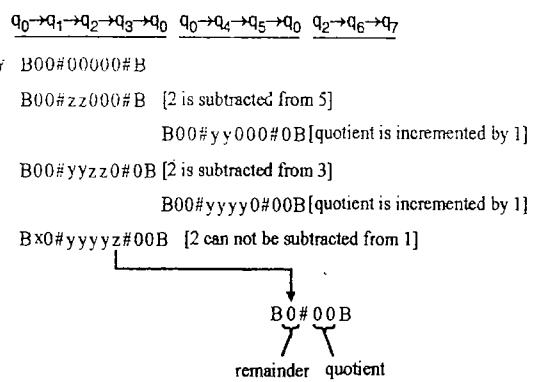
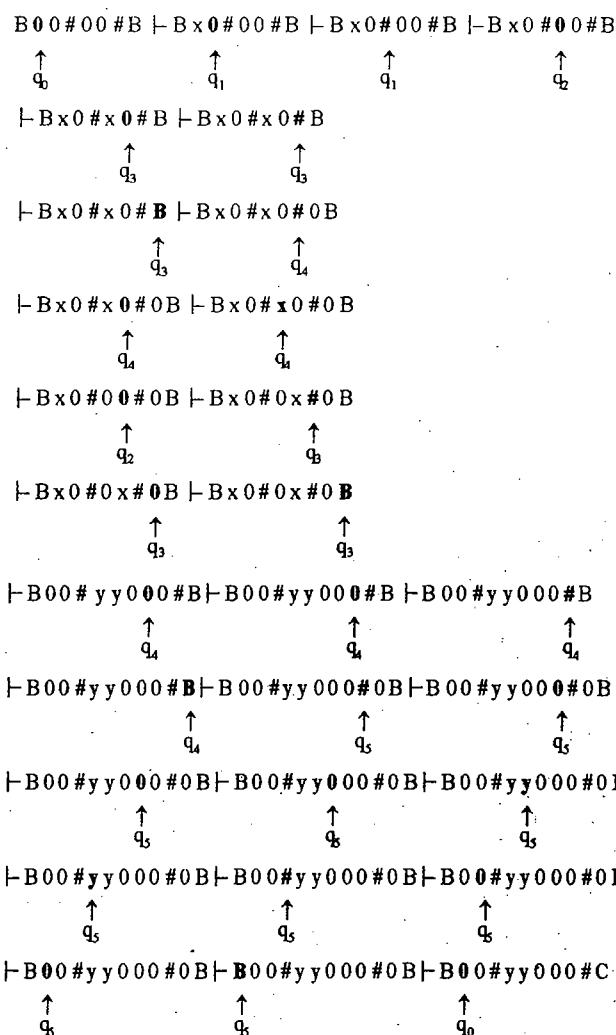
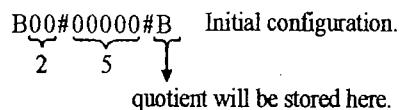


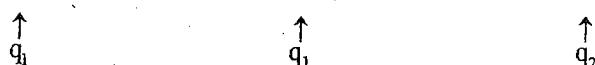
Fig. Ex. 7.3.8(b)

Working of TM for $5 \div 2$ is shown in Fig. Ex. 7.3.8(c)

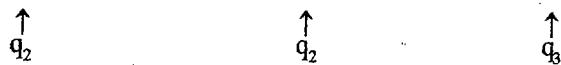




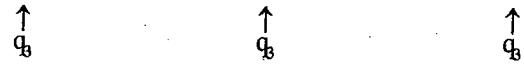
$| - B x 0 \# y y 0 0 0 \# 0 B | - B x 0 \# y y 0 0 0 \# 0 B | - B x 0 \# y y 0 0 0 \# 0 B$



$| - B x 0 \# y y 0 0 0 \# 0 B | - B x 0 \# y y 0 0 0 \# 0 B | - B x 0 \# y y z 0 0 \# 0 B$



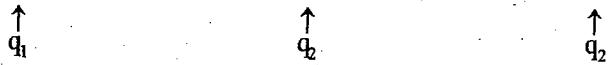
$| - B x 0 \# y y z 0 0 \# 0 B | - B x 0 \# y y z 0 0 \# 0 B | - B x 0 \# y y z 0 0 \# 0 B$



$| - B x 0 \# y y z 0 0 \# 0 B | - B 0 0 \# y y z 0 0 \# 0 B$



$| - B 0 x \# y y z 0 0 \# 0 B | - B 0 x \# y y z 0 0 \# 0 B | - B 0 x \# y y z 0 0 \# 0 B$



$| - B 0 x \# y y z 0 0 \# 0 B | - B 0 x \# y y z 0 0 \# 0 B$



$| - B 0 x \# y y z z 0 \# 0 B | - B 0 x \# y y z z 0 \# 0 B$



$| - B 0 x \# y y z z 0 \# 0 B | - B 0 x \# y y z z 0 \# 0 B$



$| - B 0 x \# y y z z 0 \# 0 B | - B 0 0 \# y y z z 0 \# 0 B$



$| - B 0 0 \# y y z z 0 \# 0 B | - B 0 0 \# y y z z 0 \# 0 B$



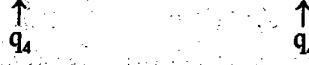
$| - B 0 0 \# y y z z 0 \# 0 B | - B 0 0 \# y y y z 0 \# 0 B$



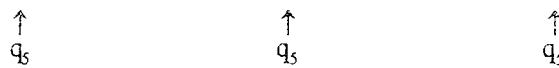
$| - B 0 0 \# y y y z 0 \# 0 B | - B 0 0 \# y y y 0 \# 0 B$



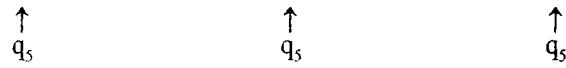
$| - B 0 0 \# y y y 0 \# 0 B | - B 0 0 \# y y y 0 \# 0 B$



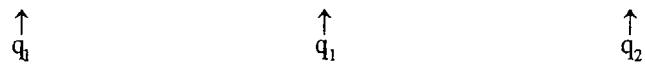


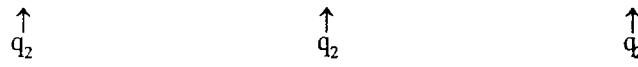
$$\vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B$$


$$\vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B$$

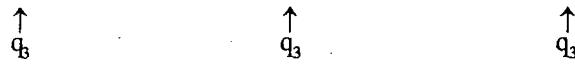

$$\vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B$$


$$\vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B \vdash B 00 \# y y y 0 \# 00B$$


$$\vdash B x 0 \# y y y 0 \# 00B \vdash B x 0 \# y y y 0 \# 00B \vdash B x 0 \# y y y 0 \# 00B$$


$$\vdash B x 0 \# y y y 0 \# 00B \vdash B x 0 \# y y y 0 \# 00B \vdash B x 0 \# y y y 0 \# 00B$$


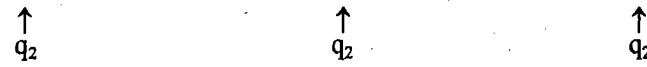
$$\vdash B x 0 \# y y y 0 \# 00B \vdash B x 0 \# y y y 0 \# 00B$$


$$\vdash B x 0 \# y y y z \# 00B \vdash B x 0 \# y y y z \# 00B \vdash B x 0 \# y y y z \# 00B$$


$$\vdash B x 0 \# y y y z \# 00B \vdash B x 0 \# y y y z \# 00B$$


$$\vdash B x 0 \# y y y z \# 00B \vdash B x 0 \# y y y z \# 00B$$


$$\vdash B 0 x \# y y y z \# 00B \vdash B 0 x \# y y y z \# 00B$$


$$\vdash B 0 x \# y y y z \# 00B \vdash B 0 x \# y y y z \# 00B \vdash B 0 x \# y y y z \# 00B$$


$$\vdash B 0 x \# y y y z \# 00B \vdash B 0 x \# y y y z \# 00B$$

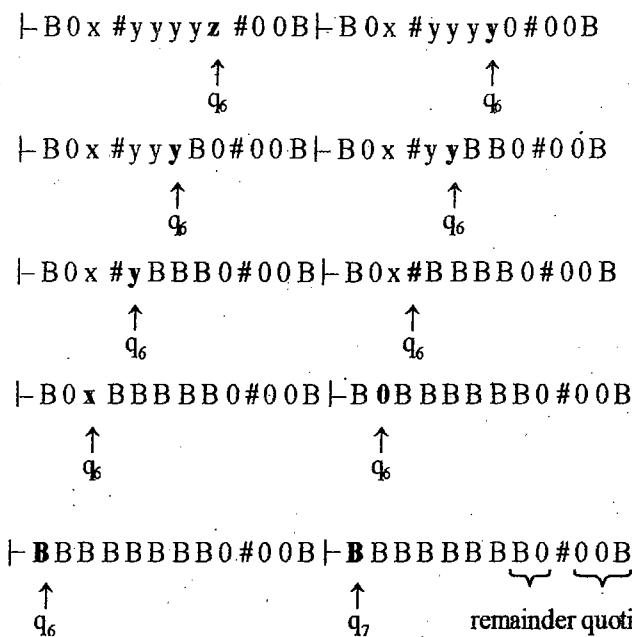



Fig. Ex. 7.3.8(c)

Example 7.3.9

Design a TM to increment a binary number by 1.

Solution : To increment a binary number by 1, following steps are required :

1. Trailing 1's should become 0.
2. First '0' from right end should become 1.
3. Remaining bits will remain as it is.

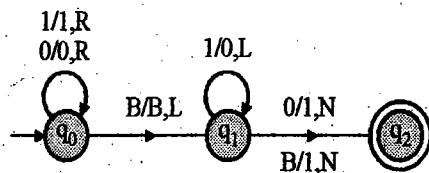


Fig. Ex. 7.3.9(a) : Transition diagram

| | 0 | 1 | B |
|-------------------|---------------|---------------|-----------------------|
| $\rightarrow q_0$ | $(q_0, 0, R)$ | $(q_0, 1, R)$ | (q_1, B, L) |
| q_1 | $(q_2, 1, N)$ | $(q_1, 0, L)$ | $(q_2, 1, N)$ |
| q_2^* | q_2 | q_2 | q_2 ← Halting state |

Fig. Ex. 7.3.9(b) : Transition table

- State q_0 moves the head to the first blank on the right side.
- A transition from q_0 to q_1 , positions the head on

the rightmost 0 or 1.

- Trailing 1's are written as 0's in state q_1 .
- A transition from q_1 to q_2 changes the first '0' or a blank from the right end towards left to 1.
- q_2 is halting state.

The working of TM being simulated for 01010111, is shown in Fig. Ex. 7.3.9(c)

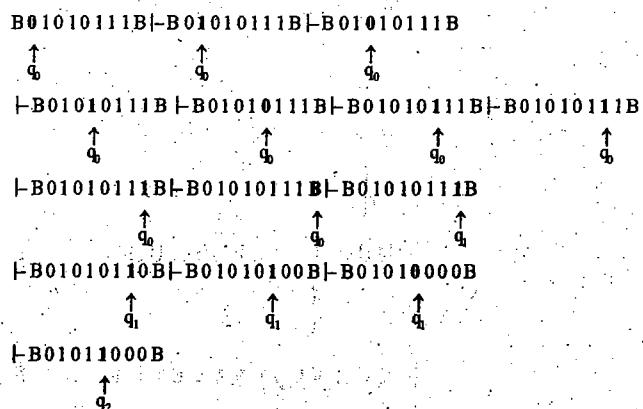


Fig. Ex. 7.3.9(c)

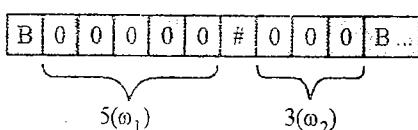
Example 7.3.10 SPPU - Dec. 14, Dec. 16, 8 Marks

Design a TM to add two unary numbers.

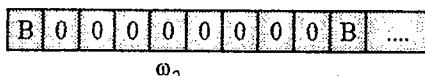
Solution : Addition of two unary numbers can be performed through append operation. To add two numbers 5 (say ω_1) and 3 (say ω_2) will require following steps :



1. Initial configuration of tape :



2. ω_1 is appended to ω_2 .



While every '0' from ω_1 is getting appended to ω_2 , '0' from ω_1 is erased. ω_2 contains 8 0's, which is sum of 5 and 3.

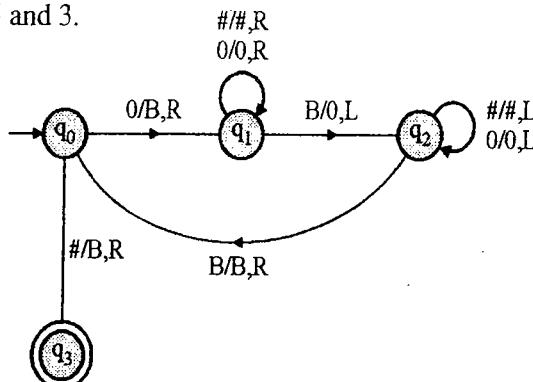


Fig. Ex. 7.3.10(a) : Transition diagram

| | 0 | # | B |
|-------------------|---------------|----------------|-----------------------|
| $\rightarrow q_0$ | (q_1, B, R) | (q_3, B, R) | - |
| q_1 | $(q_1, 0, R)$ | $(q_1, \#, R)$ | $(q_2, 0, L)$ |
| q_2 | $(q_2, 0, L)$ | $(q_2, \#, L)$ | (q_0, B, R) |
| q_3^* | q_3 | q_3 | q_3 ← Halting state |

Fig. Ex. 7.3.10(b) : Transition table

The turing machine M is given by :

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

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

$$\Sigma = \{0, \#\}, \quad \Gamma = \{0, \#, B\}$$

δ = Transition function is given in
Fig. Ex. 7.3.10 (a), (b)

q_0 = initial state, B = blank symbol

$F = \{q_3\}$, halting state.

Example 7.3.11

Design a TM to find the GCD (greatest common divisor) of two unary numbers.

Solution : Two numbers x and y are stored on the tape without any separator. Head is positioned on the last digit of x.

- Algorithm for finding GCD is based on repeated subtraction of x from y if $y \geq x$ or y from x if $y < x$. ultimately either x or y will become 0.

- o If x becomes 0 then y is GCD
- o If y becomes 0 then x is GCD

- Algorithm for finding GCD is given below in C-language :

```
while(x != 0 && y != 0)
```

```
{ if(x >= y)
```

```
    x = x - y;
```

```
else
```

```
    y = y - x;
```

```
}
```

```
if(x > 0)
```

```
    printf ("% d", x);
```

```
else
```

```
    printf ("% d", y);
```

Situation of tape for $x = 6$ and $y = 4$ is shown in Fig. Ex. 7.3.11 :

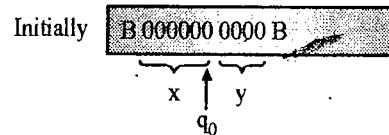


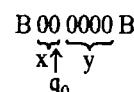
Fig. Ex. 7.3.11

1st cycle : Subtraction is carried out through several cycles. In each cycle, the last '0' of 'x' is written as x and the first '0' of y is written as Y.

1. B00000 X Y 000 B
2. B0000 XX YY 00B
3. B000 XXX YYY 0B
4. B00 XXXXX YYYYY B

Since, x is greater than y , y will end first.

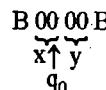
At this point Y's are erased and



X's are written as 0's

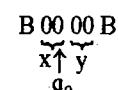
2nd cycle :

1. BOXY000B
2. BXYY00B



Since, y is greater than x, x will end first.

At this point, X's are erased and Y's are written as 0's.



3rd cycle :

1. BOXY0B 2. BXXYYB

Since, y is equal to x, both x and y have ended at the same time. At this point, x's are erased and Y's are written as 0's. The tape contains the final result.

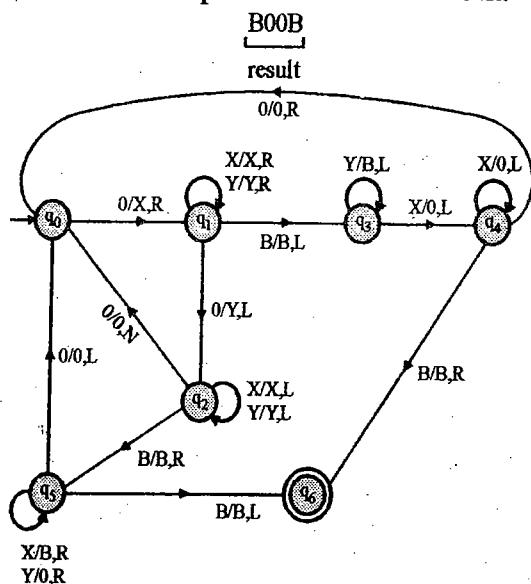


Fig. Ex. 7.3.11(a) : Transition diagram

Initial configuration $\overbrace{B000000}^6 \overbrace{0000B}^4$ head is positioned on the last digit of 6.

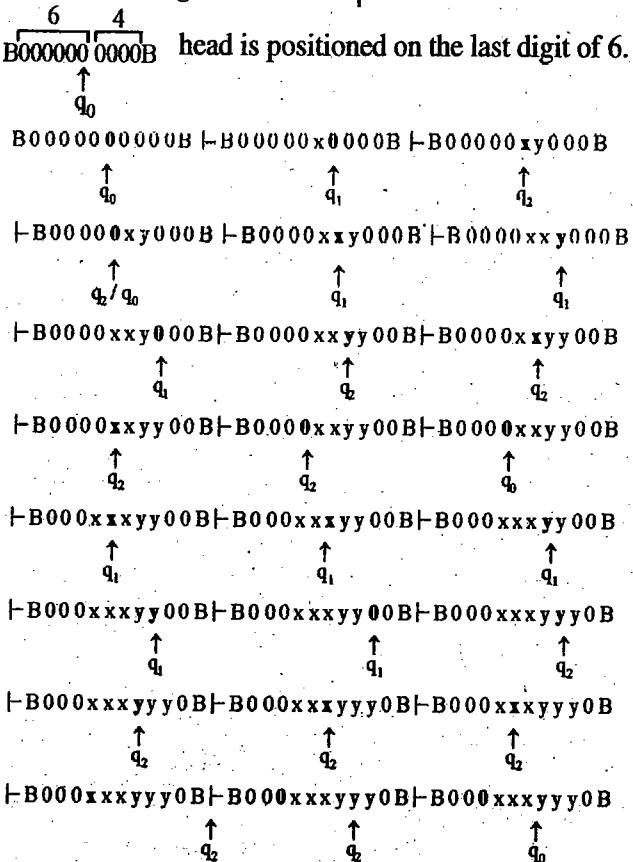


Fig. Ex. 7.3.11(b) contd....

The transition diagram of the TM is given in Fig. Ex. 7.3.11(a).

The TM can be defined as :

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

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

$$\Sigma = \{0\}$$

$$\Gamma = \{0, X, Y, B\}$$

δ = Transition function is given in

Fig. Ex. 7.3.11(a)

q_0 = initial state

B = blank symbol

F = $\{q_6\}$, halting state

Working of TM for GCD (6, 4) is shown in Fig. Ex. 7.3.11(b) :

6 is represented as 000000

4 is represented as 0000

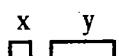
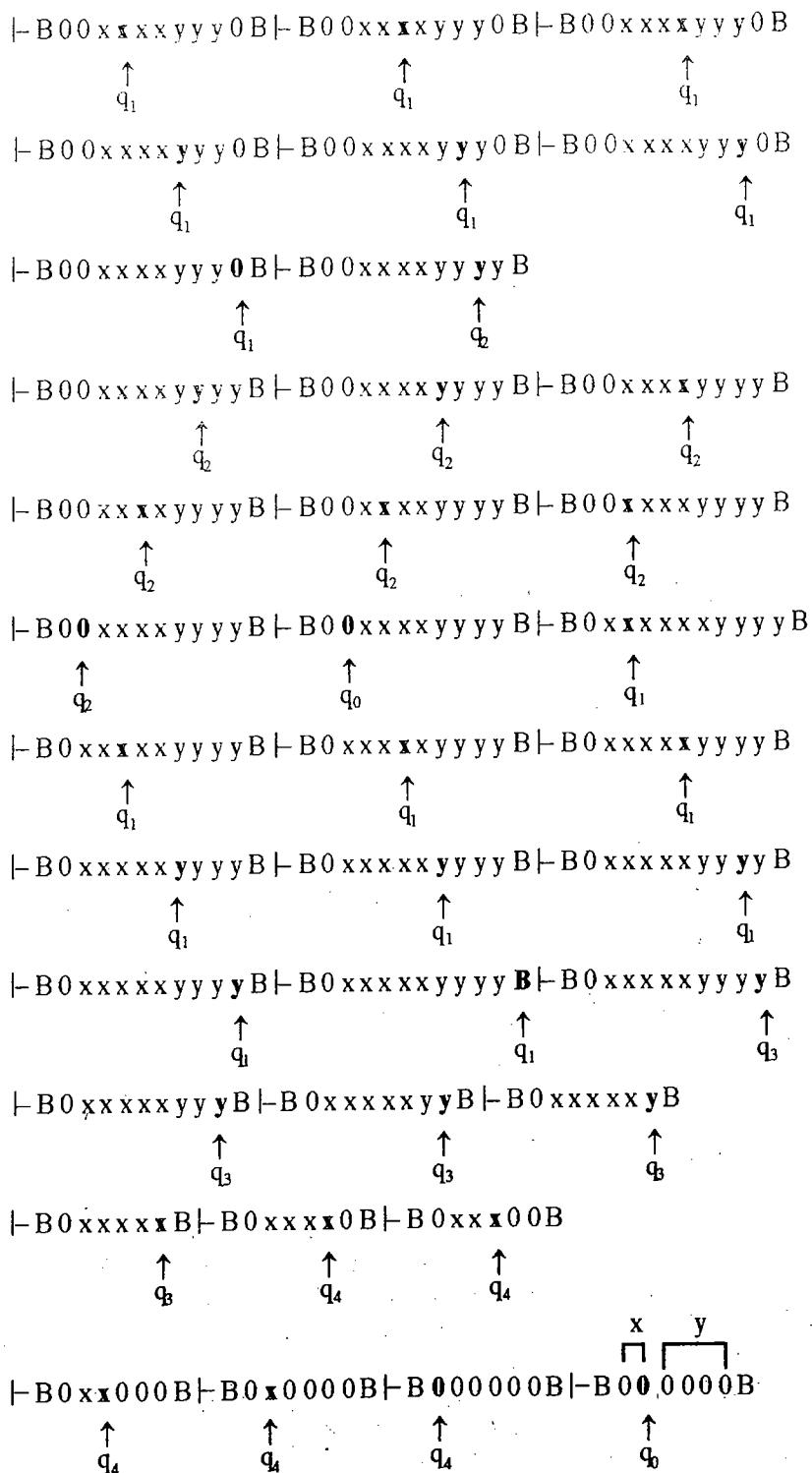
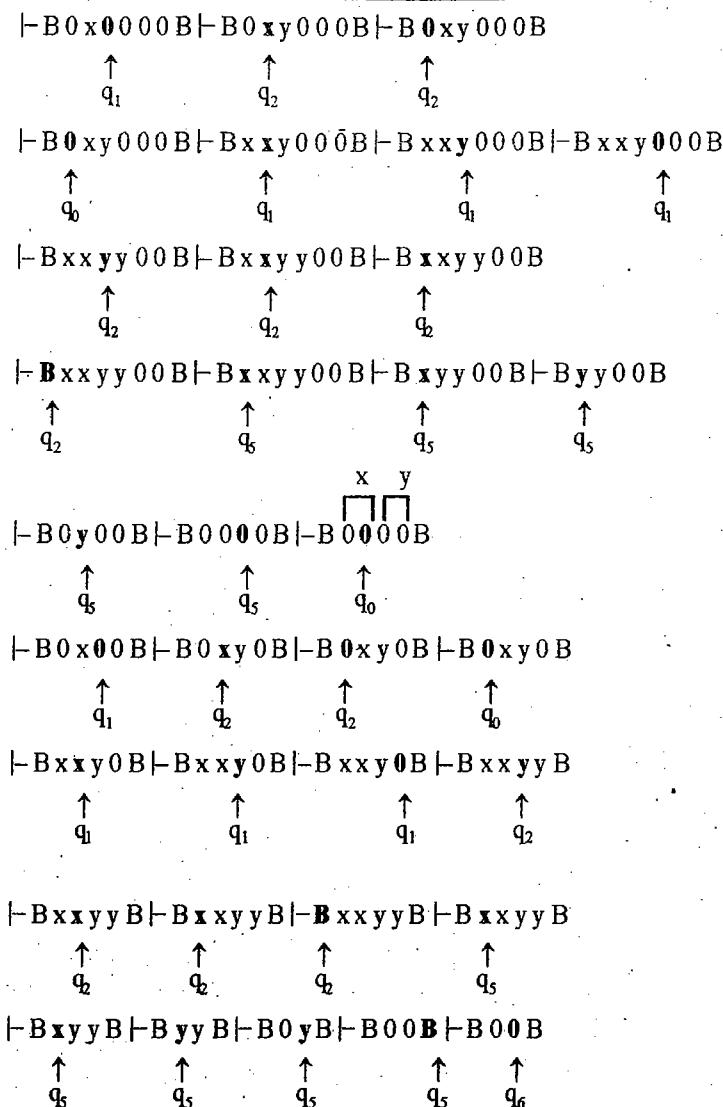


Fig. Ex. 7.3.11(b) contd....



Result = 00, which is 2.

Fig. Ex. 7.3.11(b)

Example 7.3.12

Design a TM to find the value of $\log_2(n)$, where n is any binary number.

Solution : $\log_2(n)$ of any number n lying between 2^n and 2^{n+1} is given by n .

i.e. if $2^n \leq n < 2^{n+1}$, then $\log_2(n) = n$

Let us consider the case of a number

$$n = 36 \quad 2^5 \leq 36 < 2^6$$

Therefore, $\log_2(36) = 5$

36 can be written as 100100.

- Any number n satisfying the condition $2^5 \leq n < 2^6$ can be written as 1XXXXX (where X stands for either 1 or 0).
- $\log_2(1XXXXX)$ can be calculated by erasing the most significant bit 1 and renaming other bits as '0'. Unary representation of 5 is 00000.

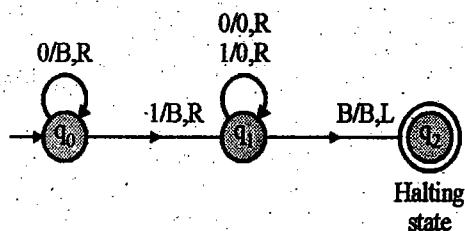


Fig. Ex. 7.3.12(a) : Transition diagram



| | 0 | 1 | B | |
|-------------------|---------------|---------------|---------------|----------------------------|
| $\rightarrow q_0$ | (q_0, B, R) | (q_1, B, R) | - | |
| q_1 | $(q_1, 0, R)$ | $(q_1, 0, R)$ | (q_2, B, L) | |
| q_2^* | q_2 | q_2 | q_2 | \leftarrow Halting state |

Fig. Ex. 7.3.12(b) : Transition table

Working of TM for $(36)_{10}$ is shown in Fig. Ex. 7.3.12(c) :

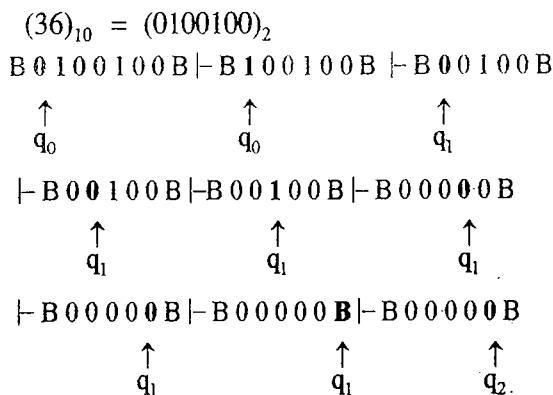


Fig. Ex. 7.3.12(c)

7.4 Languages of TM

A language $L \subseteq \Sigma^*$ is accepted by a Turing machine M which halts in the accepting state on every $\omega \in L$. The set of languages accepted by a turing machine is often called the recursively enumerable languages or RE languages.

Syllabus Topic : Variants of Turing Machine

7.5 Extension of Turing Machine

SPPU - Dec. 13. May 15

University Questions

- Q. Write a short note on : Encoding of TM
(Dec. 2013, 3 Marks)
- Q. Explain the different types of tuning machines.
(May 2015, 4 Marks)

In a standard Turing machine, the tape is semi-infinite. It is bounded on the left and unbounded on the right side. Some of the extensions of turing machine are given below :

1. Tape is of infinite length in both the directions.
2. Multiple heads and a single tape.

3. Multiple tape with each tape having its own independent head.
4. K-dimension tape.
5. Non-deterministic turing machine.

7.5.1 Two-way Infinite Turing Machine

In a standard turing machine number of positions for leftmost blanks is fixed and they are included in instantaneous description, where the right-hand blanks are not included.

In the two way infinite Turing machine, there is an infinite sequence of blanks on each side of the input string. In an instantaneous description, these blanks are never shown.

7.5.2 A Turing Machine with Multiple Heads

A turing machine with single tape can have multiple heads. Let us consider a turing machine with two heads H_1 and H_2 . Each head is capable of performing read/write /move operation independently.

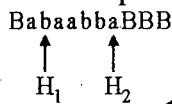


Fig. 7.5.1 : A Turing machine with two heads

The transition behavior of 2-head one tape Turing machine can be defined as given below :

$$\delta (\text{State}, \text{Symbol under } H_1, \text{Symbol under } H_2) = (\text{New state}, (S_1, M_1), (S_2, M_2))$$

Where,

- S_1 is the symbol to be written in the cell under H_1 .
- M_1 is the movement (L,R,N) of H_1 .
- S_2 is the symbol to be written in the cell under H_2 .
- M_2 is the movement (L,R,N) of H_2 .

Syllabus Topic : Multi-Tape Turing Machine

7.5.3 Multi-Tape Turing Machine

SPPU - Dec. 14. Dec. 15. May 16. Dec. 16

University Question

- Q. Write short notes on Multi-tape Turing machine.
(Dec. 2014, Dec. 2015, May 2016, Dec. 2016, 4 Marks)

Multi-Tape turing machine has multiple tuples with each tape having its own independent head. Let us consider the case of a two tape turing machine. It is shown in Fig. 7.5.2.

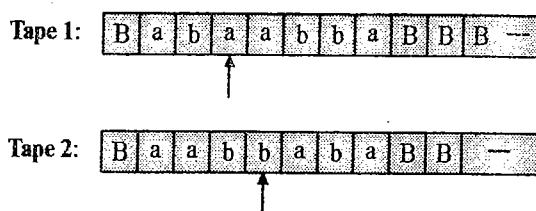


Fig. 7.5.2 : A two-tape turing machine

The transition behavior of a two-tape Turing machine can be defined as given below.

$$\delta(q_1, a_1, a_2) = (q_2, (S_1, M_1), (S_2, M_2))$$

Where,

q_1 is the current state,

q_2 is the next state,

a_1 is the symbol under the head on tape 1,

a_2 is the symbol under the head on tape 2,

S_1 is the symbol written in the current cell on tape 1,

S_2 is the symbol written in the current cell on tape 2,

M_1 is the movement (L,R,N) of head on tape 1,

M_2 is the movement (L,R,N) of head on tape 2.

Example 7.5.1

Construct a two-tape turing machine to recognize words of the form $\omega\omega\omega$ over alphabet {a,b}.

Solution :

Let the initial string be placed on tape 1 and tape 2 may contain all blanks. Initial configuration for $\omega = abb$ is shown in Fig. Ex. 7.5.1(a).

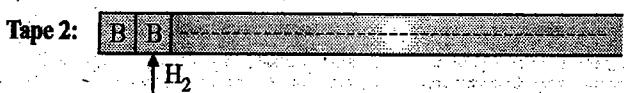
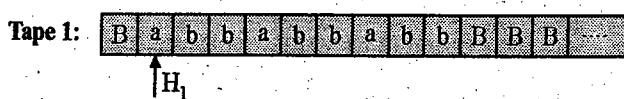


Fig. Ex. 7.5.1(a) : Initial configuration

Step 1 : This step will copy the last ω from $\omega\omega\omega$ on tape 1 to tape 2 in reverse order.

- H_1 is advanced by two places while re-writing a as x and b as y.
- H_1 is moved to the last character and the last character from tape 1 is copied to tape 2, last character of type 1 is erased.

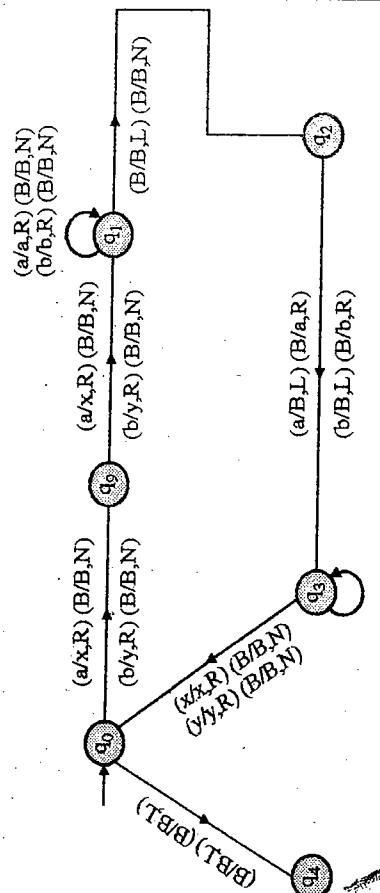
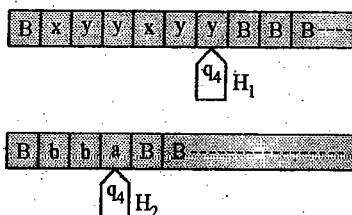


Fig. Ex. 7.5.1(b)

Contents of tapes after step 1 are shown in Fig. Ex. 7.5.1(c).

Fig. Ex. 7.5.1(c) : Last ω from tape 1 is copied as ω^R to tape 2

Step 2 : H_2 is moved to leftmost character on tape 2.

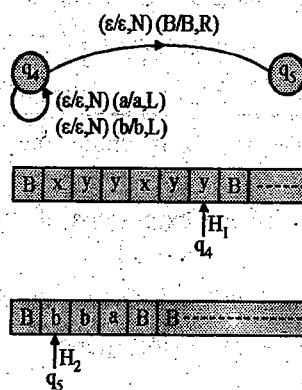


Fig. Ex. 7.5.1(d) : Situation after step 2

Step 3 : Two sets of xyy are matched with bba in the reverse order.

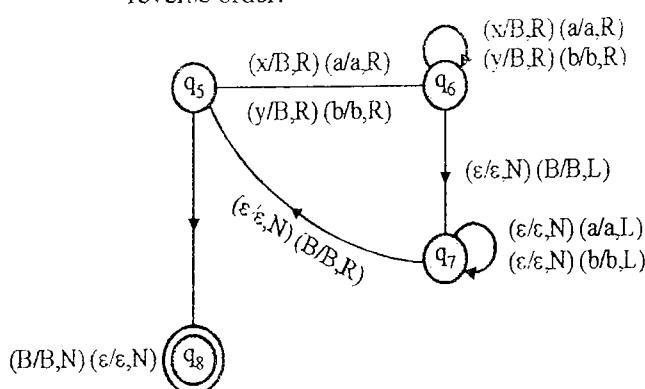


Fig. Ex. 7.5.1(e) : Two sets of xyy are matched with bba in the reverse order

Example 7.5.2

Construct a two-tape turing machine to convert an input ω into $\omega\omega^R$.

Solution : Initially, tape 1 contains ω and the tape 2 is blank String ω is assumed to be abb .

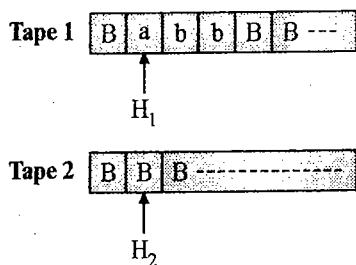
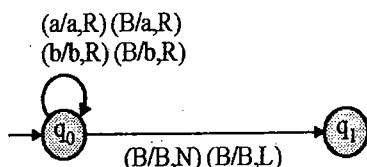


Fig. Ex. 7.5.2(a) : Initial configuration

Step 1 : String abb is copied to tape 2.



Contents of tapes after step 1 are shown in Fig. Ex. 7.5.2(b).

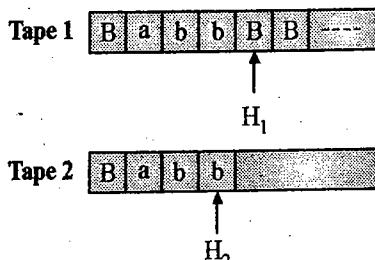
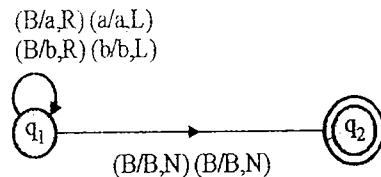


Fig. Ex. 7.5.2(b) : Contents of tapes after step 1

Step 2 : Contents of tape 2 is copied to tape 1, while moving towards left in tape 2 and moving towards right in tape 1.



Contents of tapes after step 2 are shown in Fig. Ex. 7.5.2(c)

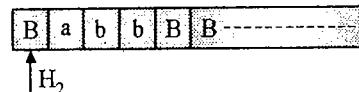
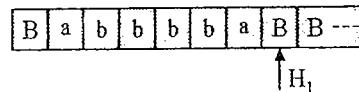


Fig. Ex. 7.5.2(c) : Contents of tapes after step 2

Transition diagram of the turing machine is shown in Fig. Ex. 7.5.2(d)

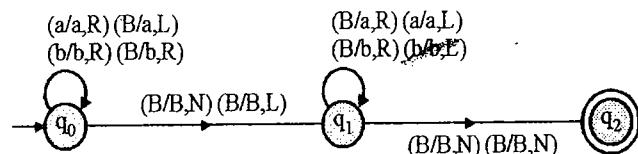


Fig. Ex. 7.5.2(d) : Transition diagram for Example 7.5.2

Syllabus Topic : Non-Deterministic Turing Machine

7.5.4 Non-Deterministic Turing Machine

SPPU - May 12, Dec. 14

University Question

Q. Write short note on : Non-deterministic TM.
(May 2012, Dec. 2014 4 Marks)

Non-deterministic is a powerful feature. A non-deterministic TM machine might have, on certain combinations of state and symbol under the head, more than one possible choice of behaviour.

- Non-deterministic does not make a TM more powerful.
- For every non-deterministic TM, there is an equivalent deterministic TM.
- It is easy to design a non-deterministic TM for certain class of problems.

- A string is said to be accepted by a NDTM, if there is at least one sequence of moves that takes the machine to final state.
- An example of non-deterministic move for a TM is shown below.

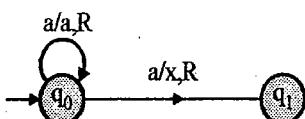


Fig. 7.5.3 : A sample move for NDTM

The transition behaviour for state q_0 for TM of Fig. 7.5.3 can be written as

$$\delta(q_0, a) = \{(q_0, a, R), (q_1, x, R)\}$$

Example 7.5.3

Construct NDTM to recognize words of the form $\omega\omega$ over alphabet {a,b}

Solution :

First character of the second ω in $\omega\omega$ can be located non-deterministically.

1. TM skips the first ω and re-writes every a as x and every b as y for every a, b belonging to first ω of $\omega\omega$. Contents of tape after step 1 are shown in Fig. Ex. 7.5.3(b).

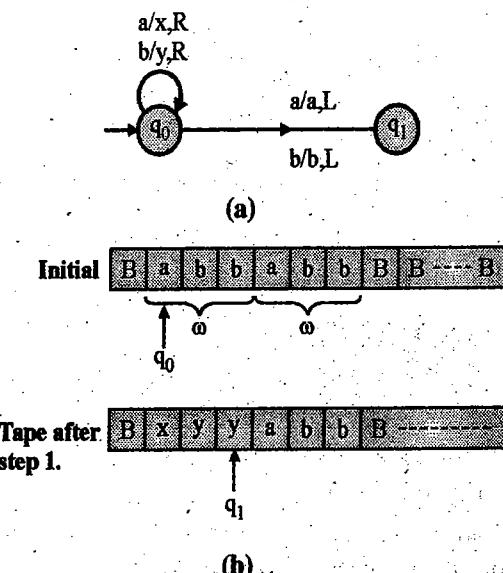


Fig. Ex. 7.5.3

2. Head is sent back to the first character and then first half is matched with the second half while matching the two ω 's every character of second ω is re-written as z.

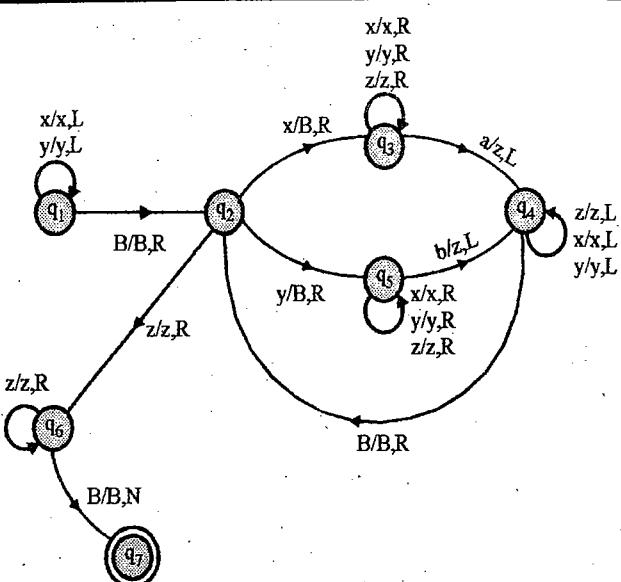
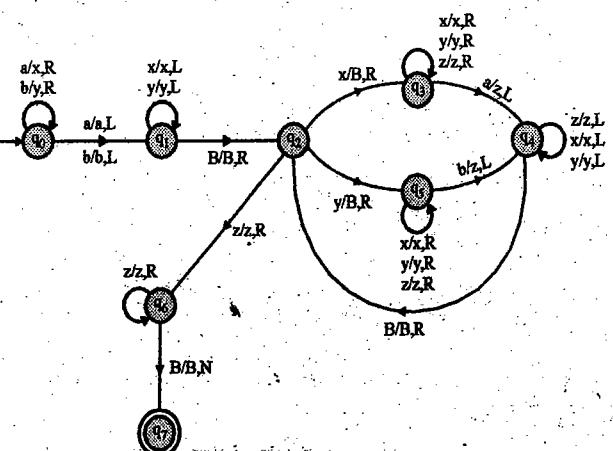


Fig. Ex. 7.5.3(c)

- A cycle through $q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_2$ matched x with a.
- A cycle through $q_2 \rightarrow q_5 \rightarrow q_4 \rightarrow q_2$ matched y with b.
- After every x, y \in first ω is matched, machine moves to state q_6 .
- The state q_6 is to establish that every character from the second ω of $\omega\omega$ has been matched.
- q_7 is the final state.
- The final TM is shown in Fig. Ex. 7.5.3(d).

Fig. Ex. 7.5.3(d) : NDTM for $\omega\omega$



Syllabus Topic : Universal Turing Machine

7.6 Universal Turing Machine

SPPU - Dec. 13, Dec. 14, Dec. 15, May 16, Dec. 16

University Question

Q. Write a short note on : Universal TM.
(Dec. 2013, Dec. 2014, May 2016, Dec. 2016, 3 Marks)

A general-purpose computer can be programmed to solve different types of problems. A TM can also behave like a general-purpose computer. A general purpose computer solves a problem as given below :

1. A program is written in a high level language and its machine-code is obtained with the help of a compiler.
2. Machine code is loaded in main memory.
3. Input to the program can also be loaded in memory.
4. Program stored in memory is executed line by line. Execution involves reading a line of code pointed by IP (instruction pointer), decoding the code and executing it.

We can follow a similar approach for a TM. Such, a TM is known as **Universal Turing Machine**. Universal Turing Machine (UTM) can solve all sorts of solvable problems.

- A Turing machine M is designed to solve a particular problem p, can be specified as :
 1. The initial state q_0 of the TM M.
 2. The transition function δ of M can be specified as given :

If the current state of M is q_i and the symbol under the head is a_i , then the machine moves to state q_j while changing a_i to a_j . The move of tape head may be :

1. To-left,
2. To-Right or
3. Neutral

Such a move of TM can be represented by tuple

$$\{(q_i, a_i, q_j, a_j, m_f) : q_i, q_j \in Q; a_i, a_j \in \Gamma; m_f \in \{\text{To-left, To-Right, Neutral}\}\}$$

- UTM should be able to simulate every turing machine. Simulation of a Turing will involve :

1. Encoding behaviour of a particular TM as a program.
2. Execution of the above program by UTM.

- A move of the form $(q_i, a_i, q_j, a_j, m_f)$ can be represented as $10^{i+1} 10^i 10^{j+1} 10^j 10^K$,

Where $K = 1$, if move is to the left

$K = 2$, if move is to the right

$K = 3$, if move is 'no-move'

State q_0 is represented by 0,

State q_1 is represented by 00,

State q_n is represented by 0^{n+1} .

First symbol can be represented by 0,

Second symbol can be represented by 00 and so on.

Two elements of a tuple representing a move are separated by 1.

- Two moves are separated by 11.

Execution by UTM : We can assume the UTM as a 3-tape turing machine.

1. Input is written on the first tape.
2. Moves of the TM in encoded form is written on the second tape.
3. The current state of TM is written on the third tape.

The control unit of UTM by counting number of 0's between 1's can find out the current symbol under the head. It can find the current state from the tape 3.

Now, it can locate the appropriate move based on current input and the current state from the tape 2.

Now, the control unit can extract the following information from the tape 2 :

1. Next state
2. Next symbol to be written
3. Move of the head.

Based on this information, the control unit can take the appropriate action.



7.6.1 Church-Turing Hypothesis

SPPU Dec.12

University Question

- Q.** Write short note on : Church Turing hypothesis.
(Dec. 2012, 5 Marks)

The Turing machine is a general model of computation. Any algorithmic procedure can be solved by a computer can also be solved by a TM. Problems computed by a computer or a TM are also known as partial recursive functions. Some enhancements to TM made the Church-Turing thesis acceptable. These enhancements are :

1. Multi-tape
2. Multi-head
3. Infinite tapes
4. Non-determinism.

Since the introduction of TM, no one has suggested an algorithm than can be solved by a computer but cannot be solved by a TM.

Syllabus Topic : Comparisons of All Automata

7.6.2 Power of Various Machines

- A post machine can simulate functioning of a turing machine. Hence a turing machine and a post machine have equal power.
- A post machine is more powerful than a PDA. A string of the $a^n b^n c^n$ can be handled by a post machine but it can not be handled by a PDA.
- An NPDA is more powerful than a DPDA. A string of the form $\omega\omega^R$ can be handled by an NPDA but it cannot be handled by a DPDA.
- DPDA is more powerful than FA. A string of the form $a^n b^n$ can be handled by a DPDA but it cannot be handled by FA.

Example 7.6.1

Explain the power of turing machine over finite state machine.

Solution :

1. Turing machine is more powerful than FA.
2. FA has no memory; Turing machine has additional memory in the form of a tape.

3. FA cannot modify its input. A TM can modify its own input.
4. FA cannot be used for arithmetic operations. A TM can perform arithmetic operations.
5. A language accepted by finite automata is regular language. It cannot handle context free language, context sensitive or recursive languages. These languages can be handled by a TM.

Example 7.6.2 SPPU - May 15, 4 Marks

Compare FA, PDA and TM.

Solution :

| Sr. No. | FA | PDA | TM |
|---------|--------------------------------------|--------------------------------------|-----------------------------------|
| 1. | Accepts regular language | Accepts context tree language | Accepts recursive language. |
| 2. | No memory | Memory in the form of stack. | Memory in the form of tape. |
| 3. | Cannot modify input. | Cannot modify input. | Can modify input. |
| 4. | Cannot perform arithmetic operation. | Cannot perform arithmetic operation. | Can perform arithmetic operation. |
| 5. | Less powerful than PDA and TM | Less powerful than TM. | More powerful than FA and PDA. |

Syllabus Topic : Recursive Language and Recursively Enumerable Languages

7.7 Recursively Enumerable and Recursive Language

SPPU - Dec. 14, May 15, Dec. 15, May 16, Dec. 16

University Questions

- Q.** Explain the recursive sets. **(Dec. 2014, 4 Marks)**
- Q.** Define and differentiate recursive languages and recursively enumerable languages. **(May 2015, 6 Marks)**
- Q.** Write a note on : Recursive language. **(Dec. 2015, 2 Marks)**



- Q.** Write short note on recursive language with suitable example. (May 2016, 3 Marks)
- Q.** Explain recursive language with suitable example. (Dec. 2016, 4 Marks)

There is a difference between recursively enumerable (Turing Acceptable) and recursive (Turing Decidable) language.

- Following statements are equivalent :
 1. The language L is **Turing acceptable**.
 2. The language L is **recursively enumerable**.
- Following statements are equivalent :
 1. The language L is **Turing decidable**.
 2. The language L is **recursive**.
 3. There is an algorithm for recognizing L.
- Every Turing decidable language is Turing acceptable.
- Every Turing acceptable language need not be Turing decidable.

7.7.1 Turing Acceptable Language

SPPU - Dec. 15

University Question

- Q.** Define decidability of problem with suitable example. (Dec. 2015, 4 Marks)

A language $L \subseteq \Sigma^*$ is said to be a Turing Acceptable language if there is a Turing machine M which halts on every $\omega \in L$ with an answer 'YES'. However, if $\omega \notin L$, then M may not halt.

Turing Decidable Language

A language $L \subseteq \Sigma^*$ is said to be turing being decidable if there is a turing machine M which always halts on every $\omega \in \Sigma^*$. If $\omega \in L$ then M halts with answer 'YES', and if $\omega \notin L$ then M halts with answer 'NO'.

- A set of solutions for any problem defines a language.
- A problem P is said to be decidable /solvable if the language $L \subseteq \Sigma^*$ representing the problem (set of solutions) is turing decidable.
- If P is solvable / decidable then there is an algorithm for recognizing L, representing the problem. It may be noted that an algorithm terminates on all inputs.
- Following statements are equivalent :

1. The language L is Turing decidable.
2. The language L is recursive.
3. There is an algorithm for recognizing L.

Remarks

1. Every turing decidable language is turing acceptable.
2. Every turing acceptable language need not be turing decidable.
3. A language $L \subseteq \Sigma^*$ many not be turing acceptable and hence not turing decidable. Thus we cannot design a turing machine / algorithm which halts for every $\omega \in L$.

Example 7.7.1 SPPU - Dec. 14, 8 Marks

Show that for two recursive languages L_1 and L_2 , each of the following is recursive.

- (i) $L_1 \cup L_2$ (ii) $L_1 \cap L_2$ (iii) L'_1

Solution :

- (i) $L_1 \cup L_2$ is recursive

Let the turing machine M_1 decides L_1 and M_2 decides L_2 .

If a word $\omega \in L_1$ then M_1 returns "Y" else it returns "N". Similarly, if a word $\omega \in L_2$ then M_2 returns "Y" else it returns "N".

Let us construct a turing machine M_3 as shown in Fig. Ex. 7.7.1(a).

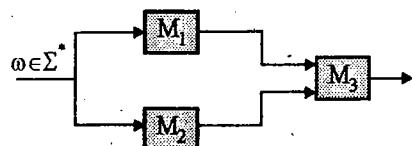


Fig. Ex. 7.7.1(a) : A turing machine for $L_1 \cup L_2$

- Output of machine M_1 is written on the tape of M_3 .
- Output of machine M_2 is written on the tape of M_3 .
- The machine M_3 returns "Y" as output, if at least one of the outputs of M_1 , or of M_2 is "Y".

It should be clear that M_3 decides $L_1 \cup L_2$. As both L_1 and L_2 are turing decidable, after a finite time both M_1 and M_2 will halt with answer "Y" or "N". The



machine M_3 is activated after M_1 and M_2 are halted. The machine M_3 halts with answer "Y" if $\omega \in L_1$ or $\omega \in L_2$, else M_3 halts with output "N".

Thus $L_1 \cup L_2$ is turing decidable or $L_1 \cup L_2$ is recursive.

(ii) $L_1 \cap L_2$ is recursive

Let the turing machine M_1 decides L_1 and M_2 decides L_2 . If a word $\omega \in L$, then M_1 returns "Y" else it returns "N". Similarly, if a word $\omega \in L_2$ then M_2 returns "Y" else it returns "N".

Let us construct a turing machine M_4 as shown in Fig. Ex. 7.7.1(b).

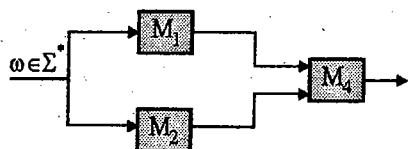


Fig. Ex. 7.7.1(b) : A turing machine for $L_1 \cap L_2$

- Output of machine M_1 is written on the tape of M_4 .
- Output of machine M_2 is written on the tape of M_4 .
- The machine M_4 returns "Y" as output, if both outputs of M_1 and M_2 are "Y"; otherwise, M_4 returns "N".

It should be clear that M_4 decides $L_1 \cap L_2$. As both L_1 and L_2 are turing decidable, after a finite time both M_1 and M_2 will halt with answer "Y" or "N". The machine M_4 is activated after M_1 and M_2 are halted. The machine M_4 halts with answer "Y" if $\omega \in L_1$ and $\omega \in L_2$, else M_4 halts with answer "N".

(iii) L'_1 is recursive

Let the turing machine M_1 decide L_1 .

Let us construct a turing machine M_5 as shown in Fig. Ex. 7.7.1(c).

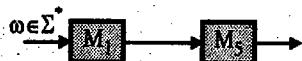


Fig. Ex. 7.7.1(c) : A turing machine for L'_1

- Output of machine M_1 is written on the tape of M_5 .

- The machine M_5 returns "Y" as output, if the output of M_1 is "N"; otherwise, M_5 returns "N".

It should be clear that M_5 decides L'_1 . As L is turing decidable, after a finite time M_1 will halt with answer "Y" or "N". The machine M_5 is activated after M_1 halts. The machine M_5 halts with answer "Y" if $\omega \notin L_1$, else M_5 halts with answer "N".

Example 7.7.2

Show that the language $L = \{a^n b^n c^n \mid n \geq 0\}$ is turing decidable.

Solution : In order to prove that L is turing decidable, we must construct TM accepting both L and L' . We can construct a 3-tape TM. The construction is given below.

Step 1 : b^n is copied to tape 2 and c^n is copied to tape 3 using the following moves. b 's and c 's are erased from the first tape. Initially the three heads H_1 , H_2 and H_3 are positioned on the leftmost symbol.

$$\delta(q_0(B,B,B)) = (q_1,(B,B,B),(R,R,R))$$

[Skip the first blank]

$$\delta(q_1,(a,B,B)) = (q_1,(a,B,B),(R,N,N))$$

[skip a's on tape 1]

$$\delta(q_1,(b,B,B)) = (q_2,(B,b,B),(R,R,N))$$

[copy first b to tape 2]

$$\delta(q_2,(b,B,B)) = (q_2,(B,b,B),(R,R,N))$$

[copy remaining b's to tape 2]

$$\delta(q_2,(c,B,B)) = (q_3,(B,B,c),(R,N,R))$$

[copy first c to tape 3]

$$\delta(q_3,(c,B,B)) = (q_3,(B,B,c),(R,N,R))$$

[copy remaining c's to tape 3]

$$\delta(q_3,(B,B,B)) = (q_4,(B,B,B),(L,N,N))$$

Step 2 : H_1 is positioned on the rightmost a

$$\delta(q_4,(B,B,B)) = (q_4,(B,B,B),(L,N,N))$$

$$\delta(q_4,(a,B,B)) = (q_5,(a,B,B),(N,L,L))$$

H_2 is positioned on the rightmost b and H_3 is positioned on the rightmost c.



Step 3 : a's on tape 1, b's on tape 2 and c's on tape 3 are matched.

$$\delta(q_5, (a, b, c)) = (q_5, (a, b, c), (L, L, L))$$

$$\delta(q_5, (B, B, B)) = (q_6, (B, B, B), (N, N, N))$$

q_6 is a halting state.

If the state q_6 is reached then the string is in the language, $\{a^n b^n c^n\}$.

If at any stage, the TM does not have any move then the string $\omega \in L'$

Thus a string of the form $a^n b^n c^n$ is turing decidable.

Example 7.7.3

If L_1 and L_2 are two recursive languages and if L is defined as :

$L = \{\omega \mid \omega \text{ is in } L_1 \text{ and not in } L_2, \text{ or } \omega \text{ is in } L_2 \text{ and not in } L_1\}$ Prove or disprove that L is recursive.

Solution : Let the turing machine M_1 decides L_1 and M_2 decides L_2 . If a word $\omega \in L_1$, then M_1 returns "Y" else it returns "N". Similarly, if a word $\omega \in L_2$ then M_2 return "Y" else it returns "N".

Let us construct a Turing machine M_3 as shown in Fig. Ex. 7.7.3.

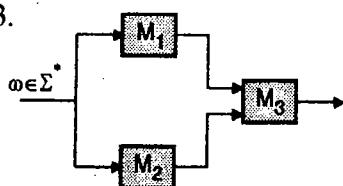


Fig. Ex. 7.7.3

- Output of machine M_1 is written on the tape of M_3 .
- Output of machine M_2 is written on the tape of M_3 .
- The machine M_3 returns "Y" as output, if one of the outputs of M_1 , or M_2 is "Y" and other is "N".
- The machine M_3 returns "N" as output, if the outputs of both M_1 and M_2 are either "Y" or "N".

It should be clear that M_3 decides L . As both L_1 and L_2 are recursive, after a finite time both M_1 and M_2 will halt with answer "Y" or "N". The machine M_3 is activated after M_1 and M_2 are halted. Finally, the machine M_3 will halt with answer "Y" or "N".

Example 7.7.4

Prove the theorem.

"A language is recursive if and only if both it and its complement are recursively enumerable".

Solution :

Let us consider a recursive language L . Let the turing machine M_1 accepts L and M_2 accepts L' .

If a word $\omega \in L$ then M_1 returns "Y" and the machine M_2 returns either "Y" or it loops forever.

Similarly, if a word $\omega \notin L$ then M_2 returns "N" and the machine M_1 returns either "N" or it loops forever.

Let us construct a Turing machine M_3 as shown in Fig. Ex. 7.7.4.

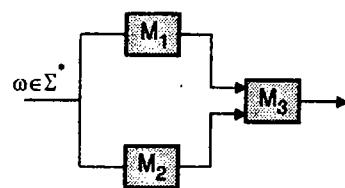


Fig. Ex. 7.7.4

- Output of machine M_1 is written on the tape of M_3 .
- Output of machine M_2 is written on the tape of M_3 .
- The machine M_3 returns "Y" if a value "Y" is found on its tape.
- The machine M_3 returns "N" if a value "N" is found on its tape.

It should be clear that if $\omega \in L$ then at least M_1 will halt with answer "Y". Similarly, if $\omega \notin L$ then, at least M_2 will halt with answer "N".

Thus, if a language and its complement are recursively enumerable then the language is recursive.

If a language is recursive then its complement is also recursive. A recursive language is subset of a recursively enumerable language. Every recursive language is recursively enumerable.

Hence, proved.

7.8 Enumerating a Language

SPPU - Dec. 15, May 16, Dec. 16

University Questions

Q. Write a note on Recursively enumerable language. (Dec. 2015, May 2016, 2 Marks)



- Q.** Explain recursive enumerable language with suitable example. (Dec. 2016, 4 Marks)

A Turing enumerable language can be enumerated by some Turing machine. To enumerate a language means to list the elements one at a time.

A Turing machine enumerating a language

- A Turing machine M can be made to list elements of a language L.
- A multi-tape Turing machine, with K-tapes can be considered for this purpose. Tape 1 can be reserved exclusively for output.
- Let M be a K-tape Turing machine with $K \geq 1$ and $L \subseteq \Sigma^*$. M is made to operate in such a fashion so that the following conditions are satisfied :
 1. The tape head on first tape moves only in the forward direction replacing blank symbols with valid strings $\omega_i \in L$.
 2. For every $\omega_i \in L$, there is some point in the operation of M when tape 1 has following contents :

$$\omega_1 \# \omega_2 \# \dots \# \omega_n \# \omega \#$$

Where, the strings $\omega_1, \omega_2, \dots, \omega_n$ are distinct strings in L.

- (a) If L is finite than nothing is printed after the # following the last word.
- (b) If L is finite than machine can either halt or continue to loop forever after the last word has appeared on the tape.
- (c) If L is infinite, M continues to move forever.

The machine M cannot hope to finish its computation on each string ω_i before beginning to work on the next. If $\omega_i \notin L$ then M may loop forever. The solution for this is given below :

Instead of completing the computation on each string as it is generated, M carries out the following sequence of operations :

1. ω_{i+1} is computed from ω_i where $\omega_{i+1} = \omega_i \Sigma$ for each alphabet in Σ .
2. If ω_{i+1} takes the machine M to an accepting state than ω_{i+1} is written to tape 1.
3. Step 1 and step 2 are carried out infinitely if L is infinite.

7.8.1 Finite and Infinite Sets

Number of elements in a set is also known as its cardinality.

- If $A \subseteq B$, then the size of A is less than or equal to that of B.
- Two sets A and B are said to be equinumerous (having same number of elements) if there is a bijection.
- $f : A \rightarrow B$ [bijection means one-to-one and onto]
- A finite set has finite number of elements.
- A set is infinite if it is not finite.
- The set N of natural numbers is infinite.
- A set S is said to be countably infinite if there is a bijection from N to S.
- $f : N \rightarrow S$ is a bijection.
- To prove that a set S is countably infinite, we must show a bijection between some countably infinite set and the set S.

Example 7.8.1

Show that any subset of a countable set is countable.

Solution :

Let A be a given countably infinite set.

Let S be an infinite subset of the set A.

Obviously, there exists a bijection between the set N of natural numbers and the set A.

$f : N \rightarrow A$, $f(n) = x$ for $x \in A$ is a bijective relation.

Elements of the set A can be arranged as :

$$f(1), f(2), f(3), \dots$$

Now, we can delete from the set A, those elements which are not present in S. The remaining elements in A must be infinite. Let us denote these elements by :

$$f(i_1), f(i_2), \dots$$

Now, we can define a function

$$g : N \rightarrow S \text{ such that } g(n) = f(i_n)$$

Thus, g is one-to-one and onto.

Hence, S is countable.

Example 7.8.2

Show that an infinite recursively enumerable set has an infinite recursive sub-set.

**Solution :**

Let n_1, n_2, n_3, \dots be a recursive enumeration of an infinite set S . For each n_i there must be, in this sequence, a later $n_j > n_i$.

Let $m_1 = n_1$,

$m_2 = n_{i_2}$, the first n_i greater than n_1

$m_3 = n_{i_3}$, the first n_i beyond n_{i_2} greater than n_{i_2} .

⋮

The sequence m_1, m_2, m_3, \dots is a recursive enumeration of a subset of S without repetition in order of magnitude. This subset is infinite and recursive.

Example 7.8.3

Show that if S is uncountable and T is countable then $S-T$ is uncountable.

Solution : We can prove this by contradiction.

Suppose $S-T$ is countable set

$$S-T = \{\omega_1, \omega_2, \dots, \omega_n\}$$

where,

$$\omega_1 = a_{11}, a_{12}, \dots$$

$$\omega_2 = a_{21}, a_{22}, \dots$$

⋮

$$\omega_n = a_{n1}, a_{n2}, \dots$$

It is given that the set T is countable.

$$T = \{x_1, x_2, \dots, x_m\}$$

Since, the set S is uncountable, we can always find a word y such that

$$y \in S, y \notin S-T \text{ and } y \notin T$$

Hence, a contradiction.

Therefore, $S-T$ is uncountable.

Example 7.8.4

Show that the set of languages L over $\{0,1\}$ so that neither L nor L' is recursively enumerable is uncountable.

Solution : To prove that the set of languages over $\{0,1\}$ that are not recursively enumerable is uncountable, we will first prove that the set of recursively enumerable languages is countable.

Proof that set of recursively enumerable language is countable

- Transitions of a Turing machine can be represented by a binary number. If ω_i represents a Turing machine then the set of all Turing machines.

$$T = \{\omega_1, \omega_2, \dots\}$$

Obviously, there exists a bijection between the set N of natural numbers and the set T .

$$f : N \rightarrow T, f(n) = \omega \text{ for } \omega \in T \text{ is a bijective relation}$$

A recursively enumerable language L can be accepted by some Turing machine. For each L , let $g(L)$ be such a Turing machine. Since every TM accepts only one language, the relation g is one-to-one.

RE is subset of T .

Since, T is countable, RE is also countable.

Now, that we have shown the set of recursively enumerable language to be countable; proving that there are uncountable many languages which are not recursively enumerable.

Example 7.8.5

Prove that "The set of real numbers, R , is not countable".

Solution : Let us define a set S .

$$S = \{x | (x \in R) \text{ and } (0 < x < 1)\}$$

Now, the proof can be given in two steps :

1. By proving that S and R have the same cardinality.
2. The set S is not countable.

These two steps will imply that R is not countable.

Step 1 : Proof that S and R have the same cardinality

Let R be the set of all positive real numbers. Let $f : R \rightarrow S$ be defined by $f(x) = \frac{x}{1+x}$ for $x \in R$.

Obviously the range of f is in S .

Further, for any $y \in S$, we have $x = \frac{y}{1-y}$. Thus f is one-to-one and onto. Therefore, the set S and the set R have the same cardinality.

Step 2 : The proof that the set S given by $S = \{x \mid x \in R \text{ and } 0 < x < 1\}$ is not countable.

We can prove this by contradiction.

Suppose S is a countable set.

$$S = \{\omega_1, \omega_2, \dots, \omega_n\}$$

where, $\omega_1 = a_{11} a_{12} a_{13} \dots$

$\omega_2 = a_{21} a_{22} a_{23} \dots$

$\omega_n = a_{n1} a_{n2} a_{n3} \dots$

Assumptions

1. Each ω_i is distinct
2. $a_{ij} \in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Using the principle of diagonalization, we can find a string $\omega_k = x_{11} x_{22} x_{33} \dots$, such that $\omega_k \notin S$.

$$\begin{aligned} x_{ii} &= a_{ii} + 1, && \text{if } a_{ii} < 5 \\ &= 4, && \text{if } a_{ii} \geq 5 \end{aligned}$$

Now, it should be clear that :

ω_k will differ from ω_1 at the first place.

ω_k will differ from ω_2 at the second place.

⋮

ω_k will differ from ω_n at the n^{th} place.

Therefore, the set S is uncountable.

Hence, proved by contradiction.

7.9 Compare Type 0, Type 1, Type 2 and Type 3 Grammars

| Type | Name of Languages Generated | Restriction on Productions | Accepting Machine |
|------|-----------------------------|--|---|
| 0 | Recursively enumerable | $\alpha_1 \rightarrow \alpha_2$ where $\alpha_1, \alpha_2 \in (V \cup T)^*$ | Turing machine |
| 1 | Context sensitive | $\alpha_1 \rightarrow \alpha_2$ where $\alpha_1, \alpha_2 \in (V \cup T)^*$ and $ \alpha_1 \leq \alpha_2 $ | Turing machine with bounded tape. Length of the tape in finite |
| 2 | Context free | $X \rightarrow \alpha_1$ where $X \in V$ and $\alpha_1 \in (V \cup T)^*$ | PDA |
| 3 | Regular | $X \rightarrow aY a\varepsilon Ya$ where, $X, Y \in V$ and $a \in T$ | FA |

A general hierarchy of various languages is given below :

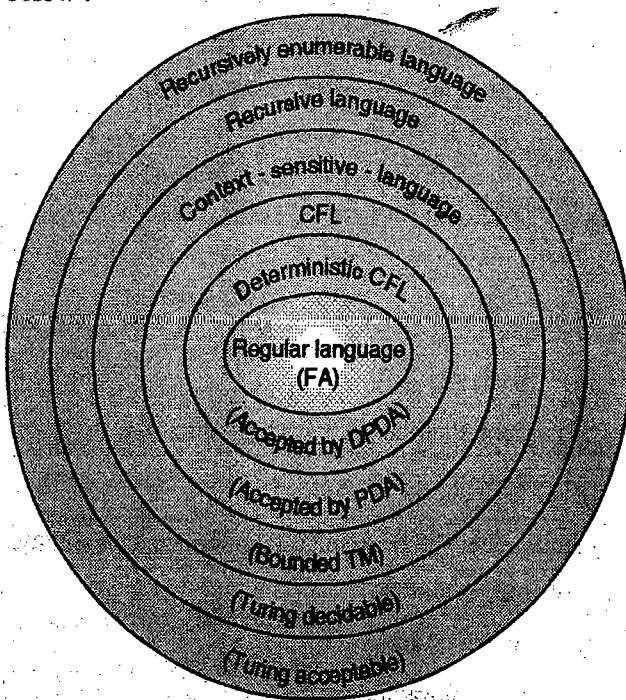


Fig. 7.9.1 : Hierarchy of languages

CHAPTER
8

Computational Complexity

Syllabus

Decidability : Decidable problems concerning regular languages, Decidable problems concerning context-free languages, Un-decidability, Halting Problem of TM, A Turing-unrecognizable language.

Reducibility : Un-decidable Problems from Language Theory, A Simple Un-decidable Problem PCP, Mapping Reducibility

Time Complexity : Measuring Complexity, The Class P, Examples of problems in P, The Class NP, Examples of problems in NP, NP-completeness.

Syllabus Topic : Un-decidability, Un-decidable Problems from Language Theory**8.1 Un-decidability**

SPPU - May 12, May 13, May 14

University Questions

- Q. Write a short Note : Un-decidability.
(May 2012, May 2014, 4 Marks)
- Q. Define undecidability. **(May 2013, 2 Marks)**

A problem is said to be decidable if there exists a Turing machine that gives the correct answer for every statement in the domain of the problem. Otherwise, the class of problems is said to be un-decidable. These two statements are equivalent :

1. A class of problems is un-decidable.
2. A class of problems is un-solvable.

How to prove that a given language is un-decidable ?

A language can be proved to be un-decidable through a method of reduction. We have already seen that the halting problem is un-decidable.

- To show that a problem A is un-decidable, we must reduce another problem that is known to be un-decidable to A.
- Having proved that the halting problem is un-decidable, we can use problem reduction to show that other problems are un-decidable.

First of all, we will provide proof for the un-decidability of some standard problems. Subsequently, these problems will be used to show that other problems are un-decidable.

Some standard un-decidable problems are :

1. Halting problem of a Turing machine.
2. Diagonalization language.
3. The post correspondence problem.
4. The universal language.

Semi-solvability

A class of the problem is said to be semi-solvable if there exists a turing machine which when applied to any problem of the class :

1. The TM always terminates with the correct answer when the answer is yes.
2. The TM may or may not terminate if the answer is no.

Syllabus Topic : Halting Problem of TM

SPPU - May 12, Dec. 12, May 13, Dec.13, Dec. 16

University Questions

- Q. What is halting problem of Turing Machine ?
(May 2012, 4 Marks)
- Q. Define Halting problem of TM with example.
(Dec. 2012, 4 Marks)
- Q. Explain in detail the "Halting problem".
(May 2013, Dec. 2016, 6 Marks)
- Q. Justify "Halting Problem of Turing machine is undecidable".
(Dec. 2013, 5 Marks)

The halting problem of a Turing machine states :

Given a Turing machine M and an input ω to the machine M, determine if the machine M will eventually halt when it is given input ω .



Halting problem of a Turing machine is unsolvable.

Proof

- Moves of a turing machine can be represented using a binary number. Thus, a Turing machine can be represented using a string over $\Sigma^*(0,1)$. This concept has already been explained in the previous chapter.
- Unsolvability of halting problem of a Turing machine can be proved through the method of contradiction.

Step 1 : Let us assume that the halting problem of a Turing machine is solvable. There exists a machine H_1 (say). H_1 takes two inputs :

1. A string describing M .
2. An input ω for machine M .

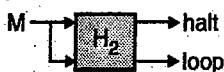
H_1 generates an output "halt" if H_1 determines that M stops on input ω ; otherwise H outputs "loop". Working of the machine H_1 is shown below.



Step 2 : Let us revise the machine H_1 as H_2 to take M as both inputs and H_2 should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.

Step 3 : Let us construct a new Turing machine H_3 that takes output of H_2 as input and does the following :

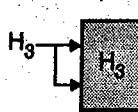
1. If the output of H_2 is "loop" than H_3 halts.
2. If the output of H_2 is "halt" than H_3 will loop forever.



H_3 will do the opposite of the output of H_2 .

Step 4 : Let us give H_3 itself as inputs to H_3 .

If H_3 halts on H_3 as input then H_3 would loop (that is how we constructed it).



If H_3 loops forever on H_3 as input H_3 halts (that is how we constructed it).

In either cases, the result is wrong.

Hence,

H_3 does not exist.

If H_3 does not exist than H_2 does not exist.

If H_2 does not exist than H_1 does not exist.

Example 8.1.1 SPPU - May 14, 4 Marks

Show that the following problem is un-decidable. "Given a Turing machine T , T halts on every input string".

Solution : This is also known as totality problem.

This can be proved by showing that the halting problem is reducible to the totality problem.

- That is, if an algorithm can solve the totality problem, it can also solve the halting problem. Since, no algorithm can solve the halting problem, the totality problem must also be unsolvable.

Reduction step (from halting problem to totality problem) :

For any Turing machine M and input ω , we construct M_1 :

1. M_1 takes an arbitrary input, ignores it, and runs M on ω .
2. M_1 halts on all inputs if and only if M halts on input ω .

Thus, M_1 halts on all inputs also tells that M halts on input ω , which is solution to the halting problem. Hence, the totality problem is un-decidable.

Syllabus Topic : Decidable Problems Concerning Regular Languages, Decidable Problems Concerning Context-free Languages, A Simple Un-decidable Problem PCP

8.1.2 Un-decidability of Post Correspondence Problem

SPPU - May 12, Dec. 12, May 13, Dec. 13,

May 14, Dec. 15, Dec. 16

University Questions

Q. Write a short Note on Post correspondence Problem. (May 2012, Dec. 2012, May 2014, Dec. 2015, 4 Marks)

Q. Explain in detail "Post's Correspondence Problem" with the help of example. (May 2013, Dec. 2016, 8 Marks)



Q. Explain in detail Post Correspondence Problem and Modified Post Correspondence Problem with suitable example. **(Dec. 2013, 6 Marks)**

Definition of post correspondence problem (PCP) :
Let A and B be two non-empty lists of strings over Σ . A and B are given as below :

$$A = \{x_1, x_2, x_3, \dots, x_k\}$$

$$B = \{y_1, y_2, y_3, \dots, y_k\}$$

We say, there is a post correspondence between A and B if there is a sequence of one or more integers i, j, k ... m such that :

The string $x_i x_j \dots x_m$ is equal to $y_i y_j \dots y_m$.

Example : Does the PCP with two lists :

$$A = \{a, aba^3, ab\} \text{ and}$$

$$B = \{a^3, ab, b\}$$

have a solution ?

we will have to find a sequence using which when the elements of A and B are listed, will produce identical strings.

The required sequence is (2, 1, 1, 3)

$$A_2 A_1 A_1 A_3 = aba^3 a aab = ab a^6 b$$

$$B_2 B_1 B_1 B_3 = aba^3 a^3 b = aba^6 b$$

Thus, the PCP has solution.

We are accepting the un-decidability of post correspondence problem without proof.

Example 8.1.2 SPPU - Dec. 13, 6 Marks

Prove that there exists no algorithm for deciding whether a given CFG is ambiguous.

Solution : The post correspondence problem can be used to prove the un-decidability of whether a given CFG is ambiguous.

Let us consider two sequences of strings over Σ .

$$A = \{u_1, u_2, u_3, \dots, u_m\}$$

$$B = \{v_1, v_2, v_3, \dots, v_m\}$$

Let us take a new set of symbols a_1, a_2, \dots, a_m such that

$$\{a_1, a_2, \dots, a_m\} \cap \Sigma = \emptyset.$$

Symbols a_1, a_2, \dots, a_m are being taken as index symbols. The index symbol a_i represents a choice of u_i from A and v_i from the list B.

A string of the form $u_i u_j u_k \dots u_m a_1 a_2 \dots a_r$ over alphabet $\Sigma \cup \{a_1, a_2, \dots, a_m\}$ can be defined using the set of productions :

$$G_A = \left\{ A \rightarrow u_1 A a_1 | u_2 A a_2 | \dots | u_m A a_m \right\}$$

Similarly a string of the form $v_i v_j v_k \dots v_m a_1 a_2 \dots a_r$ over alphabet $\Sigma \cup \{a_1, a_2, \dots, a_m\}$ can be defined using the set of productions :

$$G_B = \left\{ B \rightarrow v_1 B a_1 | v_2 B a_2 | \dots | v_m B a_m \right\}$$

Finally, we can combine the languages and grammars of two lists to form a grammar G_{AB} :

- A new start symbol S is added to G_{AB}
- Two new productions are added to G_{AB}

$$S \rightarrow A \quad S \rightarrow B$$

- All productions of G_A and G_B are taken.

Now, we will show that G_{AB} is ambiguous if and only if an instance (A, B) of PCP has a solution.

Assumption : Suppose the sequence i_1, i_2, \dots, i_m is a solution to this instance of PCP. Two derivations for the above string in G_{AB} is :

$$S \Rightarrow A \Rightarrow u_{i_1} A a_{i_1} \Rightarrow u_{i_1} u_{i_2} A a_{i_1} a_{i_2} \Rightarrow \dots \Rightarrow$$

$$u_{i_1} u_{i_2} \dots u_{i_m} a_{i_1} a_{i_2} \dots a_{i_m}$$

$$S \Rightarrow B \Rightarrow v_{i_1} B a_{i_1} \Rightarrow v_{i_1} v_{i_2} B a_{i_1} a_{i_2} \Rightarrow \dots \Rightarrow$$

$$v_{i_1} v_{i_2} \dots v_{i_m} a_{i_1} a_{i_2} \dots a_{i_m}$$

Consequently, if G_{AB} is ambiguous, then the post correspondence problem with the pair (A, B) has a solution. Conversely, if G_{AB} is unambiguous, then the post correspondence can not have a solution.

If there exists an algorithm for solving the ambiguous problem, then there exists an algorithm for solving the post correspondence problem. But, since there is no algorithm for the post correspondence problem, the ambiguity of CFG problem is unsolvable.

Example 8.1.3

Prove that the blank tape halting problem is undecidable.



Solution : The halting problem of Turing machine can be reduced to blank-tape halting problem. Given a Turing machine M, determine whether or not M halts if started with a blank tape. This is un-decidable.

Reduction : Suppose, we are given some Turing machine M and some string ω . We first construct from M a new machine M_1 that starts with a blank tape, writes ω on it, then positions itself on the first symbol of ω in starting state q_0 .

After this, M_1 will halt on a blank tape if and only if M halts on ω .

Conclusion : Since, the halting problem of the Turing machine is undecidable, the same must be true for the blank tape halting problem.

Example 8.1.4

Let G_1 and G_2 be context free grammars, and r be a regular expression. Then the following are un-decidable :

- (a) $L(G_1) \cap L(G_2) = \emptyset$
- (b) $L(G_1) = L(G_2)$
- (c) $L(G_1) = L(r)$
- (d) $L(G_1) \subseteq L(G_2)$
- (e) $L(r) \subseteq L(G_1)$

Solution :

- (a) $L(G_1) \cap L(G_2) = \emptyset$ is un-decidable

Proof : Let G_1 be a grammar that generates

$$L(G_1) = \{\omega C \omega^R \text{ such that } \omega \in \Sigma^* \text{ and } C \notin \Sigma\}$$

Let $P \subseteq \Sigma^* \times \Sigma^*$, a member of P will be of the form (u_i, v_i) .

Let G_2 be a grammar that generates

$$L(G_2) = \{u_1 u_2 \dots u_n C v_n^R v_{n-1}^R \dots v_1^R \mid n \geq 1 \text{ and } (u_i, v_i) \in P\}$$

- G_1 and G_2 can be constructed easily.

- $L(G_1) \cap L(G_2) = \{\omega C \omega^R \text{ such that } \omega \text{ is match of } P\}$

$L(G_1) \cap L(G_2)$ is the set of solutions to this instance of PCP. The intersection is empty if and only if there is no solution.

If there is an algorithm for finding whether $L(G_1) \cap L(G_2) = \emptyset$ then there exists an algorithm for solving the post correspondence problem. But, since there is no algorithm for the post correspondence problem there is no solution for

$$L(G_1) \cap L(G_2) = \emptyset ?$$

(b) $L(G_1) = L(G_2)$ is un-decidable :

Proof : Un-decidability of $L(G_1) = L(G_2)$ will imply un-decidability of similar unsolvable problem about Turing machine.

Assumptions

$$L(G_1) = L(G_2) \text{ is decidable}$$

Construction of TM for G_1 and G_2

We can design a TM for $L(G_1)$ and similarly for $L(G_2)$. We can design a Turing machine M_1 for $L(G_1)$ as follows :

1. Given an input string ω , machine M_1 saves the string ω on its tape.
2. Machine M_1 can produce all strings over the alphabet of G_1 in lexicographic order.
3. Whenever M_1 produces a string derivable by $L(G_1)$, it compares the last string in derivation with the saved string ω . M_1 halts if they are same otherwise M_1 continues to generate later strings.

Obviously, $L(G_1)$ is Turing acceptable. If a language is Turing acceptable then it must be the output language of some Turing machine.

Let us assume that $L(G_1)$ is acceptable by M_1 and $L(G_2)$ is acceptable by M_2 .

From the assumption, $L(G_1) = L(G_2)$

For every string $\omega_i \in L(G_1)$, ω_i should be in $L(G_2)$ and for every string $\omega_i \in L(G_2)$, ω_i should be in $L(G_1)$.

$\omega_i \in L(G_1)$ is similar to whether M_1 accepts ω_i and $\omega_i \in L(G_2)$ is similar to whether M_2 accepts ω_i .

Solvability of above problem contradicts the halting problem of Turing machine.

- (c) $L(G_1) = L(r)$ is un-decidable

Proof

Let the grammar G_1 and the regular expressions r is defined over the alphabet Σ .

Let the language $L(G_1)$ is the output language of the Turing machine M_1 .

The decidability of $L(G_1) = L(r)$ will imply :

1. For every ω_i in $L(r)$, ω_i is in $L(M_1)$ is Turing decidable.



2. For every ω_i in $\overline{L(r)}$, ω_i is in $\overline{L(M_1)}$ is Turing decidable.

These implications contradict the halting problem of Turing machine.

Hence, $L(G_1) = L(r)$ is un-decidable.

(d) $L(G_1) \subseteq L(G_2)$ is un-decidable

Proof : Let the grammars G_1 and G_2 are defined over the alphabet Σ .

Let the languages $L(G_1)$ and $L(G_2)$ are the output languages of Turing machines M_1 and M_2 respectively.

The decidability of $L(G_1) \subseteq L(G_2)$ will imply :

For every ω_i in $L(G_1)$, ω_i is in $L(M_2)$ is Turing decidable.

This implication contradicts the halting problem of Turing machine.

Hence, $L(G_1) \subseteq L(G_2)$ us un-decidable.

(e) $L(r) \subseteq L(G_1)$ is un-decidable

Proof

Let the grammars G_1 and the regular expression R is defined over the alphabet Σ . Let the language $L(G_1)$ is the output language of the Turing machine M_1 .

The decidability of $L(R) \subseteq L(G_1)$ will imply :

For every ω_i in $L(r)$, ω_i is in $L(M_1)$ is Turing decidable.

This implication contradicts the halting problem of Turing machine.

Hence, $L(r) \subseteq L(G_1)$ is un-decidable.

Example 8.1.5

For following decision problem, determine whether it is decidable or un-decidable, prove the same.

- (1) Given a TM T, does it ever reach a state other than its initial state if it starts with a blank tape ?
- (2) Given a TM T, and a non halting state 'q' of T, does T ever enter state 'q' when it begins with a blank tape ?

Solution :

- (1) We know that the blank tape halting problem is un-decidable. If we can find that given a TM T, reaches a state other than its initial state if it starts with a blank tape. This will make blank tape halting problem as decidable.

Thus, the given decision problem is undividable.

- (2) It is given that q is a non-halting state. It may be noted that a halting state is a case of non-halting state. Thus the solution of the above problem will make blank tape halting problem as decidable.

Thus, the given decision problem is undividable.

8.1.3 Modified PCP Problem

The modified PCP problem is as follows :

Let us consider two lists A and B of K strings each. Each string is from Σ^* .

$$A = x_1, x_2, \dots, x_k \text{ and } B = y_1, y_2, \dots, y_k$$

does there exist a sequence of integers i, i_2, \dots, i_r , such that

$$x_{i_1} x_{i_2} x_{i_3} \dots x_{i_k} = y_1, y_{i_1}, y_{i_2} \dots y_{i_k} ?$$

In the modified PCP, the solution is required to start with the first string on each list.

Syllabus Topic : A Turing Unrecognizable Language

8.1.4 A Turing Unrecognizable Language

- A turing machine M recognizes language L if $L = L(M)$. We say L is turing recognizable if there is a TM M such that $L = L(M)$.
- Let us consider the two languages :
 - $A_{TM} = \{ \langle M, \omega \rangle \mid M \text{ is a TM and } M \text{ accepts } \omega \}$
 - $A_H = \{ \langle M, \omega \rangle \mid M \text{ is a TM and } M \text{ halts on } \omega \}$
- Both A_{TM} and A_H are undecidable but turning recognizable.
- \bar{A}_{TM} and \bar{A}_H are not turing recognizable if they were, then A_{TM} and A_H would be decidable.

Syllabus Topic : Measuring Complexity

8.2 Computational Complexity

SPPU - Dec. 15. Dec. 16

University Questions

- Q. Explain Computational complexity with example. (Dec. 2015, 2 Marks)**
- Q. What is Computational Complexity ? Explain. (Dec. 2016, 4 Marks)**



The time complexity of a Turing machine is given by the function $T(n)$, where

$T(n) = \text{Maximum number of moves made by the TM to process a string of length } n.$

The space complexity of Turing machine is given by the function $S(n)$, where

$S(n) = \text{Maximum number of tape squares used by the TM for an input of length } n.$

Time complexity of a simple Turing machine

Consider the language

$$L = \{\omega c \omega^R \mid \omega \in (0+1)^*\}$$

To recognize a string of the form $\omega c \omega^R$, the TM will require :

Compare first and the last character = $2n + 1$ moves

Compare second character from two ends

$$\therefore = 2(n - 1) + 1 \text{ moves}$$

Find the centre character $c = 1$ move

$$\begin{aligned} \therefore \text{Total number of moves} &= (2n + 1) + (2(n - 1) + 1) \\ &+ \dots + (2(n - (n - 1)) + 1) + 1 \end{aligned}$$

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

$$\therefore T(n) = n^2 + 2n = O(n^2)$$

The time complexity can be reduced by taking a two tape machine.

- The machine copies the input string left of c onto the second tape.
- When c is found on the input tape, the TM moves its second tape head to the left.
- Input tape head continues moving to its right and second tape head to its left.
- The symbols under the two heads are compared as the heads move.
- If all the symbols match and the centre character is c then the string is accepted.

The TM makes a maximum of $n + 1$ moves.

$$\therefore T(n) = n + 1 = O(n)$$

The space complexity of TM is given by

$$S(n) = n + 1 \quad [n - \text{string length and one blank symbol}]$$

Nondeterministic time and space complexity

A nondeterministic TM is of time complexity $T(n)$, if no sequence of choices of move causes the machine to make more than $T(n)$ moves. It is of space complexity $S(n)$ if no sequence of choices need more than $S(n)$ tape cells.

8.2.1 P and NP-class Problem

The Classes P and NP : P denotes the class of problems, for each of which, there is at least one known polynomial time deterministic TM solving it.

NP denotes the class of all problems, for each of which, there is at least one known non-deterministic polynomial time solution. However, this solution may not be reducible to a polynomial time deterministic TM.

- Time complexity of an algorithm is defined as a function of the size of the problem.
- For comparative study of algorithms, growth rate is considered to be very important.
- Size of a problem is often measured in terms of the size of the input.
- An algorithm with time complexity which can be expressed as a polynomial of the size of the problem is considered to have an efficient solution.
- A problem which does not have any (known) polynomial time algorithm is called an **intractable** problem, otherwise it is called **tractable**.
- A solution by deterministic TM is called an algorithm. A solution by a Non-deterministic TM may not be an algorithm.
- For every non-deterministic TM solution, there is a deterministic TM solution of a problem. But there is no computation equivalence between deterministic TM and non-deterministic TM.

In other words

1. If a problem is solvable in polynomial time by non deterministic TM then there is no guarantee that there exists a deterministic TM that can solve it in polynomial time.

If P is set of tractable problem then $P \subseteq NP$. It follows from the fact that every deterministic TM is a special case of nondeterministic TM.

It is still not known whether $P = NP$ or $P \subset NP$.

8.2.2 Intractable Problems

An algorithm that takes an unreasonably large amount of resources (time / space) are called intractable problems. It is impractical to solve such problems on any conventional computer.

Syllabus Topic : The Class P, Examples of Problems in P, The Class NP, Examples of Problems in NP

8.3 The Classes P and NP

SPPU - Dec. 14, Dec. 15, May 16

University Questions

- Q. What do you mean by class NP problems ?
(Dec. 2014, 4 Marks)
- Q. Write note on NP problems with examples.
(Dec. 2014, 8 Marks)
- Q. Write note on P, NP problems with examples.
(Dec. 2015, 8 Marks)
- Q. What do you mean by NP-Problem ?
(May 2016, 4 Marks)

The class of problem denoted by P are solvable by a Deterministic Turing Machine in polynomial time.

- These problems are feasible or theoretically not difficult to solve by computational means.
- The distinguishing feature of the problems is that for each instance of any of these problems, there exists a deterministic Turing machine that solves the problem having time-complexity as a polynomial function of the size the problem.

The class of problem denoted by NP are solvable by a non-deterministic Turing machine polynomial time.

8.3.1 Problem Solvable in Polynomial Time

The time complexity of a Turing machine T is the function $T(n)$ where for input string of length n, the Turing machine will make a maximum of $T(n)$ moves. When $T(n)$ is a polynomial in n, we say that the problem is solvable in polynomial time. A language L is in class P if there is some polynomial $T(n)$ such that $L = L(M)$ for some deterministic Turing machine M of time complexity $T(n)$.

8.3.2 An Example : Kruskal's Algorithm

A spanning tree of a graph $G = (V, E)$ is a connected subgraph of G having all vertices of G and

no cycles in it. If the graph G is not connected then there is no spanning tree of G. A graph may have multiple spanning trees. Fig. 8.3.2 gives some of the spanning trees of the graph shown in Fig. 8.3.1.

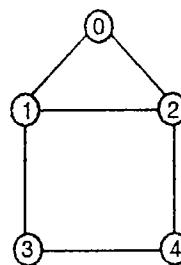


Fig. 8.3.1 : A sample connected graph

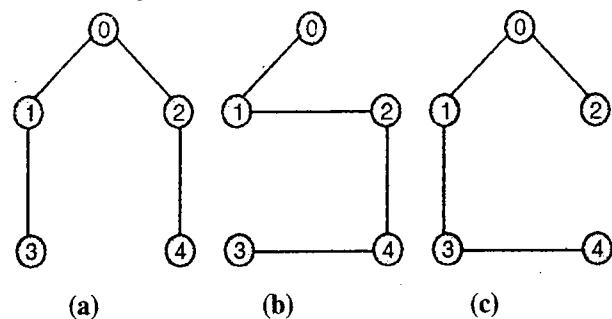


Fig. 8.3.2 : Spanning trees of the graph of Fig. 8.3.1

8.3.3 Minimal Spanning Tree

The cost of a graph is the sum of the costs of the edges in the weighted graph. A spanning tree of a group $G = (V, E)$ is called a minimal cost spanning tree or simply the minimal spanning tree of G if its cost is minimum.

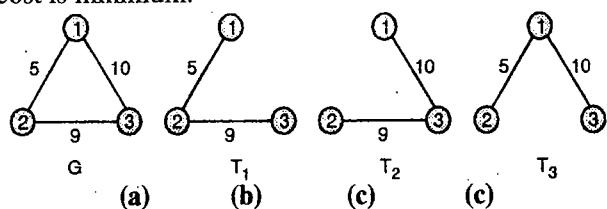


Fig. 8.3.3 : An example of minimal spanning tree

$G \rightarrow$ A sample weighted graph

$T_1 \rightarrow$ A spanning tree of G with cost $5 + 9 = 14$

$T_2 \rightarrow$ A spanning tree of G with cost $10 + 9 = 19$

$T_3 \rightarrow$ A spanning tree of G with cost $5 + 10 = 15$

Therefore, T_1 with cost 14 is the minimal cost spanning tree of the graph G.

There are two popular techniques for constructing a minimum cost spanning tree from a weighted graph $G = (V, E)$.

1. Prim's algorithm
2. Kruskal's algorithm

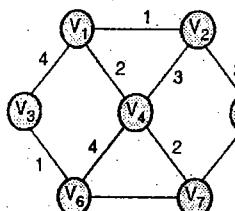
8.3.4 Kruskal's Algorithm

SPPU - May 16

University Question

Q. What is Kruskal's Algorithm ? (May 2016, 4 Marks)

It is another method for finding the minimum cost spanning tree of the given graph. In kruskal's algorithm, edges are added to the spanning tree in increasing order of cost. If any selected edge e forms a cycle in the spanning tree, it is discarded. Fig. 8.3.4 shows the sequence in which the edges are added to the spanning tree.



(a) Given graph G

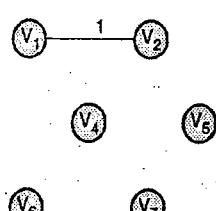
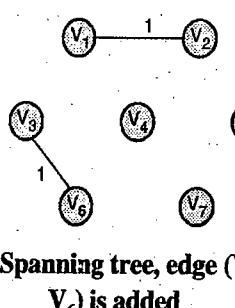
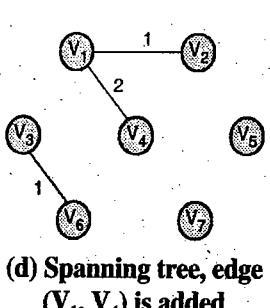
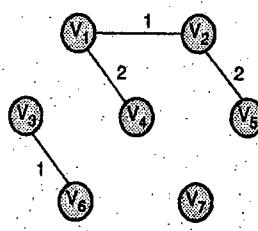
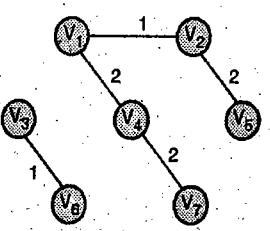
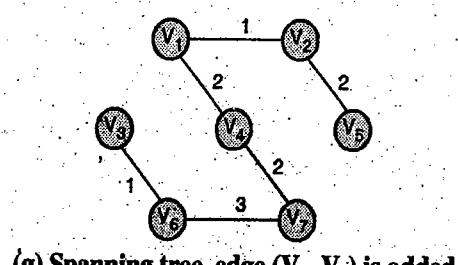
(b) Spanning tree, edge (V₁, V₂) is added(c) Spanning tree, edge (V₃, V₆) is added(d) Spanning tree, edge (V₁, V₄) is added(e) Spanning tree, edge (V₂, V₅) is added(f) Spanning tree, edge (V₄, V₇) is added(g) Spanning tree, edge (V₆, V₇) is added

Fig. 8.3.4 : A graph G and its minimum cost spanning tree

Algorithm :

1. Arrange the edges of the graph G in ascending order of weight. Let the sequence be given by e_1, e_2, \dots, e_k .
2. Let the graph $G_1 = (V, E)$ has n vertices. We have to construct a minimum cost spanning tree $G_T = (V_T, E_T)$.
Initially,
 $V_T = V$
and $E_T = \{ \}$
3. for every edge e_i in (e_1, e_2, \dots, e_k)
if e_i does not form a cycle in G_T
then
 $E_T = E_T \cup \{e_i\}$

It may be noted that the above algorithm will terminate after $n - 1$ edges are added to the spanning tree.

8.3.5 Kruskal's Algorithm using a Turing Machine

SPPU - May 16

University Question

Q. How can Kruskal's algorithm solve this problem using Turing Machine ? (May 2016, 4 Marks)

Kruskal's algorithm can be implemented using a multitape TM. To implement the Kruskal's algorithm, we maintain a list of components. An edge of minimum weight is selected to connect two components. Initially, every node is in its a component by itself.

1. One tape of TM can be used to store every node with its current component. This list will be of the length $O(n)$.
2. A tape can be used for finding the least edge-weight among the edges which have not been used in the spanning tree. This can be done in $O(n)$ time.
3. When an edge is selected, its 2 vertices are copied on a tape. Then we look for the components of the two vertices. This can be done in $O(n)$ time.
4. If the two components (i and j) found in the previous step are not the same component then they can be merged into a single component with the help of another tape. This can be done in $O(n)$ time.



Using the above algorithm, we can find a MST in n rounds. Thus a multitape TM will require $O(n^2)$ to compute MST. Thus the given problem is in P.

8.3.6 Polynomial-Time Reduction

SPPU - May 15, Dec. 15, May 16

University Questions

- Q.** Explain in detail, the polynomial-time reduction approach for proving that a problem is NP-Complete. (May 2015, 8 Marks)
- Q.** What do you mean by Polynomial-time reductions? Describe any problem in detail that is solvable through polynomial time reduction. Explain with suitable example. (Dec. 2015, May 2016, 8 Marks)

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. A polynomial-time computable function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is called a polynomial-time reduction from L_1 to L_2 if and only if, for each $x \in L_1$, $f(x) \in L_2$.

A **polynomial-time reduction** is a polynomial-time algorithm which constructs the instance of a problem P_2 from the instances of some other problem P_1 .

- A problem P_1 equivalently represents a language L_1 .
- We say that a problem P_2 can be solved in polynomial time if we can reduce, another problem P_1 , which is known not be in P to P_2 .

Syllabus Topic : NP-completeness

8.3.7 NP-Complete Problems

A problem P or equivalently its language L_1 is said to be NP-complete if the following two conditions are satisfied.

1. The problem L_2 is in the class NP
2. For any problem L_2 in NP, there is a polynomial-time reduction of L_1 to L_2 .

More explicitly, a problem is NP-complete if it is in NP and for which no polynomial-time Deterministic TM solution is known so far.

The most interesting aspect of NP-complete problems is that for each of these problems neither, so far it has been possible to design a deterministic polynomial-time TM solving the problem nor it has been possible to show that deterministic polynomial-time TM solution can not exist.

8.4 An NP-Complete Problem

Some of the NP-complete problems are listed below without justifying why these problems are in the class.

Problem 1 : Satisfiability problem (in short SAT)

Problem 2 : Traveling salesman problem (TSP)

Problem 3 : Hamiltonian circuit problem (HCP)

Problem 4 : The vertex cover problem (VCP)

Problem 5 : K-colourability problem

Problem 6 : The complete subgraph problem (CSP) or Clique problem

Problem 7 : The subgraph isomorphism problem.

Problem 8 : Exact cover problem

8.4.1 The Satisfiability Problem (SAT)

The satisfiability problem is :

"Given a Boolean expression, is it satisfiable?"

A Boolean expression is said to be satisfiable if at least one truth assignment makes the boolean expression true.

For example : The boolean expression

$((x_1 \wedge x_2) \vee \sim x_3)$ is true for $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$. Therefore, $((x_1 \wedge x_2) \vee \sim x_3)$ is satisfiable.

8.4.2 Traveling Salesman Problem (TSP)

Given a set of cities $C = \{C_1, C_2, \dots, C_n\}$ with $n > 1$, and a function d which assigns to each pair of cities (C_i, C_j) some cost of traveling from C_i to C_j . Further, a positive integer/real number B is given. There problem is to find a route, covering each city exactly once, with cost at most B .

8.4.3 Tractable and Intractable

SPPU - May 15

University Question

Q. Explain Tractable and In-tractable Problem.

(May 2015, 4 Marks)

Problems which, though theoretically can be solved by computational means, yet are infeasible. These problems require so large amount of computational resources that practically it is not feasible to solve these problems by computational means. These problems are called **intractable** or



infeasible problems. Tractable can be solved using computational means. These problems can be solved within reasonable time and space constraints. We normally expect a tractable problem to be solvable in polynomial time.

Example 8.4.1 SPPU - Dec. 14, 4 Marks

What do you mean by class NP problems? Justify why the Travelling Salesman Problem is a class P or class NP.

Solution : Travelling salesman problem

Given a set of cities $C = \{C_1, C_2, \dots, C_n\}$ with $n > 1$ and a function d which assigns to each pair of cities (C_i, C_j) some cost of traveling from C_i to C_j . Further, a positive integer/real number B is given. The problem is to find a route, covering each city exactly once, with cost at most B .

It is a class NP problem

Given a graph G , an upper bound B , and a possible solution in the form of a Hamiltonian path, it is possible to verify or reject that solution with few additions and a comparison. This can be done in polynomial time. Because a potential solution can be verified or rejected in polynomial time, the travelling salesman problem is NP.

8.5 Representing SAT Instances

A boolean expression is represented using the symbols \wedge , \vee , \neg , the left and right parenthesis and symbols representing variables. An instance of this problem is a logical expression containing variables x_i and the logical connectives \wedge , \vee , \neg and parenthesis.

We can encode instances of SAT in a straightforward way :

1. The symbols \wedge , \vee , \neg , (, and) are represented by themselves.
2. The variable x_i is represented by the variable x followed by a binary number representing i .

For example, the expression

$(x_1 \wedge \neg(x_2 \vee x_3))$ will be represented by the string

$(x_1 \wedge \neg(x_{10} \vee x_{11}))$

An expression whose length is m positions can have a code as long as $n = O(m \log m)$ symbols.

8.5.1 NP-Completeness of the SAT Problem

SPPU - Dec. 14, May 15

University Questions

Q. Write a brief note on NP-Completeness of SAT problem. (May 2015, 8 Marks)

Q. Justify that the SAT Problem is NP-complete. (May 2015, 8 Marks)

In general, the process of establishing a problem as NP-complete is a two step process.

The first step, consists of guessing possible solutions of the instances, one instance at a time, of the problem and then verifying whether the guess actually is a solution or not.

The second step involves designing a polynomial-time algorithm which reduces instances of an already known NP-complete problem to instances of the problem which is intended to be shown as NP-complete.

The above technique of reduction can not be applied unless we already have established at least one problem as NP-complete:

- Stephen cook established satisfiability (SAT) as the first NP-complete problem.
- The proof was based on explicit reduction of the language of any non-deterministic polynomial time TM to the satisfiability problem.

Proof

The first part of the proof is easy. We have to show that SAT is in NP.

1. An expression with n variables can have upto 2^n different combinations of the truth values. We can use the non-deterministic power of a non-deterministic TM to guess a truth assignment of a boolean expression E . At each stage, NTM will have many choices of move. As many as 2^n different ID's can be seen at the end of the guessing process.
2. Evaluation of the boolean expression E for a given set of truth values T can be carried out with the help of a deterministic TM. If the expression E evaluates to 1 then it is accepted. Several branches of the nondeterministic TM may not lead to



acceptance, but if even one satisfying truth assignment is found, the non-deterministic TM accepts.

The second step of proof is being skipped as it is quite lengthy. Interested students may consult any of the reference books.

8.5.2 3-SAT problem

SPPU - Dec. 15

University Question:

Q. Explain 3-SAT problem with example.

(Dec. 2015, 4 Marks)

3SAT is the special case of SAT problems discussed earlier, so it must be in NP. To show that 3SAT is NP complete, we will show that any of the SAT instance can be reduced to instance of 3SAT. By this we mean, we have to show that how to convert the clauses which do not contain three literals into the ones which do.

Let us start with the clauses having 2 literals (x_1, x_2). We can introduce the new dummy literal u such that (x_1, x_2) is equivalent to $(x_1, x_2, u) (x_1, x_2, \bar{u})$.

If clause has only one literal (x), we can recursively introduce literals u_1 and u_2 such that (x) corresponds to $(x, u_1, u_2), (x, u_1, \bar{u}_2), (x, \bar{u}_1, u_2), (x, \bar{u}_1, \bar{u}_2)$.

If the clause has more than three literals $(x_1, x_2, x_3, \dots, x_n)$, we can rewrite it as follows:

$$(x_1, x_2, u_1), (x_3, \bar{u}_1, u_2), (x_4, \bar{u}_2, u_3), \dots,$$

$$(x_{n-2}, \bar{u}_{n-4}, \bar{u}_{n-3}), (x_{n-1}, x_n, \dots, \bar{u}_{n-3})$$

One of the x_i must be true for original clause to be true. To make new arrangement equivalent to this, we shall set all u_i true till x_i encounters in new arrangement. It can be seen that, if original clause is satisfiable, then its equivalent 3SAT representation is satisfiable too.

Now, let us consider the counter case. For original clause to be non-satisfiable, all x_i must be false. To prove this we will start with the counter argument that some of the will be true in 3SAT representation. Assume that last clause is true in 3SAT representation. As x_{n-1} and x_n are false, u_{n-3} must be false for

$(x_{n-1}, x_n, \bar{u}_{n-3})$ clause to be true. For second last clause to be true, u_{n-4} must be false, as x_{n-2} and x_{n-3} are already false. This probation will ensure that when we reach to first clause, all u_i must be false, and none of the x_i is true either. And hence it is proved that if original

clause is not satisfiable, then its equivalent 3SAT representation is also not satisfiable.

This transformation is polynomial time transformation, this $\text{SAT} \leq_p \text{3SAT}$, and hence 3SAT is NP complete problem.

8.6 A Restricted Satisfiability Problem

Assuming the satisfiability (SAT) problem as NP-complete, the rest of the problems that we establish as NP-complete, are established by reduction method.



Fig. 8.6.1

Assuming P is already established as NP-complete, the NP-completeness of Q is established through a polynomial-time reduction from P to Q .

8.6.1 Normal Forms for Boolean Expressions

There are two kinds of normal forms :

1. Conjunctive normal form
2. Disjunctive normal form

A boolean expression is said to be in conjunctive normal form if it is a conjunction (\wedge) of disjunctions (\vee) of literals. A boolean expression in CNF is written in the form

$$\bigwedge_{i,j} b_{i,j}$$

where $b_{i,j}$ either a variable or its negation.

A boolean expression is said to be in disjunctive normal form if it is a disjunction (\vee) of conjunction (\wedge) of literals. A boolean expression in DNF is written in the form

$$\bigvee_{i,j} b_{i,j}$$

where $b_{i,j}$ is either a variable or its negation.

8.6.2 Converting Expressions to CNF

An expression is said to be in K-conjunctive normal form (K-CNF) if it is the product of clauses, each of which is the sum of exactly K distinct literals.

For example :

1. $(x + \bar{y})(y + z)$ is in 2-CNF
 2. xyz is in 1-CNF
 3. $(x + y + z)(x + \bar{y} + \bar{z})(x + \bar{y} + z)$ is in 3-CNF
- CSAT is the problem : given a boolean expression in CNF, is it satisfiable.
 - We have to show that CSAT is NP-complete.
 - In order to reduce SAT to CSAT, we must develop a polynomial-time reduction from SAT to CSAT.
 - This reduction will show that CSAT is NP-complete.
 - Only way, we can reduce SAT to CSAT is to find a method of converting an arbitrary boolean expression to an expression in CNF. All we have to do is take a SAT instance E and convert it to a CSAT instance F such that F is satisfiable if and only if E is.
 - A boolean expression in SAT can be converted to CSAT in two steps :
 1. First, we push all \neg 's down the expression tree i.e., the boolean expression becomes an AND and OR of literals. The rules for the same are :
 - (a) $\neg(E \wedge F) \Rightarrow \neg(E) \vee \neg(F)$ – De Morgan's law
 - (b) $\neg(E \vee F) \Rightarrow \neg(E) \wedge \neg(F)$ – De Morgan's law
 - (c) $\neg(\neg(E)) \Rightarrow E$ – Double negation.
 2. The second step is to write an expression that is the AND and OR of literals as a product of clauses.

8.6.3 Clique Problem is NP-Complete

SPPU - May 16

University Question

Q. What is Clique Problem ? Show that it is a NP-Complete problem. (May 2016, 8 Marks)

Given a graph G and positive integer K, does G have a complete subgraph with K vertices ?

A complete subgraph of a given graph G is a subgraph in which every pair of vertices is adjacent in the subgraph.

The verification of whether every pair of vertices is connected by an edge is done for different pair of vertices by a non-deterministic TM, i.e. in parallel. Hence, it takes only polynomial time because for each of n vertices we need to verify at most $n(n+1)/2$ edges, the maximum number of edges in a graph with n vertices.

We next show that 3-CNF-SAT problem can be transformed to clique problem in polynomial time.

Take an instance of 3-CNF-SAT. An instance of 3-CNF-SAT consists of a set of n clauses, each containing exactly 3 literals, each being either a variable or negated variable. It is satisfiable if we can choose literals in such a way that :

1. At least one literal from each clause is chosen.
2. If literal of form x is chosen, no literal of form $\neg x$ is considered.

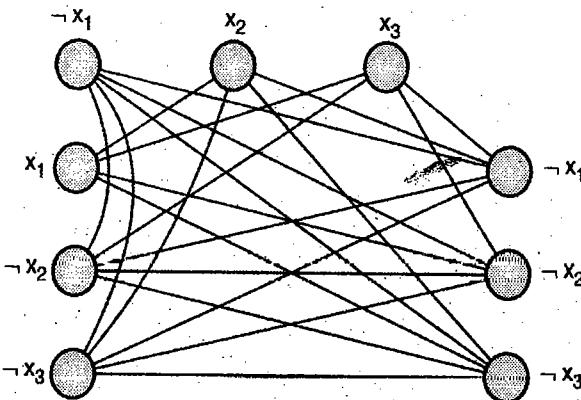


Fig. 8.6.2

For each of the literals, create a graph node, and connect each node to every node in other clauses, except those with the same variable but different sign. This graph can be easily computed from a boolean formula ϕ in 3-CNF-SAT in polynomial time.

Consider an example, we have

$$\begin{aligned}\phi = & (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)\end{aligned}$$

then G is the graph shown in Fig. 8.4.1.

In the given example, a satisfying argument of ϕ is $(x_1 = 0, x_2 = 0, x_3 = 1)$. A corresponding clique of size $k = 3$ consists of the vertices corresponding to x_2 from the first clause, $\neg x_3$ from the second clause, and $\neg x_3$ from the third clause.

The problem of finding n -element clique is equivalent to finding a set of literals satisfying SAT. Because there are no edges between literals of the same clause, such a clique must contain exactly one literal from each clause. And because there are no edges between literals of the same variable but different sign, if node of literal x is in the clique, no node of literal is of form $\neg x$. This proves the finding n -element clique in $3n$ -element graph is NP-complete.

8.6.4 The Problem of Independent Sets

An independent set of a graph $G = (V, E)$ is a subset $V_1 \subseteq V$ of vertices such that no two nodes of V_1 are connected by an edge in G . The independent set problem is to find a maximum size independent set in G .

The independent set problem is the optimization problem of finding an independent set of maximum size in a graph. This problem can also be stated as a decision problem :

INDEPENDENT-SET = { $\langle G, K \rangle \mid G$ has an independent set of at least size K }

To show that the independent set problem \in NP, for a given graph $G = (V, E)$ we take $V_1 \subseteq V$ and verifies to see if it forms an independent set. Verification can be done by checking for $u \in V_1$ and $v \in V_1$, does $(u, v) \in E$. This verification can be done in polynomial time.

Now, we show that clique problem can be transformed to independent set problem in polynomial time. This transformation is based on the notion of the complement of a graph G . Given an undirected graph $G = (V, E)$, we define the complement of G as

$$G_1 = (V, E_1), \text{ where}$$

$E_1 = \{(u, v) \mid (u, v) \notin E\}$ i.e. G_1 is the graph containing exactly those edges that are not in G . The transformation takes a graph G_1 and K of the clique problem. It computes the complement G_1 , which can be done in polynomial time.

To complete the proof, we can show that this transformation is indeed reduction : the graph has a clique of size K if and only if the graph G , has an independent set of size $|V| - K$.

Suppose that G has a clique $V_1 \subseteq V$ with $|V_1| = K$. We claim that $V - V_1$ is an independent set in G_1 . Let (u, v) be an edge in E_1 . Then, $(u, v) \notin E$, which implies that atleast one of u or v does not belong to V_1 , since every pair of vertices in V_1 is connected by an edge of E equivalently, at least one of u or v is in $V - V_1$, which means that edge (u, v) is covered by $V - V_1$. Since (u, v) was chosen arbitrarily from E_1 , every edge of E_1 is covered by a vertex in $V - V_1$. So, either u or v is in $V - V_1$ and no two vertices are in $V - V_1$. Hence, the set $V - V_1$ which has size $|V| - K$, forms an independent set of G_1 .

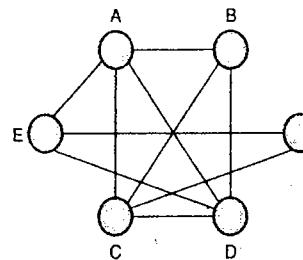


Fig. 8.6.3

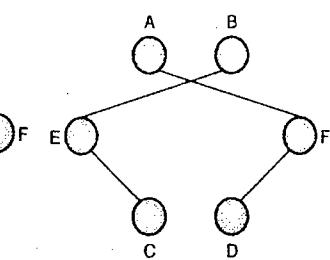


Fig. 8.6.4

For example, the graph $G(V, E)$ has a clique, $\{A, B, C, D\}$ given by Fig. 8.6.3. The complement of graph G is given by G_1 , shown in Fig. 8.6.4 and have independent set given by $\{E, F\}$.

This transformation can be performed in polynomial time. This proves that finding the independent set problem is NP-complete.

8.6.5 The Node-Cover Problem

SPPU - May 15

University Question

Q. Explain the Node-Cover Problem with a suitable example. (May 2015, 8 Marks)

We have to show that node-cover problem is NP-complete.

A node cover of an undirected graph $G = (V, E)$ is a subset V' of the vertices of the graph which contains at least one of the two endpoints of each edge.

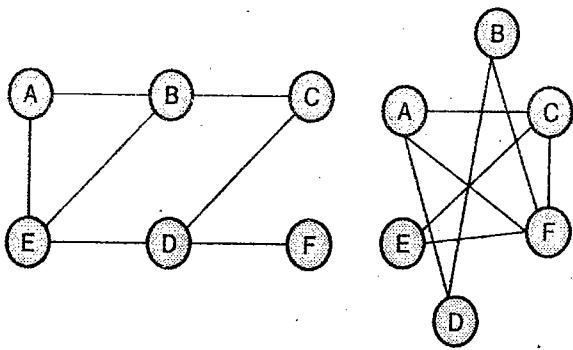


Fig. 8.6.5

Fig. 8.6.6

The node cover problem is the optimization problem of finding a node cover of minimum size in a graph. This problem can also be stated as a decision problem.

NODE - COVER = { $\langle G, K \rangle$ graph G has a vertex cover of size K }

1. To show that node cover problem \in NP, for a given graph $G = (V, E)$ we take $V_1 \subseteq V$ and verifies to see if it forms a node cover. Verification can be done by checking for each edge $(u, v) \in E$, whether $u \in V_1$ or $v \in V_1$. This verification can be done in polynomial time.
2. Now, we show that clique problem can be transformed to node cover problem in polynomial time. This transformation is based on the notion of the complement of a graph G . Given an undirected graph $G = (V, E)$, we define the complement of G as $G_1 = (V, E_1)$ where

$$E_1 = \{(u, v) \mid (u, v) \notin E\}$$

The transformation takes a graph G and K of the clique problem. It computes the complement G_1 , which can be done in polynomial time.

To complete the proof, we can show that this transformation is indeed reduction : the graph has a clique of size K if and only if the graph G_1 has a vertex cover of size $|V| - K$.

Suppose that G has a clique $V_1 \subseteq V$ with $|V_1| = K$. We claim that $V - V_1$ is a vertex cover in G_1 . Let (u, v) be any edge in E_1 . Then $(u, v) \notin E$, which implies that atleast one of u or v does not belong to V_1 , since every pair of vertices in V_1 is connected by an edge in E . Equivalently, atleast one of u or v is in $V - V_1$, which means that edge (u, v) is covered, by a

node in $V - V_1$, which has size $|V| - K$, forms a vertex cover for G_1 .

Conversely, suppose that G_1 has a vertex cover $V_1 \subseteq V$, where $|V_1| = |V| - K$. Then for all $u, v \in V$, if $(u, v) \in E_1$, then $u \in V_1$ or $v \in V_1$ or both.

The contrapositive of this implication is that for all $u, v \in V$, if $u \notin V_1$ and $v \notin V_1$, then $(u, v) \in E$. In other words, $V - V_1$ is a clique, and it has size $|V| - |V_1| = K$.

For example, the graph $G(V, E)$ has a clique $\{A, B, E\}$ given by Fig. 8.4.5. The complement of graph G is given by G_1 shown in Fig. 8.4.6 and has independent set given by $\{C, D, F\}$.

This proves the finding that the node cover is NP-complete.

8.6.6 The Directed Hamilton-Circuit Problem

Finding a Hamilton-circuit in a directed graph is NP-complete. A Hamiltonian circuit of a graph $G = (V, E)$ is a set of edges that connects the nodes into a single cycle, with each node appearing exactly once. It may be noted that the number of edges on a Hamiltonian circuit must be equal to the number of nodes in the graph.

1. The proof of directed Hamiltonian-circuit problem is in NP is easy. Guess a cycle and verify that all the edges are present in the given graph.
2. A directed Hamiltonian-circuit instance can be constructed from a 3-CNF-SAT boolean expression in polynomial time.

8.6.7 Undirected Hamiltonian Circuit

Finding a Hamiltonian-circuit in an undirected graph is NP-complete. A directed Hamiltonian-Circuit Problem (DHC) problem can be reduced to undirected Hamiltonian circuit (HC) problem.

Construction of HC from DHC

1. Suppose G_d is the given directed graph and G_u , the undirected graph is constructed from G_d . For every node V of G_d , there are three nodes V_0 , V_1 and V_2 in G_u .

The edges of G_u are :

1. For all nodes V of G_d , there are edges $(V^{(0)}, V^{(1)})$ and $(V^{(1)}, V^{(2)})$ in G_u .
2. If there is an edge (V, W) in G_d , then there is an edge $(V^{(2)}, W^{(0)})$ in G_u .

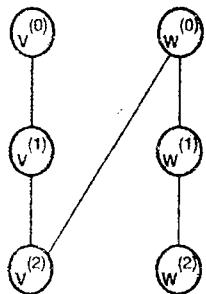


Fig. 8.6.7

Fig. 8.6.7 shows the edges for (V, W) in G_d .

The construction of G_u from G_d can be done in polynomial time.

- G_u has a Hamiltonian circuit if and only if G_d has a directed Hamiltonian circuit.
- Each node $V^{(i)}$ of G_u has only two edges, and therefore must appear in a Hamiltonian circuit.

8.7 Partial Recursive Function

SPPU - Dec. 15

University Question

- Q. Write a note on : Partial recursive function.**
(Dec. 2015, 2 Marks)

Turing machine is viewed as a mathematical model of partial recursive function.

Some basic concepts :

1. Total function

A total function from x to y assigns a unique element of y to every element of x .

$f_1(x) = 2x$ is a **total function**

2. Partial function

A partial function from x to y assigns atmost one element of y to every element of x .

$f_2(x) = +\sqrt{x}$ is a partial function.

$f_2(x)$ is not defined if x is a negative real number.

3. Function of K-variables

A function of K -variables is represented as $f(x_1, x_2 \dots x_k)$.

$f(x_1, x_2) = x_1 + 2x_2$ is a function of two variables.

4. Primitive recursive function

- Some initial functions are taken as primitive recursive functions. These initial functions are

1. Zero function
2. Successor function
3. Projection function

- A function derived from combination/composition of primitive recursive functions is primitive recursive.

The **zero function** Z is defined by $Z(x) = 0$

The **successor function** S is defined by $S(x) = x + 1$

The **projection function** U_i^n is defined by

$$U_i^n(x_1, x_2, \dots, x_n) = x_i$$

The projection function for selecting i^{th} element from a tape containing n elements.

Example 8.7.1

Show that $f(x, y) = x + y$ is primitive recursive.

Solution :

$$f(x, 0) = x + 0 = x = U_1^1(x)$$

Only element x is selected from the tape.

$$\begin{aligned} f(x, y+1) &= f(x, y) + 1 = S(f(x, y)) \\ &= S[U_3^3(x, y, f(x, y))] \end{aligned}$$

Select the third element

Increment it by 1

Thus the function f is obtained by applying composition and recursion finite number of times to initial functions. Hence, f is primitive recursive.

Example 8.7.2

State True or false.

- (i) FSM is a special case of the TM
- (ii) TM can be deterministic



- (iii) TM has an external memory which can remember arbitrary long sequence of inputs.
- (iv) Basic of TM is to divide the process into primitive operations.

Solution :

- (i) True (ii) True (iii) True (iv) True.

Syllabus Topic : Mapping Reducibility

8.8 Turing Reducibility

SPPU - May 15, May 16, Dec. 16

University Questions

- Q. Explain with example turing reduceability.
(May 2015, May 2016, 6 Marks)**
- Q. Write short note on : Turing reducibility.
(Dec. 2016, 4 Marks)**

To show that a language L_1 is not Turing-decidable, we must find a language L_2 , which is not turing-decidable, and then reduce L_2 to L_1 .

Once we have shown that the halting problem is undecidable, we can show that a large class of other problems are undecidable.

We need to understand the concept of problem reduction. Reducing problem P_2 to problem P_1 means finding a way to convert problem P_2 to problem P_1 , so that a solution to problem P_1 can be used to solve problem P_2 .

A reduction of problem P_2 to problem P_1 shows that the problem P_1 is at least as difficult to solve as problem P_2 .

- To show that a problem P_1 is undecidable, we reduce another problem that is known to be undecidable to P_1 .
- Having proved that the halting problem is undecidable, we use problem reduction to show that other problems are undecidable.