Theory of Computation
(Code : 310241)

Semester V - Computer Engineering
(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.

Tech-Max Publications, Pune
Innovation Throughout
Engineering Division

P0264A

Theory of Computation

(Semester V, Computer Engineering, SPPU)

Dilip Kumar Sultania

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 978-93-5224-582-6

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

[310241] (FID : TP461) (Book Code : PO264A)

Syllabus

Savitribai Phule Pune University
Third Year of Computer Engineering (2015 Course)

310241 : Theory of Computation

Teaching Scheme	Credit	Examination Scheme
TH : 03 Hrs/Week	03	In-Sem (paper) : 30 Marks
		End-Sem (paper) : 70 Marks

Prerequisite Courses

Discrete Mathematics (210241), Principles of Programming Languages (210254)

Course Objectives

- To Study abstract computing models
- To learn Grammar and Turing Machine
- To learn about the theory of computability and complexity.

Course Outcomes

On completion of the course, student will be able to

- design deterministic Turing machine for all inputs and all outputs
- subdivide problem space based on input subdivision using constraints
- apply linguistic theory

Course Contents

Unit I : Formal Language Theory and Finite Automata (08 Hours)

Introduction to Formal language, introduction to language translation logic, Essentials of translation, Alphabets and languages, Finite representation of language, Finite Automata (FA) : An Informal Picture of FA, Finite State Machine (FSM), Language accepted by FA, Definition of Regular Language, Deterministic and Nondeterministic FA(DFA and NFA), epsilon- NFA, FA with output : Moore and Mealy machines -Definition, models, inter-conversion.

Case Study: FSM for vending machine, spell checker. **(Refer Chapters 1 and 2)**

Unit II : Regular Expressions (RE) (07 Hours)

Introduction, Operators of RE, Building RE, Precedence of operators, Algebraic laws for RE, Conversions: NFA to DFA, RE to DFA Conversions: RE to DFA, DFA to RE Conversions: State/loop elimination, Arden's theorem Properties of Regular Languages: Pumping Lemma for Regular languages, Closure and Decision properties.

Case Study: RE in text search and replace. **(Refer Chapter 3)**

Unit III : Context Free Grammars (CFG) and Languages **(08 Hours)**

Introduction, Regular Grammar, **Context Free Grammar-** Definition, Derivation, Language of grammar, sentential form, parse tree, inference, derivation, parse trees, ambiguity in grammar and Language- ambiguous Grammar, **Simplification of CFG:** Eliminating unit productions, useless production, useless symbols, and ε-productions, **Normal Forms-** Chomsky normal form, Greibach normal form, Closure properties of CFL, Decision properties of CFL, Chomsky Hierarchy, **Application of CFG:** Parser, Markup languages, XML and Document Type Definitions.

Case Study- CFG for Palindromes, Parenthesis Match. **(Refer Chapter 4)**

Unit IV : Turing Machines (TM) **(08 Hours)**

Turing Machine Model, Representation of Turing Machines, Language Acceptability by Turing Machines, Design of TM, Description of TM, Techniques for TM Construction, Variants of Turing Machines, The Model of Linear Bounded Automata , TM & Type 0 grammars, TM's Halting Problem. **(Refer Chapter 6)**

Unit V : Pushdown Automata(PDA) **(07 Hours)**

Basic Definitions, Equivalence of Acceptance by Finite State & Empty stack, PDA & Context Free Language, Equivalence of PDA and CFG, Parsing & PDA: Top-Down Parsing, Top-down Parsing Using Deterministic PDA, Bottom-up Parsing, Closure properties and Deterministic PDA. **(Refer Chapter 5)**

Unit VI : Undecidability & Intractable Problems **(07 Hours)**

A Language that is not recursively enumerable, An un-decidable problem that is RE, Post Correspondence Problem, The Classes P and NP : Problems Solvable in Polynomial Time, An Example: Kruskal's Algorithm, Nondeterministic Polynomial Time, An NP Example: The Traveling Salesman Problem, Polynomial-Time Reductions, NP Complete Problems, An NP-Complete Problem: The Satisfiability Problem, Tractable and Intractable, Representing Satisfiability, Instances, NP Completeness of the SAT Problem, A Restricted Satisfiability Problem: Normal Forms for Boolean Expressions, Converting Expressions to CNF, The Problem of Independent Sets, The Node-Cover Problem. **(Refer Chapter 7)**

□□□

Chapter 5 : Pushdown Automata (PDA) 5-1 to 5-30

❑❑❑

CHAPTER 1

Unit I

Introduction to Formal Languages

Syllabus

Introduction to formal language, Introduction to language translation logic, Essential of translation, Alphabets and languages, Finite representation of language.

Syllabus Topic : Introduction to Formal Language

1.1 Introduction to Formal Language

SPPU - May 13

University Question

Q. Define formal language with example. **(May 2013, 6 Marks)**

Natural languages are not suitable for computer programming, logic, or in mathematics A Formal language has a fixed set of symbols and formulas, which can precisely specify syntactic and semantic relations in a language. "A formed language is a set of strings of symbols together with a set of rules that are specific to it.

Syllabus Topic : Alphabets and Languages

1.2 Alphabets and Languages

SPPU - May 13

University Question

Q. Define an alphabet with example. **(May 2013, 6 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 sting. 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,

  $$\text{if } \Sigma = \{0, 1\}$$
  $$\text{then } \Sigma^1 = \{0, 1\}$$
  $$\Sigma^2 = \{00, 01, 10, 11\}$$
  $$\text{And } \Sigma^0 = \{\varepsilon\}$$
- The **reversal** of string ω is denoted by ω^R.

  For example,

  $$\text{if } \omega = xyz$$
  $$\text{then } \omega^R = zyx.$$
- 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 ω = 'abcdef' is a string.

  then 'cde' is a substring of ω.
- For each string ω, the string ω^i is defined as,

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

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

For example,

$$\text{if } \omega = abb$$
$$\text{then } \omega^0 = \varepsilon$$
$$\omega^1 = abb$$
$$\omega^2 = \omega \cdot \omega = abbabb$$
$$\omega^3 = \omega^2 \cdot \omega = abbabbabb$$

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

$$\Sigma^* = \underbrace{\Sigma^0}_{\varepsilon} \cup \underbrace{\Sigma^1}_{\text{String of length 1}} \cup \underbrace{\Sigma^2}_{\text{String of length 2}} \ldots\ldots\ldots\ldots$$

For example,

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

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

- 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,

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

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

$\therefore \quad L_1 = \{\varepsilon, 01, 10, 0011, 0101, 1100, 1010, \ldots\}$

Thus, a language can be specified by :

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

Some examples of language are :

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

1.2.1 Kleene Closure **SPPU - May 13**

University Question

Q. Define Kleene closure with examples. **(May 2013, 2 Marks)**

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

$$\Sigma^* = \underbrace{\Sigma^0}_{\varepsilon} \cup \underbrace{\Sigma^1}_{\text{strings of length 1}} \cup \underbrace{\Sigma^2}_{\text{strings of length 2}} \ldots\ldots$$

For example :

1. If $\Sigma = \{x\}$, then
   $\Sigma^* = \{\varepsilon, x, xx, \ldots\}$
2. If $\Sigma = \{0, 1\}$, then
   $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

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

$L^* = \{\varepsilon, b, aa, bb, bbb, baa, aab, \ldots\}$

It may be noted that a always appears in pair.

- If $\Sigma = \phi$ (an empty language)
  then $\Sigma^* = \{\varepsilon\}$
- 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

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

Example 1.2.1

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^*.

1.2.2 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 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

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

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 1.2.2

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)

Syllabus Topic : Finite Representation of Language

1.3 Finite Representation of Language

A finite language can be represented by exhaustive enumeration of all the strings in the language.

This method becomes challenging when infinite languages are considered:

A finite representation of a language involves :

1. A finite sequence of symbols over some alphabet Σ.
2. Different languages should have differed representation for the representation to be appropriate.

Regular expression to represent a language

Every regular expression represents a string.

Example :

$0^* \rightarrow$ represents strings with zero or more 0's

$(0 + 1)^* 1 (0 + 1)^* \rightarrow$ represents strings with one or more 1's.

Regular expressions and the languages they represent can be defined formally and unambiguously. Unfortunately, every language cannot be represented by regular expression.

Content free grammars to represent language

Here 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 G is the set

$$L(G) = \left\{ w \mid w \in T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$$

If the production rule is applied once, then we write. $\alpha \Rightarrow \beta$. When it is applied a number of times then we write $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$.

The productions in a grammar appear as given below :

$S \rightarrow A \mid B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid \epsilon$

Syllabus Topic : Introduction to Language Translation Logic

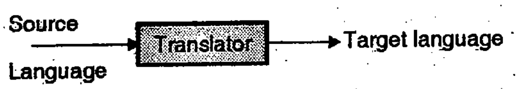
1.4 Introduction to Language Translation Logic

Language processing activity involves translation of a program written in a high level language into machine code. A translator permits the programmer to express his algorithm in a language other than that of a machine. Some of the benefits of writing a program in a language other than a machine language are as follows :

(1) **Increase in the programmer's productivity** – It is easy to write a program in a high level language.

(2) **Machine independence** – A program written in a high level language is machine independent.

The input to a translator is a program written in a source language (high level language). The output of a translator is a program in target language (machine code).



(S1.1)**Fig. 1.4.1 : A translator**

There is a difference between source language and the target language.

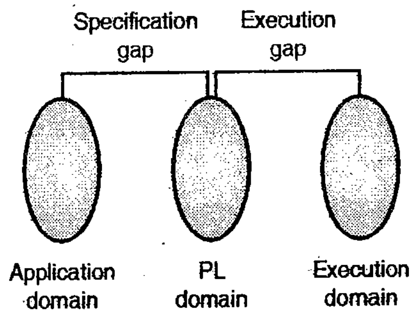
- A source language is closure to application domain. A source language should provide features necessary for expressing an algorithm.
- The ideas expressed in source domain must ultimately be interpreted in terms of execution domain of the computer system.
- Semantic gap between the two domains often implies efforts needed by a translator to convert a program written in source language into machine code. If the semantic gap is less then it is easy to write a translator. But this will have adverse consequences in program development, particularly large development times, large development efforts and poor quality of software.

The development of large programs involve two steps :

(1) Specification, design and writing of algorithm.

(2) Implementation of algorithm through a programming language (PL).

Software development using a programming language introduces a new domain, the PL domain. It is shown in Fig. 1.4.2.



(S1.2)**Fig. 1.4.2 : Specification and execution gap**

- Specification gap can be reduced by selecting a programming language which provides features suited for writing a particular application; but this will increase the execution gap.

Syllabus Topic : Essential of Translation

1.5 Essential of Translation

Translation involves converting a set of strings in the source language into target language.

- The source language must be a formal language. A formal language L can be considered to be a collection of valid sentences. Each sentence can be seen as a sequence of words, and each word as a sequence of alphabets.
- A formal language grammar specifies a set of rules which precisely specify the sentences of a language.
- An algorithm to recognize words of the given formal language.
- An algorithm to recognize sentences of the given formal language.
- An algorithm to convert a valid sentence in source language into the target language.

❑❑❑

CHAPTER 2

Finite Automata

Syllabus

Finite representation of language, Finite Automata (FA): An Informal Picture of FA, Finite State Machine (FSM), Deterministic and Nondeterministic FA(DFA and NFA), epsilon- NFA, FA with output: Moore and Mealy machines -Definition, models, inter-conversion. Case Study: FSM for vending machine, spell checker

Syllabus Topic : Finite Automata (FA) - An Informal Picture of FA, Finite State Machine (FSM)

2.1 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 is 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.

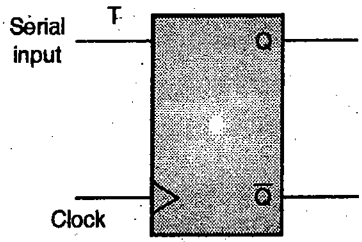


Fig. 2.1.1 : A T-flip flop as a finite automata

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. 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 a finite automata. It is shown in Fig. 2.1.3.

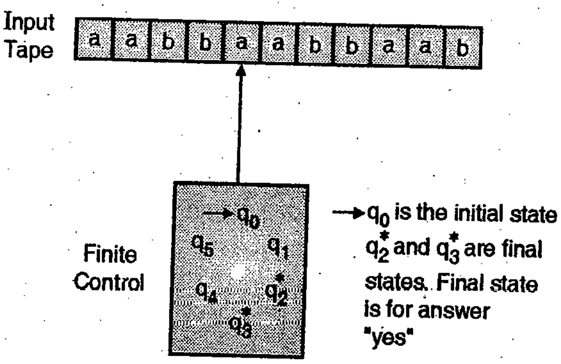


Fig. 2.1.3 : Model of a finite state machine

Operation of the finite automata is given below :

- Input string is fed to the machine through a tape. Tapc 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.

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

  Machine transits from q_i to q_j on input A_i.
- 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 finial states (q_2, q_3) then the answer is 'yes' otherwise the answer is 'no'.

Example 2.1.1 SPPU - May 15, 6 Marks

Explain Basic Machines. What are its limitations ? How is Finite Automata more capable than Basic Machines? Justify with examples.

Solution :

- A machine is defined as system where energy, materials and information are transformed or transmitted. It is used for performing some functions without direct participation of man. Machines are used in automation, examples are automatic packing machines, automatic bottling plant, and automatic photo printing machines.
- It is very difficult to incorporate decision making power in a machine using mechanical control system.
- A finite state automata is mathematical model for actual physical process and it can be viewed as a severely restricted model of a computer. Finite state automata is 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
  5. Control of lift and working machine.
- Thus, a finite state machine has a decision making power and this feature can be used to enhance the power of basic machines.

Example 2.1.2 SPPU - Aug. 15, 2 Marks

What are the limitations of Finite Automata ? Justify with suitable examples.

Solution :

- It has a finite number of states and hence it cannot recognize strings of the form $a^n b^n$. Here, the machine has to remember the number of a's in the string, which will not be possible with finite number of states.

- It cannot be used for computation.
- It cannot modified its own input.
- It cannot be used for context free language.
- It cannot be used for recursive and recursively enumerable languages.

2.1.2 Some Important Terms

2.1.2.1 Alphabet

SPPU - May 14

University Question

Q. Define alphabet formally with example. **(May 2014, 2 Marks)**

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

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

2.1.2.2 Strings (words)

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,

  $$\text{If } \omega = 01101$$
  $$\text{Then } |\omega| = 5$$

  Length of an empty string is zero.

  i.e., $|\varepsilon| = 0$.
- 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,

  $$\text{if } \Sigma = \{0, 1\}$$
  $$\text{then } \Sigma^1 = \{0, 1\}$$
  $$\Sigma^2 = \{00, 01, 10, 11\}$$
  $$\text{And } \Sigma^0 = \{\varepsilon\}$$
- The **reversal** of string ω is denoted by ω^R.

  For example,

  $$\text{If } \omega = xyz$$
  $$\text{Then } \omega^R = zyx.$$
- A string v is a **substring** of a string ω if and only if there are strings x and y such that,

  ω = x v y, where both x and y could be null

  Every string is a substring of itself.

  For example,

  If ω = 'abcdef' is a string.

  then 'cde' is a substring of ω.
- For each string ω, the string ω^i is defined as,

  ω^0 = ε, the empty string

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

  For example,

  $$\text{If } \omega = abb$$
  $$\text{Then } \omega^0 = \varepsilon$$
  $$\omega^1 = abb$$
  $$\omega^2 = \omega \cdot \omega = abbabb$$
  $$\omega^3 = \omega^2 \cdot \omega = abbabbabb$$
- The set of all strings over an alphabet Σ is denoted by Σ^*.

  $$\Sigma^* = \underbrace{\Sigma^0}_{\varepsilon} \cup \underbrace{\Sigma^1}_{\text{String of length 1}} \cup \underbrace{\Sigma^2}_{\text{String of length 2}} \ldots\ldots\ldots\ldots$$

  For example,

  $$\text{if } \Sigma = \{0, 1\}$$
  $$\text{then } \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$$

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,

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

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

$\therefore \quad L_1 = \{\varepsilon, 01, 10, 0011, 0101, 1100, 1010, \ldots\}$

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.

   $\{0, 1, 00, 01, 10, 11, \ldots\}$
2. The set of strings of 0's and 1's with an equal number of each.

   $\{\varepsilon, 01, 10, 0011, 0101, 1010, 1100, \ldots\}$
3. The set of strings of 0's and 1's, ending in 11.

   $\{11, 011, 111, 0011, 0111, 1011, \ldots\}$
4. Σ^* is a language for any alphabet Σ.
5. ϕ, the empty language, is a language over an alphabet.
6. $\{\varepsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.

2.1.2.4 Symbol

A symbol is an abstract entity like letters and digits.

2.1.3 Application of Finite Automata

Finite automata is 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.

Syllabus Topic : Deterministic FA (DFA)

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.

   $F \subseteq Q$ [F 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.

2.2.1 Definition of a DFA

SPPU - Dec. 12, Dec. 13, May 14

University Questions

Q. Define Deterministic Finite Automata with example. **(Dec. 2012, May 2014, 2 Marks)**

Q. Give formal definitions of Deterministic Finite Automata with suitable example. **(Dec. 2013, 2 Marks)**

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.

2.2.2 Representation of a DFA

Let the machine M be a deterministic finite automata.

$$M = \{Q, \Sigma, \delta, q_0, F\}, \text{ where}$$
$$Q = \{q_0, q_1\}$$
$$\Sigma = (0, 1)$$
$$F = \{q_1\}.$$

q_0 is the starting state 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.
→q_0	q_0	q_1 → 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 '→'.
- o Final state is marked as '*'.

2. State transition diagram

The Fig. 2.2.1(a) shows the state transition diagram of the transition function discussed in this section.

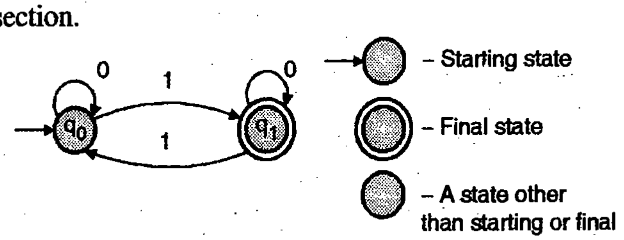


Fig. 2.2.1(a) : State transition diagram

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.

Or,

L (M) = {ω ∈ {0, 1} * | ω 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).

$(q_0, 0010110) \overset{\delta}{\Rightarrow} (q_0, 010110)$ [input 0 in state q_0]

$\overset{\delta}{\Rightarrow} (q_0, 10110)$[input 0 in state q_0]

$\overset{\delta}{\Rightarrow} (q_1, 0110)$[input 1 in state q_0]

$\overset{\delta}{\Rightarrow} (q_1, 110)$[input 0 in state q_1]

$\overset{\delta}{\Rightarrow} (q_0, 10)$[input 1 in state q_1]

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

$\overset{\delta}{\Rightarrow} (q_1, \varepsilon)$[input 0 in state q_1]

Therefore,

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

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 for are 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

- o 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.

- o 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.

   - o Preceding symbol is a. (a is the first symbol in abb).
   - o Preceding two symbols are ab. (ab constitutes first two symbols of abb).
   - o 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.
   - o 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.

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. 2.2.2.

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

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

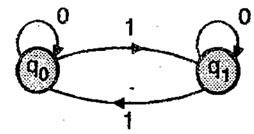
$\delta(q_1,0) \Rightarrow q_1$

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

(a) Tabular form

	0	1
q_0	q_0	q_1
q_1	q_1	q_0

(b) Transition table



(c) Transition diagram

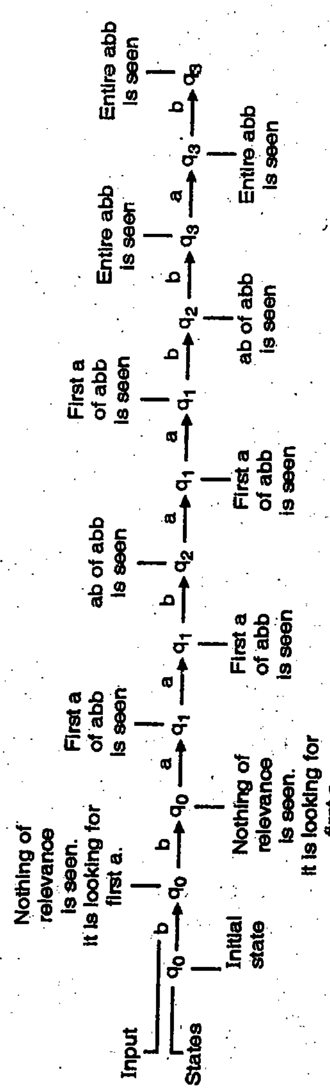
Fig. 2.2.2 : Transition behaviour

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 given below.



Transition behaviour is described using Fig. 2.2.3.

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

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

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

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

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

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

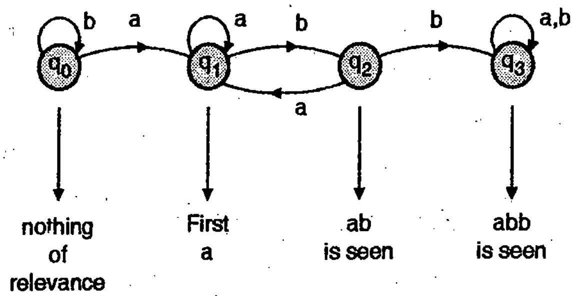
$\delta(q_3,a) \Rightarrow q_3$

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

(a) Tabular form

	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

(b) Transition table



(c) Transition diagram

Fig. 2.2.3 : 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.

Starting state and final states

In Fig. 2.2.2, 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. 2.2.2 is redrawn in Fig. 2.2.4 with initial and final states marked.

	0	1
$\rightarrow q_0^*$	q_0	q_1
q_1	q_1	q_0

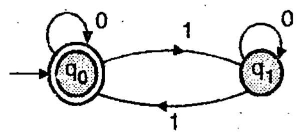


Fig. 2.2.4 : Final DFA for Fig. 2.2.2

In Fig. 2.2.3, 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. 2.2.3 is redrawn in Fig. 2.2.5 with initial and final states marked.

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

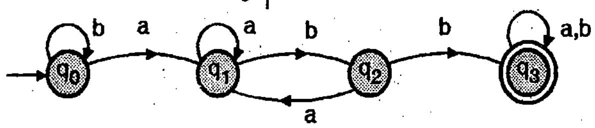


Fig. 2.2.5 : Final DFA for Fig. 2.2.3

2.2.4 Solved Examples on DFA

2.2.4.1 Examples on Counting of Symbols

Example 2.2.1 SPPU - Dec. 16. 6 Marks

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

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

(b) Number of 1's not multiple of 3.

Solution :

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

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

(i) 3n (ii) 3n + 1 (iii) 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

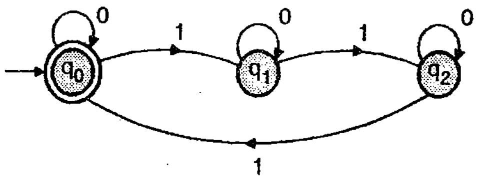
An input of 1 will cause a transition from :

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(A) : Final DFA

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

(b) 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.

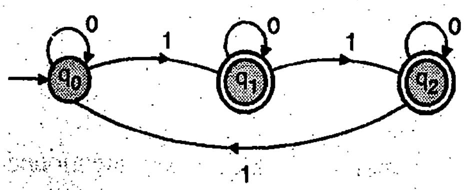
Thus, L = $[\omega \in \{0, 1\}^* \mid$ no. of 1's in ω is multiple of 3]

Complement of L is given by :

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

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

1. Every final state of L should become a non-accepting state in L′.
2. Every non-accepting state of L should become a final state in L′.



(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(B) : Final DFA

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.

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

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

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

- 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.

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

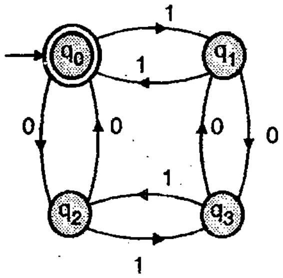
- An input 0 in state q_3, will make number of 0's even.

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

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

$$\delta(q_3, 0) \Rightarrow q_2$$

- 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.



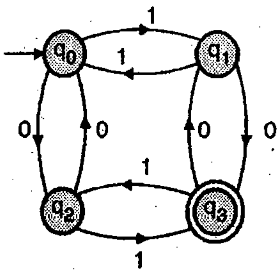
	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

(a) Transition diagram **(b) Transition table**

Fig. Ex. 2.2.2(A) : 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.



	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

(a) Transition diagram **(b) Transition table**

Fig. Ex. 2.2.2(B) : Final DFA for example 2.2.2(b)

Example 2.2.3

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.

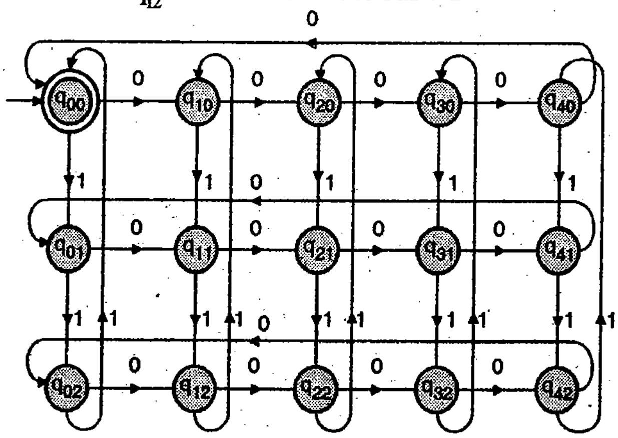
Thus, depending on number of 0's and 1's there will be 5 × 3 = 15 cases.

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

	0	1
$\rightarrow q_{00}^*$	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}

(b) State transition diagram

Fig. Ex. 2.2.3 : Final DFA for Fig. Ex. 2.2.3

Example 2.2.4

Draw DFA for the following language over {0, 1} :

(i) All strings of length at most five

(ii) All strings with exactly two 1's

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

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

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

Solution :

(i) 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.

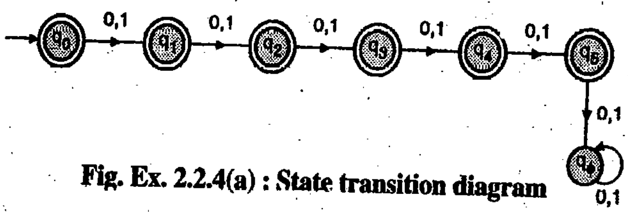


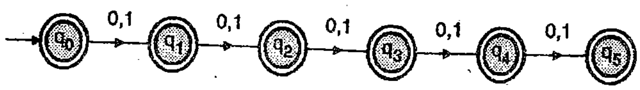
Fig. Ex. 2.2.4(a) : State transition diagram

	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. 2.2.4(A) : Final DFA for example 2.2.4(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 with out explicit failure state.



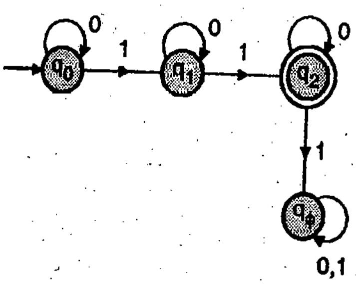
(a) State transition diagram

	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^*	ϕ	ϕ

(b) State transition table

Fig. Ex. 2.2.4(B) : Final DFA for example 2.2.4(i) without a failure state or with failure transition omitted

(ii) All strings with exactly two 1's



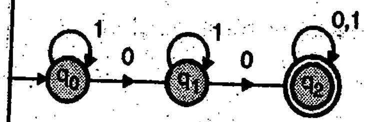
	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_1	q_2
q_2^*	q_2	q_ϕ
q_ϕ	q_ϕ	q_ϕ

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.4(C) : Final DFA for example 2.2.4 (ii)

- First 1 takes the machine from q_0 to q_1.
- Second 1 take 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



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

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.4(D) : Final DFA for example 2.2.4 (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

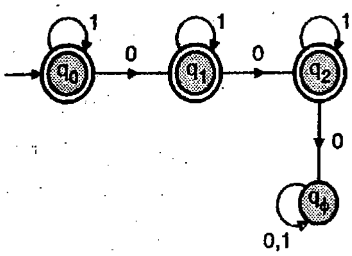
These are the valid strings :

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.

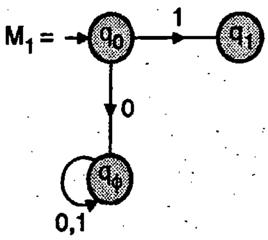


	0	1
$\rightarrow q_0^*$	q_1	q_0
q_1^*	q_2	q_1
q_2^*	q_ϕ	q_2
q_ϕ	q_ϕ	q_ϕ

(a) State transition diagram (b) State transition table
Fig. Ex. 2.2.4(E) : Final DFA for example 2.2.4 (iv)

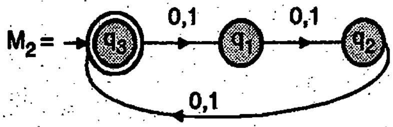
(v) All strings starting with 1 and length of the string is divisible by 3

Step 1 : A machine M_1, representing a DFA accepting a string starting with 1 is given below :



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

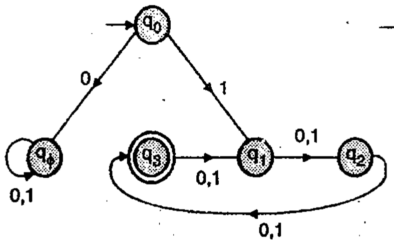
Step 2 : A machine M_2, representing a DFA accepting a strings starting having length divisible by 3 is given below :



(b)

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.



	a	b
$\rightarrow q_0$	q_ϕ	q_1
q_1	q_2	q_2
q_2	q_3	q_3
q_3^*	q_1	q_1
q_ϕ	q_ϕ	q_ϕ

(c) State transition diagram (d) State transition table
Fig. Ex. 2.2.4(F) : Final DFA for example 2.2.4 (v)

2.2.4.2 Examples on Substring

Example 2.2.5

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

(a) All strings starting with abb.

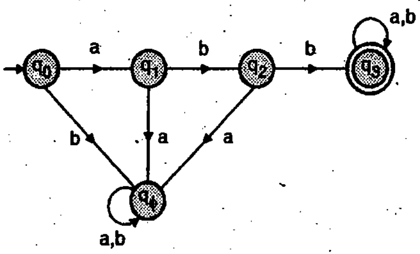
(b) All strings with abb as a substring i.e., abb anywhere in the string.

(c) All strings ending in abb.

Solution :

(a) All strings starting with abb

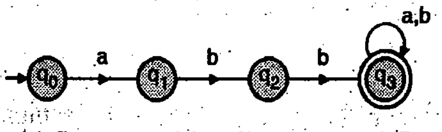
- First input as 'b' will take the machine to a failure state.
- First two inputs as 'aa' will take the machine to a failure state.
- First three inputs as 'aba' will take the machine to a failure state.
- First three inputs as 'abb' will take the machine to a final state.



	a	b
$\rightarrow q_0$	q_1	q_ϕ
q_1	q_ϕ	q_2
q_2	q_ϕ	q_3
q_3^*	q_3	q_3
q_ϕ	q_ϕ	q_ϕ

(a) State transition diagram (b) State transition table
Fig. Ex. 2.2.5(A) : Final DFA for example 2.2.5 (a)

A DFA without explicit failure state is given in Fig. Ex. 2.2.5.



	a	b
$\rightarrow q_0$	q_1	ϕ
q_1	ϕ	q_2
q_2	ϕ	q_3
q_3^*	q_3	q_3

(a) State transition diagram (b) State transition table
Fig. Ex. 2.2.5(B) : Final DFA for example 2.2.5 (a), without a failure / dead state

(b) All strings with abb as a substring

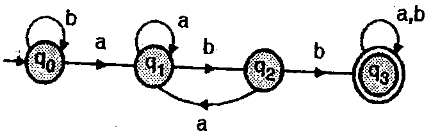
The machine will have fours states :

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.



	a	b
$\to q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3^*	q_3	q_3

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.5(C) : Final DFA for example 2.2.5 (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 :

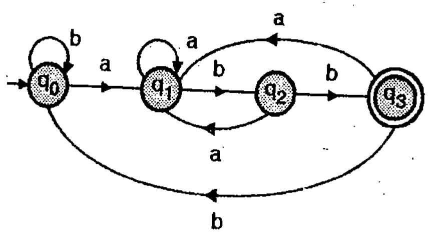
q_3 is for preceding three characters as 'abb'.

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 extention of solution which is given in example 2.2.5 (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'.



	a	b
$\to q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3^*	q_1	q_0

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.5(D) : Final DFA for example 2.2.5 (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 2.2.6

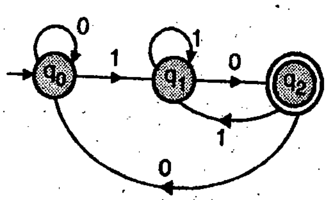
Design DFA for a language of string 0 and 1 that :

I) Ending with 10 II) Ending with 11

III) Ending with 1

Solution :

I) Ending with 10



	0	1
$\to q_0$	q_0	q_1
q_1	q_2	q_1
q_2^*	q_0	q_1

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.6(A) : DFA for string ending with 10

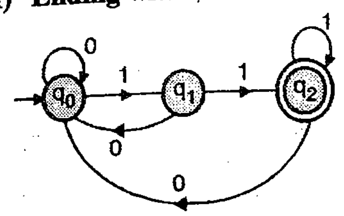
Meaning of various states :

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



(a) State transition diagram

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

(b) State transition table

Fig. Ex. 2.2.6(B) : DFA for string ending with 11

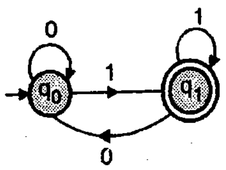
Meaning of various states :

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



(a) State transition diagram

	0	1
$\rightarrow q_0$	q_0	q_1
q_1^*	q_0	q_1

(b) State transition table

Fig. Ex. 2.2.6(C) : DFA for string ending with 1

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

Example 2.2.7

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

Solution :

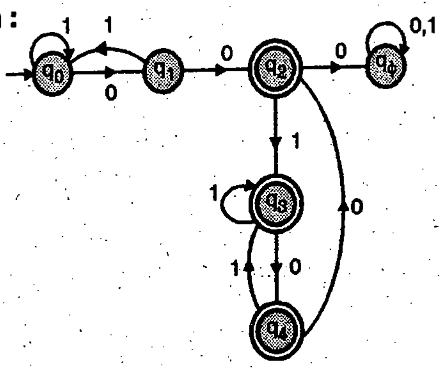


Fig. Ex. 2.2.7 : Final DFA for example 2.2.7

- 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_ϕ.

Example 2.2.8

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.

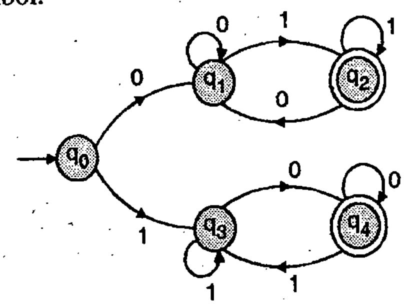
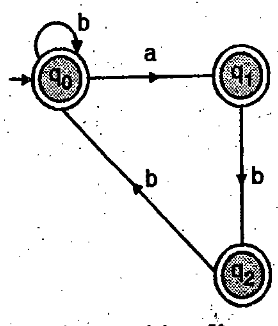


Fig. Ex. 2.2.8 : Final DFA for example 2.2.8

Example 2.2.9

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	b
$\rightarrow q_0^*$	q_1	q_0
q_1^*	ϕ	q_2
q_2^*	ϕ	q_0

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.9 : Final DFA (without explicit failure state) for example 2.2.9

- 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 2.2.10

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

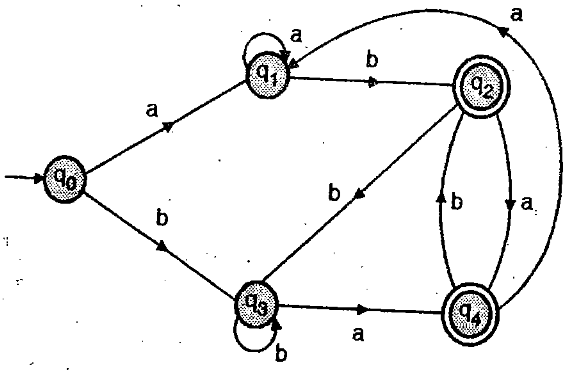


Fig. Ex. 2.2.10 : DFA for example 2.2.10

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 maching 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 2.2.11

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

Solution :

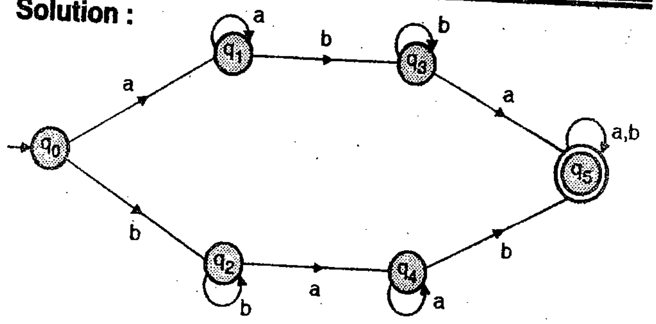


Fig. Ex. 2.2.11 : DFA for Example 2.2.11

- 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 2.2.12

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_ϕ on seeing either aa or bb.
- An input b is 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 input sequence contains a and b alternatively.

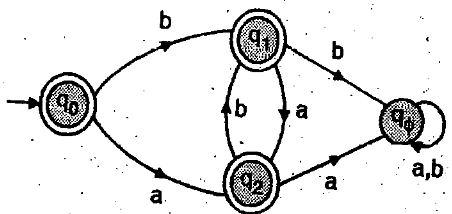
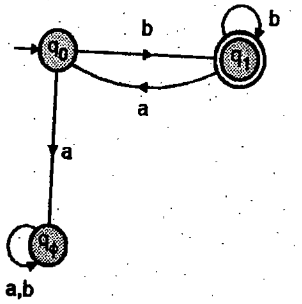


Fig. Ex. 2.2.12 : DFA for Example 2.2.12

Example 2.2.13

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	b
$\rightarrow q_0$	q_ϕ	q_1
q_1^*	q_0	q_1
q_ϕ	q_ϕ	q_ϕ

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.13 : DFA for Example 2.2.13

Example 2.2.14

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.

Solution :

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

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

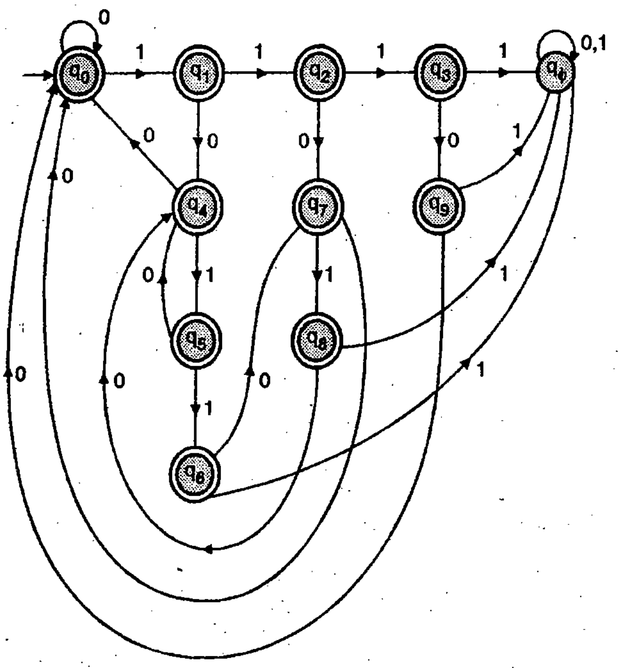


Fig. Ex. 2.2.14 : DFA for example 2.2.14

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.

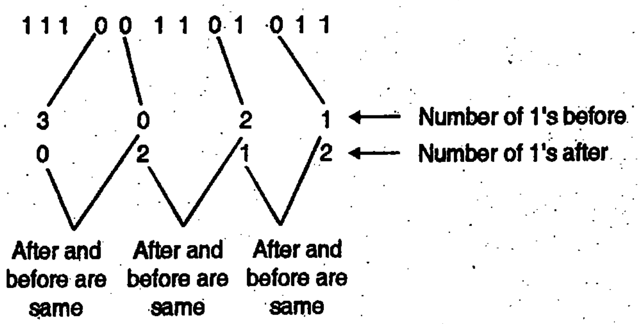
$q_9 \rightarrow$ number of 1's before 0 is three.

$q_5 \rightarrow$ 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.

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.



Example 2.2.15

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

Solution :

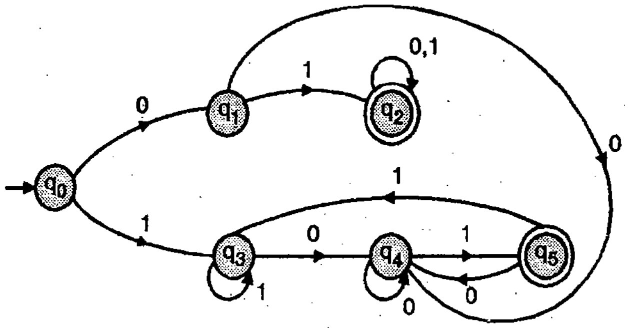


Fig. Ex. 2.2.15 : DFA for example 2.2.15

- 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 2.2.16

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

Solution :

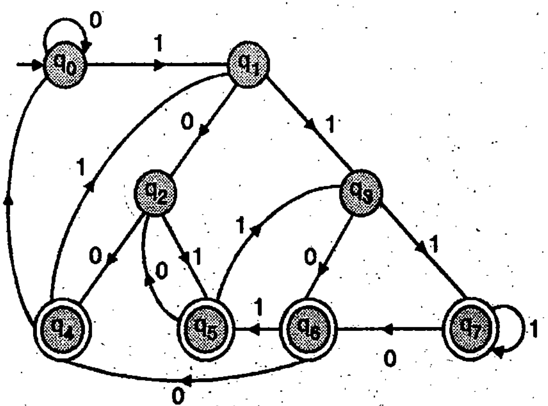


Fig. Ex. 2.2.16 : DFA for example 2.2.16

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, maching 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 2.2.17

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

Solution :

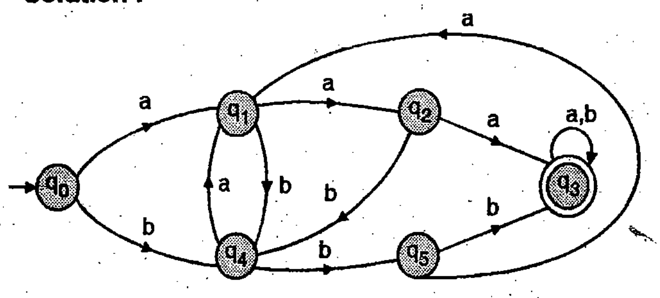


Fig. Ex. 2.2.17 : DFA for example 2.2.17

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 2.2.18

Construct a DFA for accepting a set of strings over alphabet {0, 1} not ending in 010.

Solution :

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-finals state as final state.

Step 1 : DFA for accepting strings ending in 010.

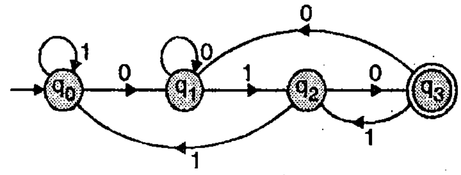


Fig. Ex. 2.2.18(a) : DFA for strings ending in 010

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

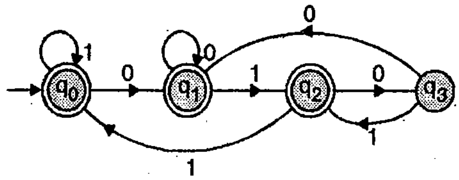
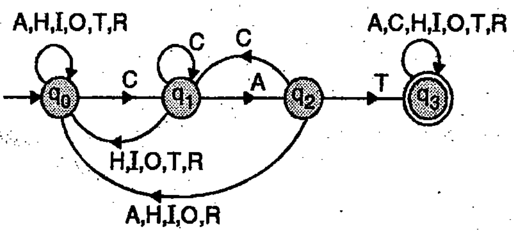


Fig. Ex. 2.2.18(b) : DFA for strings not ending in 010

Example 2.2.19 SPPU - May 13, 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	q_3	q_3	q_3	q_3	q_3	q_3

(b) State transition diagram

Fig. Ex. 2.2.19 : DFA for example 2.2.19

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 2.2.20

Design Finite Automata to accept following formula language specification. Justify your design. $L = \{ (a.b)^n \mid n >= 1\}$

Solution :

- The smallest string accepted by the given DFA is 'ab'. It is accepted through the path $q_0 \to q_1 \to q_2$. It may be noted that the following strings are not accepted :

  $\{\in, a, b, aa, bb, ba\}$

- Let us assume that a string of the form $(ab)^k$ is accepted by the given DFA. We can easily show that the string $(ab)^k$ (ab) will be accepted by the DFA.

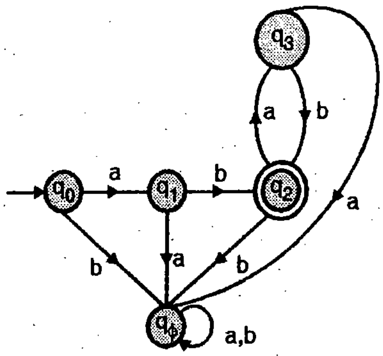


Fig. Ex. 2.2.20

- If the DFA is in the state q_2 after application of the input $(ab)^k$ then the subsequent input of ab will take it back to state q_2 through the path $q_2 \to q_3 \to q_2$.

All other input combinations will be rejected by the DFA.

Example 2.2.21 SPPU - Dec. 14, 6 Marks

Design FA that read strings made up of letters in the word "CHARIOT" and recognize those strings that contain the word "RAT" as a substring.

Solution :

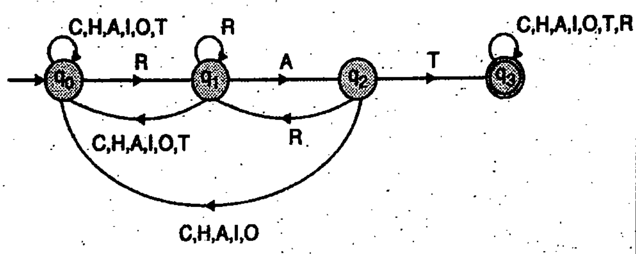


Fig. Ex. 2.2.21

2.2.4.3 Examples of Divisibility

Example 2.2.22

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 device 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 :

  $0 \to$ associated state, q_0

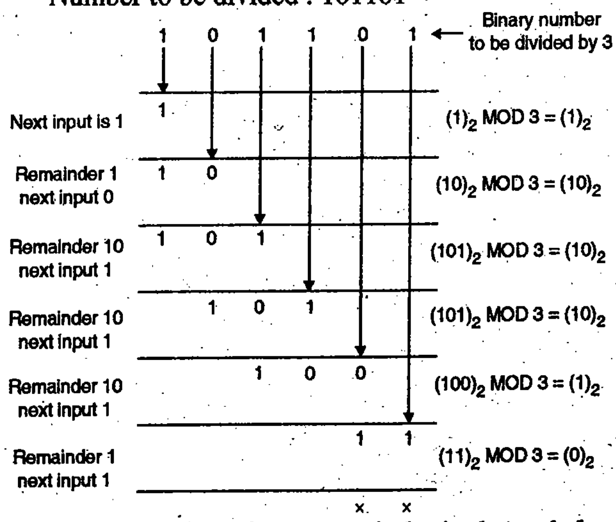
  $1 \to$ associated state, q_1

  $2 \to$ 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.

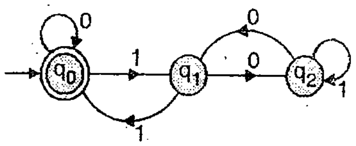
The process of finding the running remainder is being explained with the help of an example.

Number to be divided : 101101



The calculation of next remainder is shown below, in a tabular form.

Previous remainder	Next input	Calculation of remainder		Next remainder
0 (q_0)	0	00 % 3	⇒	0 (q_0)
0 (q_0)	1	01 % 3	⇒	1 (q_1)
1 (q_1)	0	10 % 3	⇒	10 (q_2)
1 (q_1)	1	11 % 3	⇒	0 (q_0)
10 (q_2)	0	100 % 3	⇒	1 (q_1)
10 (q_2)	1	101 % 3	⇒	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

(a) State transition diagram (b) State transition table

Fig. Ex. 2.2.22 : DFA for example 2.2.22

Example 2.2.23

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

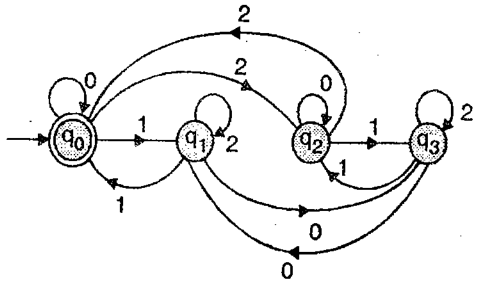
$(10)_3 = 3 \rightarrow$ associated state, q_3

↑ 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	0	00 $\Rightarrow$ 0 MOD 4 $\Rightarrow$	0 (q_0)
	1	01 $\Rightarrow$ 1 MOD 4 $\Rightarrow$	1 (q_1)
	2	02 $\Rightarrow$ 2 MOD 4 $\Rightarrow$	2 (q_2)
(q_1) 1	0	10 $\Rightarrow$ 3 MOD 4 $\Rightarrow$	3 (q_3)
	1	11 $\Rightarrow$ 4 MOD 4 $\Rightarrow$	0 (q_0)
	2	12 $\Rightarrow$ 5 MOD 4 $\Rightarrow$	1 (q_1)
(q_2) 2	0	20 $\Rightarrow$ 6 MOD 4 $\Rightarrow$	2 (q_2)
	1	21 $\Rightarrow$ 7 MOD 4 $\Rightarrow$	3 (q_3)
	2	22 $\Rightarrow$ 8 MOD 4 $\Rightarrow$	0 (q_0)
(q_3) 10	0	100 $\Rightarrow$ 9 MOD 4 $\Rightarrow$	1 (q_1)
	1	101 $\Rightarrow$ 10 MOD 4 $\Rightarrow$	2 (q_2)
	2	102 $\Rightarrow$ 11 MOD 4 $\Rightarrow$	3 (q_3)
↑ Ternary		↑ 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. 2.2.23 : DFA for example 2.2.23

Example 2.2.24

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

$0 \rightarrow$ associated state, q_0

$1 \rightarrow$ associated state, q_1

$2 \rightarrow$ associated state, q_2

Transition behaviour will depend on the current remainder (current rate) 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 :

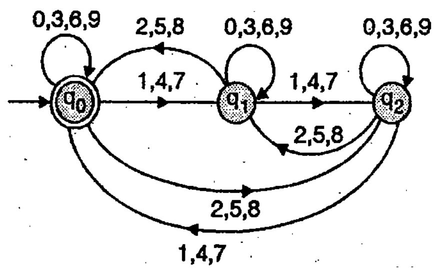
0 for (0, 3, 6, 9)

1 for (1, 4, 7)

2 for (2, 5, 8)

Transition behaviour is explained below :

Current remainder	Next input	Calculation of remainder		Remainder
0	0	$00 \Rightarrow 0$ MOD 3	$\Rightarrow$	0 (q_0)
	1	$01 \Rightarrow 1$ MOD 3	$\Rightarrow$	1 (q_1)
	2	$02 \Rightarrow 2$ MOD 3	$\Rightarrow$	2 (q_2)
1	0	$10 \Rightarrow 10$ MOD 3	$\Rightarrow$	1 (q_1)
	1	$11 \Rightarrow 11$ MOD 3	$\Rightarrow$	2 (q_2)
	2	$12 \Rightarrow 12$ MOD 3	$\Rightarrow$	0 (q_0)
2	0	$20 \Rightarrow 20$ MOD 3	$\Rightarrow$	2 (q_2)
	1	$21 \Rightarrow 21$ MOD 3	$\Rightarrow$	0 (q_0)
	2	$22 \Rightarrow 22$ 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. 2.2.24 : DFA for example 2.2.24

Example 2.2.25 SPPU - Dec. 2014, 6 Marks

Design FSM over {a, b) accepting the string where number of a' s are divisible by 2 and number of b's are divisible by 3.

Solution :

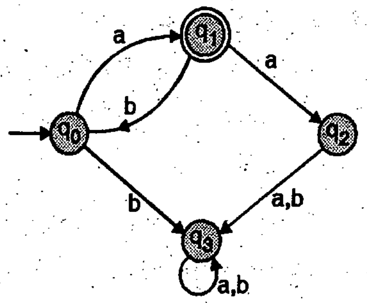


Fig. Ex. 2.2.25

2.2.5 Language of DFA

The language of a DFA M = {Q, Σ, δ, q_0, F} is denoted by L (M) and is defined by :

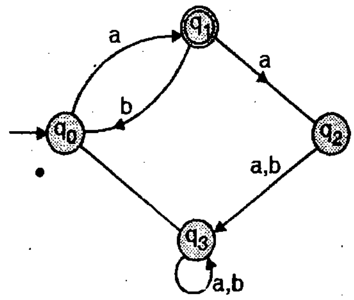
L (M) = {ω | δ*(q_0, ω) is in F}

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. δ*(q_0, ω) stands for a series of transitions starting from q_0.

Example 2.2.26

Describe the language accepted by the deterministic finite automata shown below

(i)



(ii)

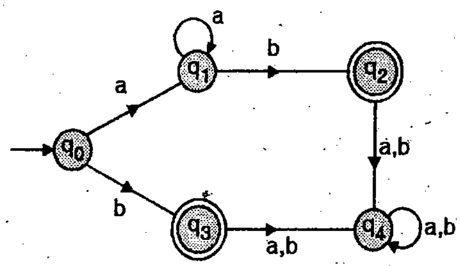


Fig. Ex. 2.2.26

Solution :

(i)

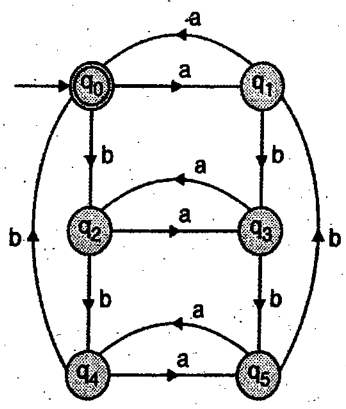


Fig. Ex. 2.2.26(a)

- 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.
- The DFA can be redrawn after elimination of q_2 and q_3.

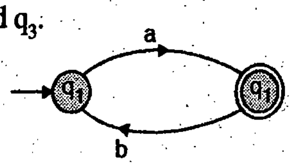


Fig. Ex. 2.2.26(b)

- The language of the DFA can be defined as given below :

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

(ii)

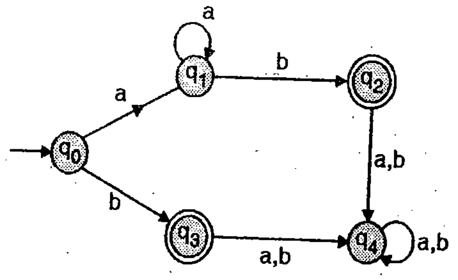


Fig. Ex. 2.2.26(c)

- q_4 is dead state. The DFA can be redrawn after elimination of q_4.
- A string 'b' of length 1 is accepted through the path $q_0 \rightarrow q_3$.
- A string having 1 or more a's followed by a 'b' is accepted through the path.

$q_0 \rightarrow q_1 \rightarrow q_1 \ldots q_1 \rightarrow q_2$

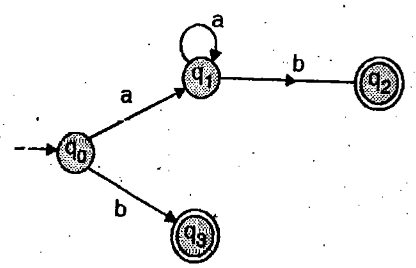


Fig. Ex. 2.2.26(d)

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

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

Example 2.2.27

Consider the following transition diagram.

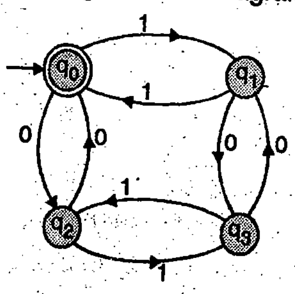


Fig. Ex. 2.2.27

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

Solution :

$$(q_0, 110101) \overset{\delta}{\Rightarrow} (q_1, 10101)$$
$$\overset{\delta}{\Rightarrow} (q_0, 0101)$$
$$\overset{\delta}{\Rightarrow} (q_2, 101)$$
$$\overset{\delta}{\Rightarrow} (q_3, 01)$$
$$\overset{\delta}{\Rightarrow} (q_1, 1)$$
$$\overset{\delta}{\Rightarrow} (q_0, \varepsilon) \Rightarrow q_0$$

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

2.3 Equivalence of DFAs

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,

$s_i \in S$ of M_1 and $q_i \in Q$ 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 $\boldsymbol{\delta_3}$ of M_3 is given below :

1. Add the state $< s_i, q_0 >$ to M_3.
2. For every state $< s_i, q_i >$ in M_3.

{

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

{

for each alphabet $a_i \in \Sigma$

{

add a transition $< \delta_1 (s_i, a_i), \delta_2 (q_i, a_i)$ to M_3

}

}

}

M_1 and M_2 are **equivalent** if :

for every state $< s_i, q_i >$ 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 $< s_i, q_i >$ in M_3

{ s_i is a final states and q_i is a non-final state

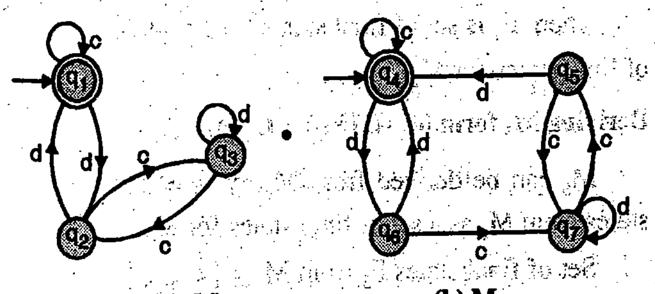
or

s_i is a non-final state and q_i is a final state

}

Example 2.3.1

Show whether the automata M_1 and M_2 in given Fig. Ex. 2.3.1(a) and (b) are equivalent or not.



(a) M_1 (b) M_2

Fig. Ex. 2.3.1

Solution :

Construction of combined DFA M_3

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

$\delta_3 (< q_1, q_4 >, c) \Rightarrow < \delta_1 (q_1, c), \delta_2 (q_4, c) >$

$\Rightarrow < q_1, q_4 >$

$\delta_3 (< q_1, q_4 >, d) \Rightarrow < \delta_1 (q_1, d), \delta_2 (q_4, d) >$

$\Rightarrow < q_2, q_6 >$

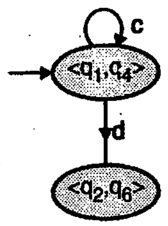


Fig. Ex. 2.3.1(c)

Step 2 : State $< q_2, q_6 >$ is selected for expansion

$\delta_3 (< q_2, q_6 >, c) \Rightarrow < \delta_1 (q_2, c), \delta_2 (q_6, c) >$

$\Rightarrow < q_3, q_7 >$

$\delta_3 (< q_2, q_6 >, d) \Rightarrow < \delta (q_2, d), \delta_2 (q_6, d) >$

$\Rightarrow < q_1, q_4 >$

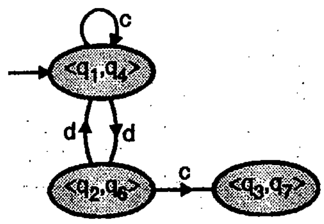


Fig. Ex. 2.3.1(d)

Step 3 : State $< q_3, q_7 >$ is selected for expansion.

$\delta_3 (< q_3, q_7 >, c) \Rightarrow < \delta (q_3, c), \delta_2 (q_7, c) >$

$\Rightarrow < q_2, q_5 >$

$\delta_3 (< q_3, q_7 >, d) \Rightarrow < \delta (q_3, d), \delta_2 (q_7, d) >$

$\Rightarrow < q_3, q_7 >$

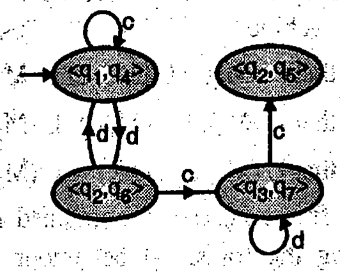


Fig. Ex. 2.3.1(e) : M_3

Step 4 : State $< q_2, q_5 >$ is selected for expansion.

$$\delta_3 (< q_2, q_5 >, c) \Rightarrow < \delta_1 (q_2, c), \delta_2 (q_5, c) >$$
$$\Rightarrow < q_3, q_7 >$$
$$\delta_3 (< q_2, q_5 >, d) \Rightarrow < \delta_1 (q_2, d), \delta_2 (q_5, d) >$$
$$\Rightarrow < q_1, q_4 >$$

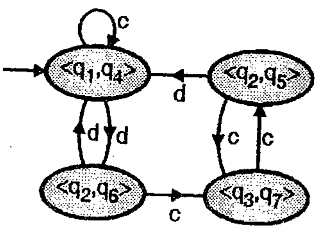


Fig. Ex. 2.3.1(f) : M_3

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.

$< q_1, q_4 >$: both q_1 and q_4 are final states.

$< q_2, q_5 >$: q_2 and q_5 are non final states.

$< q_2, q_6 >$: both are non final states.

$< q_3, q_7 >$: q_3 and q_7 are non final states.

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

2.4 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.

2.4.1 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 $< s_i, q_i >$ where, $s_i \in S$ of M_1 and $q_i \in Q$ of M_2.

The transition function δ_c is defined as :

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

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 $< s_i, q_i >$ in M_c.

```
{
    if (< si, qi > has not been expanded)
    {
        for each alphabet ai ∈ Σ
        {
        add a transition < δ1 (si, ai), δ2(qi, ai) > to Mc.
        }
    }
}
```

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 $M_3 = \{< s_i, q_i > \in$ states of $M_c \mid s_i \in F_i \vee q_i \in F_2\}$

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 form 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 M_c as a set of final states for M_4.

Set of final states F_4 for $M_4 = \{< s_i, q_i > \in$ 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 form 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 $M_5 = \{< s_i, q_i > \in$ 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 2.4.1

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) $L(M_2) \cap L(M_2)$

(iii) $L(M_1) - L(M_2)$

Solution :

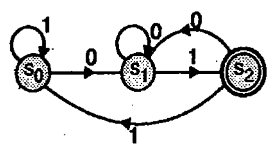
M_1 = 

Fig. Ex. 2.4.1(a) : DFA for strings ending in 01

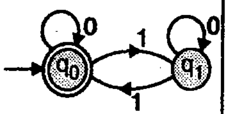
M_2 = 

Fig. Ex. 2.4.1(b) : DFA for strings having even number of 1's

Construction of combined DFA M_3

Step 1 : We start with the state $< s_0, q_0 >$, where s_0 is the initial state of M_1 and q_0 is the initial state of M_2.

$< s_0, q_0 >$ is expanded and necessary transitions added.

$\delta(< s_0, q_0 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_0, 0) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_0, q_0 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_0, 1) >$

$\Rightarrow < s_0, q_1 >$

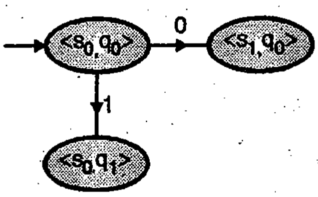


Fig. Ex. 2.4.1(c)

Step 2 : State $< s_1, q_0 >$ and $< s_0, q_1 >$ are selected for expansion.

$\delta(< s_0, q_1 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_1, 0) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_0, q_1 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_1, 1) >$

$\Rightarrow < s_0, q_0 >$

$\delta(< s_1, q_0 >, 0) \Rightarrow < \delta(s_1, 0), \delta(q_0, 0) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_1, q_0 >, 1) \Rightarrow < \delta(s_1, 1), \delta(q_0, d) >$

$\Rightarrow < s_2, q_1 >$

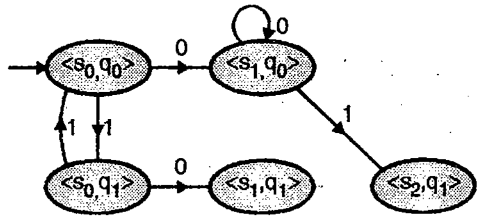


Fig. Ex. 2.4.1(d)

Step 3 : State $< s_1, q_1 >$ and $< s_2, q_1 >$ are selected for expansion.

$\delta(< s_1, q_1 >, 0) \Rightarrow < \delta(s_1\ 0), \delta(q_1, 0) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_i, q_1 >, 1) \Rightarrow < \delta(s_i, 1), \delta(q_1, 1) >$

$\Rightarrow < s_2, q_0 >$

$\delta(< s_2, q_1 >, 0) \Rightarrow < \delta(s_2, 0), \delta(q_1, 1) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_2, q_1 >, 1) \Rightarrow < \delta(s_2, 1), \delta(q_1, 1) >$

$\Rightarrow < s_0, q_0 >$

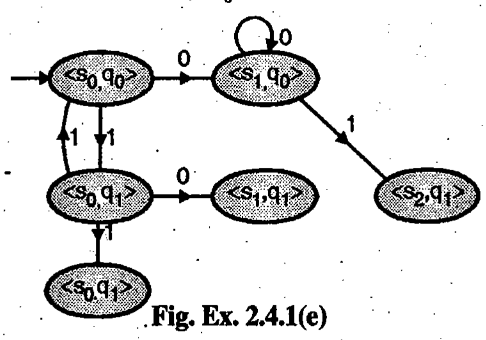


Fig. Ex. 2.4.1(e)

Step 4 : State $< s_2, q_0 >$ is selected for expansion.

$\delta(< s_2, q_0 >, 0) \Rightarrow < \delta(s_2, 0), \delta(q_0, 1) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_2, q_0 >, 1) \Rightarrow < \delta(s_2, 1), \delta(q_0, 1) >$

$\Rightarrow < s_0, q_1 >$

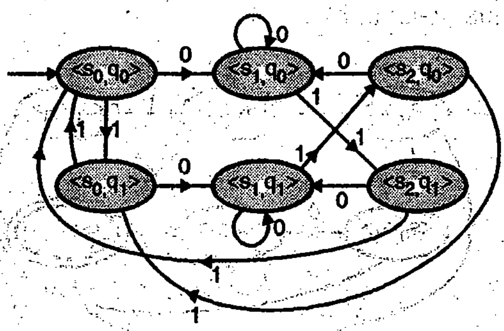


Fig. Ex. 2.4.1(f)

Step 4 : State $< q_2, q_5 >$ is selected for expansion.

$\delta_3 (< q_2, q_5 >, c) \Rightarrow < \delta_1 (q_2, c), \delta_2 (q_5, c) >$

$\Rightarrow < q_3, q_7 >$

$\delta_3 (< q_2, q_5 >, d) \Rightarrow < \delta_1 (q_2, d), \delta_2 (q_5, d) >$

$\Rightarrow < q_1, q_4 >$

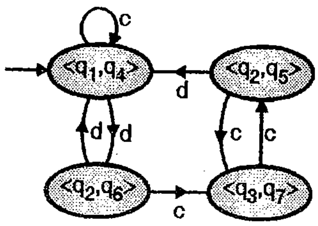


Fig. Ex. 2.3.1(f) : M_3

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.

$< q_1, q_4 >$: both q_1 and q_4 are final states.

$< q_2, q_5 >$: q_2 and q_5 are non final states.

$< q_2, q_6 >$: both are non final states.

$< q_3, q_7 >$: q_3 and q_7 are non final states.

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

2.4 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.

2.4.1 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 $< s_i, q_i >$ where,

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

The transition function δ_c is defined as :

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

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 $< s_i, q_i >$ in M_c.

```
{
    if (< si, qi > has not been expanded)
    {
        for each alphabet ai ∈ Σ
        {
            add a transition < δ1 (si, ai), δ2(qi, ai) > to Mc.
        }
    }
}
```

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 $M_3 = \{< s_i, q_i > \in$ states of $M_c \mid s_i \in F_i \vee q_i \in F_2\}$

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 form 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 M_c as a set of final states for M_4.

Set of final states F_4 for $M_4 = \{< s_i, q_i > \in$ 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 form 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 $M_5 = \{< s_i, q_i > \in$ 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 2.4.1

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) $L(M_2) \cap L(M_2)$

(iii) $L(M_1) - L(M_2)$

Solution :

M_1 =

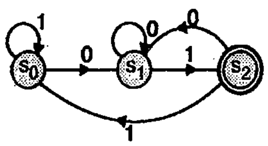


Fig. Ex. 2.4.1(a) : DFA for strings ending in 01

M_2 =

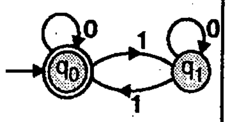


Fig. Ex. 2.4.1(b) : DFA for strings having even number of 1's

Construction of combined DFA M_3

Step 1 : We start with the state $< s_0, q_0 >$, where s_0 is the initial state of M_1 and q_0 is the initial state of M_2.

$< s_0, q_0 >$ is expanded and necessary transitions added.

$\delta(< s_0, q_0 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_0, 0) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_0, q_0 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_0, 1) >$

$\Rightarrow < s_0, q_1 >$

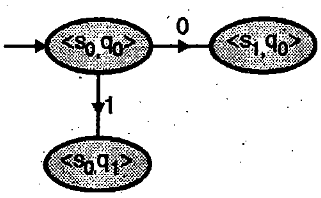


Fig. Ex. 2.4.1(c)

Step 2 : State $< s_1, q_0 >$ and $< s_0, q_1 >$ are selected for expansion.

$\delta(< s_0, q_1 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_1, 0) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_0, q_1 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_1, 1) >$

$\Rightarrow < s_0, q_0 >$

$\delta(< s_1, q_0 >, 0) \Rightarrow < \delta(s_1, 0), \delta(q_0, 0) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_1, q_0 >, 1) \Rightarrow < \delta(s_1, 1), \delta(q_0, d) >$

$\Rightarrow < s_2, q_1 >$

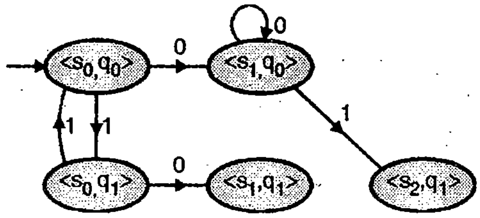


Fig. Ex. 2.4.1(d)

Step 3 : State $< s_1, q_1 >$ and $< s_2, q_1 >$ are selected for expansion.

$\delta(< s_1, q_1 >, 0) \Rightarrow < \delta(s_1\ 0), \delta(q_1\ 0) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_i, q_1 >, 1) \Rightarrow < \delta(s_i, 1), \delta(q_1, 1) >$

$\Rightarrow < s_2, q_0 >$

$\delta(< s_2, q_1 >, 0) \Rightarrow < \delta(s_2, 0), \delta(q_1, 1) >$

$\Rightarrow < s_1, q_1 >$

$\delta(< s_2, q_1 >, 1) \Rightarrow < \delta(s_2, 1), \delta(q_1, 1) >$

$\Rightarrow < s_0, q_0 >$

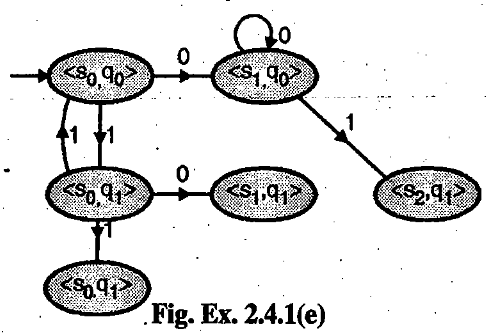


Fig. Ex. 2.4.1(e)

Step 4 : State $< s_2, q_0 >$ is selected for expansion.

$\delta(< s_2, q_0 >, 0) \Rightarrow < \delta(s_2, 0), \delta(q_0, 1) >$

$\Rightarrow < s_1, q_0 >$

$\delta(< s_2, q_0 >, 1) \Rightarrow < \delta(s_2, 1), \delta(q_0, 1) >$

$\Rightarrow < s_0, q_1 >$

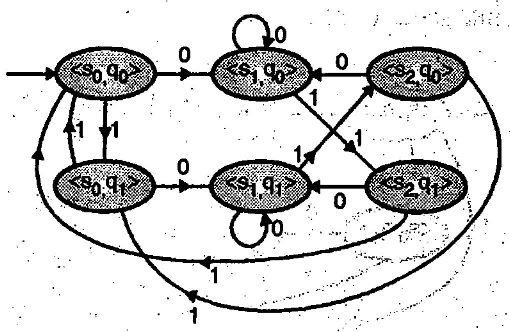


Fig. Ex. 2.4.1(f)

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

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

Every $< s_i, q_i >$ such that

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

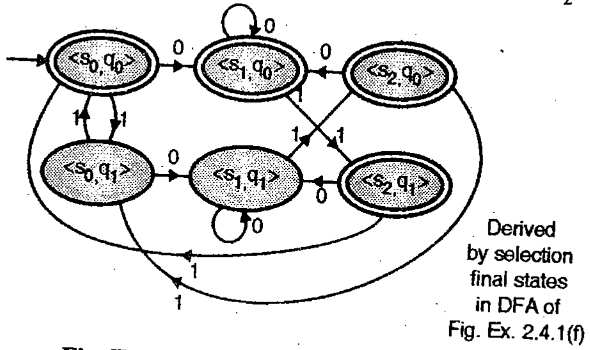


Fig. Ex. 2.4.1(g) : DFA for L $(M_1) \cup$ L (M_2)

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

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

Every $< s_i, q_i >$ such that $s_i \in$ final states of M_1 **and** $q_i \in$ final states of M_2.

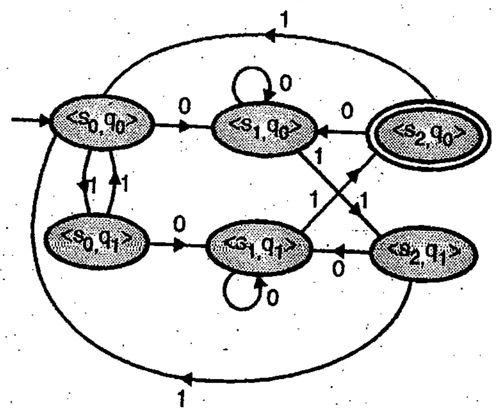


Fig. Ex. 2.4.1(h) : DFA for L $(M_1) \cap$ L (M_2)

Finding DFA for L (M_1) – L (M_2)

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

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

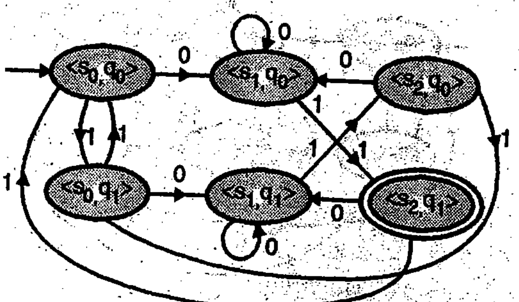


Fig. Ex. 2.4.1(i) : DFA for L (M_1) – L (M_2)

2.4.2 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 :

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

2.5 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.

2.5.1 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.

States of the DFA in Fig. 2.5.1 can be divided into two blocks.

block 1 = set of non-final states

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

block 2 = set of final states.

$= (q_2)$

$\therefore$ 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'

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

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

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

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

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

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

$\delta(q_7, 0) \Rightarrow q_6$ {block 1, q_6 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'

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

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

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

$\delta(q_4, 1) \Rightarrow q_5$ {block 1}

$\delta(q_5, 1) \Rightarrow q_6$ {block 1}

$\delta(q_6, 1) \Rightarrow q_4$ {block 1}

$\delta(q_7, 1) \Rightarrow q_2$ {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 or 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,

$$P_1 = \underset{\text{block 11}}{\underset{\uparrow}{(q_3, q_5)}}\quad \underset{\text{block 12}}{\underset{\uparrow}{(q_1, q_7)}}\quad \underset{\text{block 13}}{\underset{\uparrow}{(q_0, q_4, q_6)}}\quad \underset{\text{block 2}}{\underset{\uparrow}{(q_2)}}$$

- 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

$$\left.\begin{array}{l}\delta(q_3, 0) = q_2\\ \delta(q_5, 0) = q_2\end{array}\right\}\text{mapped to same block } (q_2)$$

$$\left.\begin{array}{l}\delta(q_3, 1) = q_6\\ \delta(q_5, 1) = q_6\end{array}\right\}\text{mapped to same block } (q_0, q_4, q_6)$$

q_3 and q_5 are 2-equivalent.

Checking (q_1, q_7) for equivalence – block 12

$$\left.\begin{array}{l}\delta(q_1, 0) = q_6\\ \delta(q_7, 0) = q_6\end{array}\right\}\text{mapped to same block } (q_0, q_4, q_6)$$

$$\left.\begin{array}{l}\delta(q_1, 1) = q_2\\ \delta(q_7, 1) = q_2\end{array}\right\}\text{mapped to same block } (q_2)$$

q_1 and q_7 are 2-equivalent.

Checking (q_0, q_4, q_6) for equivalence – block 13

$\delta(q_0, 0) = q_1$ [block 12]

$\delta(q_4, 0) = q_7$ [block 12]

$\delta(q_6, 0) = q_6$ [block 13]

$\delta(q_0, 1) = q_5$ [block 11]

$\delta(q_4, 1) = q_5$ [block 11]

$\delta(q_6, 1) = q_4$ [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. 2.5.1 is given in Fig. 2.5.2. The process of drawing of minimum-state DFA is as below :

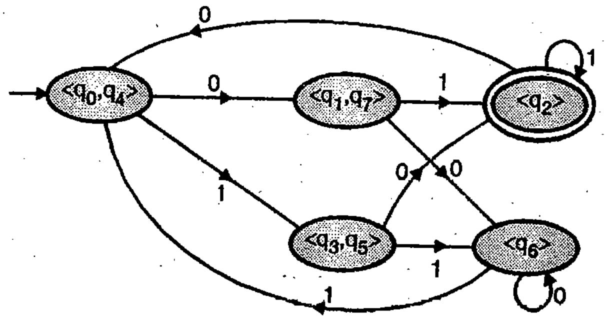


Fig. 2.5.2 : Minimum state DFA equivalent to Fig. 2.5.1

1. Start state is <q_0, q_4>, since q_0 was start state in Fig. 2.5.1.
2. Only accepting state is <q_2 >, since q_2 is the only accepting state of Fig. 2.5.1.
3. Since, there is a transition from q_0 to q_1 on input 0 in Fig. 2.5.1, we add a transition from <q_0, q_4> to <q_1, q_7> on input 0 in Fig. 2.5.1.
4. Since, there is a transition from q_0 to q_5 on input 1 in Fig. 2.5.1, we add a transition from <q_0, q_4> to <q_3, q_5> on input 1 in Fig. 2.5.2.
5. Other transitions can be added in a similar fashion.

Example 2.5.1

Construct the minimum state automata equivalent to given DFA.

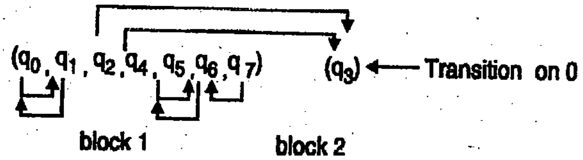
	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3^*	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

Solution :

Step 1 : Finding 0-equivalence partitioning of states by putting final and non-final states into independent block.

$P_0 = \underbrace{(q_0, q_1, q_2, q_4, q_5, q_6, q_7)}_{\text{block 1}}\quad \underbrace{(q_3)}_{\text{block 2}}$

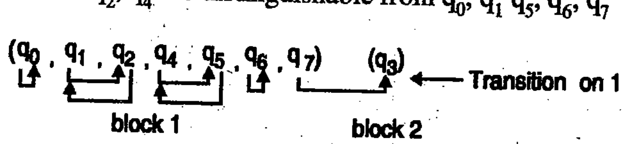
Step 2 : Finding 1-equivalence partitioning of states by considering transition on '0' and transition on '1'.



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



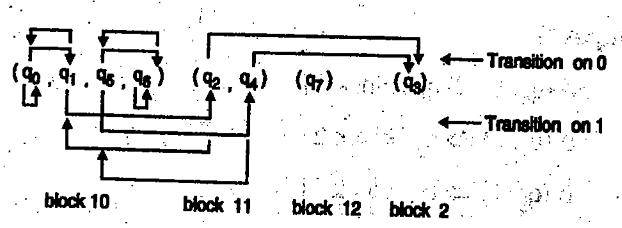
On input 1, block 2 is successor of q_7.

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'.



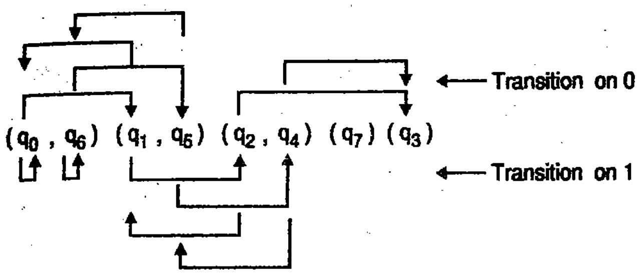
On input 1, block 11 is successor of q_1, q_5.

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.



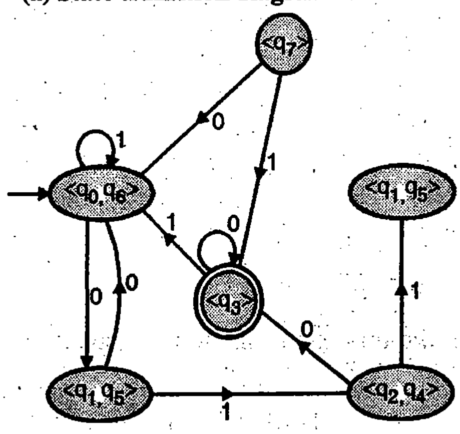
Blocks can not be divided further.

$\therefore P_3 = P_2 = (q_0, q_6)\ (q_1, q_5)\ (q_2, q_4)\ (q_7)\ (q_3)$ 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-



(b) State transition diagram for minimum-state DFA state DFA

Fig. Ex. 2.5.1

Example 2.5.2

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

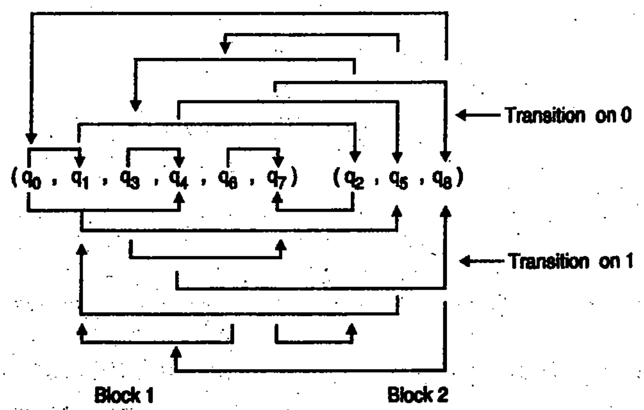
Solution :

Step 1 : Finding 0-equivalence partitioning of states by putting final and non-final states into independent blocks.

$$P_0 = \underbrace{(q_0, q_1, q_3, q_4, q_6, q_7)}_{\text{block 1}}\ \underbrace{(q_2, q_5, q_8)}_{\text{block 2}}$$

Step 2 : Finding 1-equivalence partitioning of states by considering transitions on '0' on '1'.

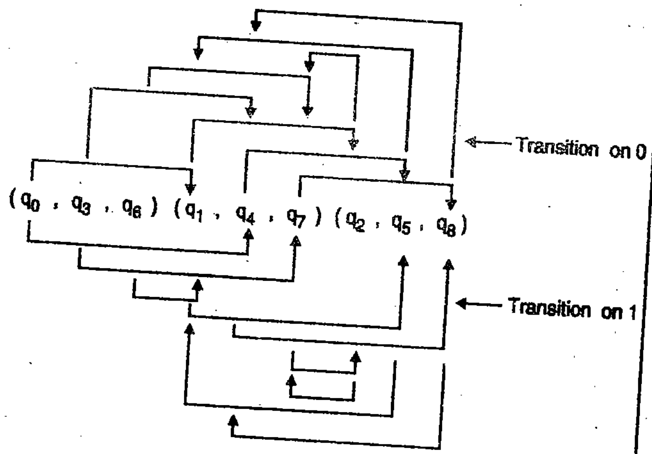
$$\left.\begin{array}{l}
\text{On input 0, block 1 is successor of } q_0, q_3, q_6. \\
\text{On input 0, block 2 is successor of } q_1, q_4, q_7. \\
\text{On Input 1, block 1 is successor of } q_0, q_3, q_6. \\
\text{On input 1, block 2 is successor of } q_1, q_4, q_7. \\
\text{States } q_0, q_3, q_6 \text{ are distinguishable from } q_1, q_4, q_7.
\end{array}\right\} \text{block 1}$$



Block 2 is 1-equivalent

$$P_1 = \underbrace{(q_0, q_3, q_6)}_{\text{block 11}}\ \underbrace{(q_1, q_4, q_7)}_{\text{block 12}}\ \underbrace{(q_2, q_5, q_8)}_{\text{block 2}}$$

Step 3 : Finding 2-equivalence partitioning of states by considering transition on '0' and transition on '1'.



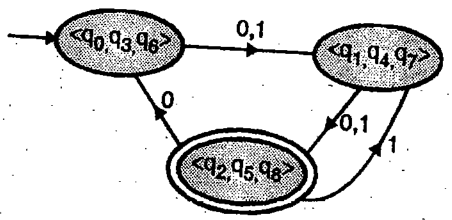
Blocks can not be divided further.

$\therefore$ $P_2 = P_1 = (q_0, q_3, q_6)\ (q_1, q_4, q_7)\ (q_2, q_5, q_8)$ 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. 2.5.2 : Final DFA for example 2.5.2

Syllabus Topic : Non-deterministic FA (NFA)

2.6 Non-deterministic Finite Automata

SPPU - Dec. 12. Dec. 13. May 14

University Questions

Q. Define Non-deterministic finite automaton with example. **(Dec. 2012, May 2014, 2 Marks)**

Q. Give formal definitions of Non-Deterministic Finite Automata with suitable examples. **(Dec. 2013, 2 Marks)**

A Non-deterministic finite automata can reside in multiple states at the same time. The concept of nondeterministic finite automata is being explained with the help of transition diagram (Fig. 2.6.1).

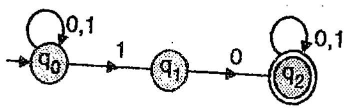


Fig. 2.6.1 : Transition diagram of a nondeterministic finite automata

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.

- A 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.

2.6.1 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. 2.6.2, 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 below in Fig. 2.6.2.

	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2,\}$	ϕ
q_2^*	$\{q_2\}$	$\{q_2\}$

Fig. 2.6.2 : A transition table for NFA in Fig. 2.6.1

2.6.2 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 below (Fig. 2.6.3).

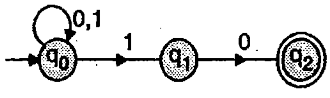


Fig. 2.6.3 : A sample NFA

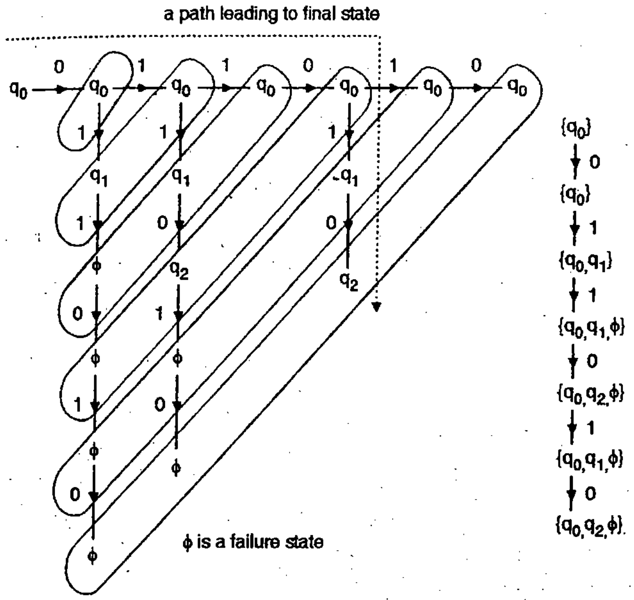


Fig. 2.6.4 : States reached while processing of 011010

- 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 a non-deterministic finite automata, if there exists a path that takes the machine to a final state.

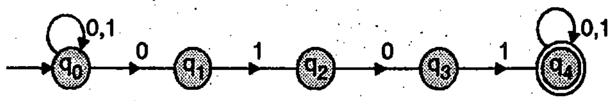
  Thus, 011010 will be accepted by the NFA of Fig. 2.6.3.
- The language accepted by the above NFA consists of string ending in 10. A string of length n ($|\omega| = n$), ending 10 can be accepted by the above NFA if :
  1. The machine remains in q_0 for first n-2 inputs.
  2. On last two inputs 10, the 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 (Fig. 2.6.3) accepts a string, if and only if it ends in 10.

Example 2.6.1

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	ϕ	$\{q_2\}$
q_2	$\{q_3\}$	ϕ
q_1	ϕ	$\{q_4\}$
q_4^*	$\{q_4\}$	$\{q_4\}$

(b) State transition table

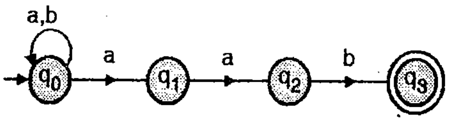
Fig. Ex. 2.6.1

- 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 2.6.2

Construct a NFA that accepts any positive number of occurrences of various strings from the following language L given by, L = {x ∈ {a,b} | x ends with aab}

Solution :



(a) State transition diagram

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	q_2	–
q_2	–	q_3
q_3	–	–

(b) State transition table

Fig. Ex. 2.6.2

- The language accepted by the above NFA consist of string ending in aab. A string of length n, ending in aab can be accepted by the above NFA if :
  1. The machine remains in q0 for first n-3 inputs.
  2. On last three inputs aab, the machine makes the transition as shown below :

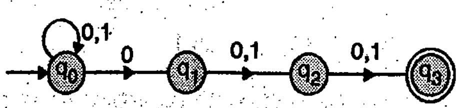
$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$$

- 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 2.6.3

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^*	ϕ	ϕ

(b) State transition table

Fig. Ex. 2.6.3 : NFA for example 2.6.3

- 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 2.6.4

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 :

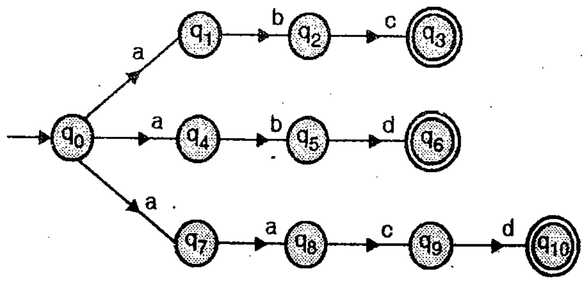


Fig. Ex. 2.6.4(a) : State diagram of NFA

- Machine can guess and take one of the three paths as shown :
  1. Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$ for string abc.
  2. Path $q_0 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6$ for string abd.
  3. Path $q_0 \rightarrow q_7 \rightarrow q_8 \rightarrow q_9 \rightarrow q_{10}$ for string aacd.

	a	b	c	d
$\rightarrow q_0$	$\{q_1, q_4, q_7\}$	ϕ	ϕ	ϕ
q_1	ϕ	q_2	ϕ	ϕ
q_2	ϕ	ϕ	q_3	ϕ
q_3^*	ϕ	ϕ	ϕ	ϕ
q_4	ϕ	q_5	ϕ	ϕ
q_5	ϕ	ϕ	ϕ	q_6
q_6^*	ϕ	ϕ	ϕ	ϕ
q_7	q_8	ϕ	ϕ	ϕ
q_8	ϕ	ϕ	q_9	ϕ
q_9	ϕ	ϕ	ϕ	q_{10}
q_{10}^*	ϕ	ϕ	ϕ	ϕ

Fig. Ex. 2.6.4 (b) : State table for NFA

Example 2.6.5

Construct the NFA and DFA for the following languages.

(i) L = {x ∈ {a, b, c}* : x contains exactly one b immediately following c}

(ii) $L = \{x \in \{0, 1\}^*$: x is starting with 1 and $|x|$ is divisible by 3$\}$

(iii) $L = \{x \in \{a, b\}^*$: x contains any number of a's followed by at least one b$\}$

Solution :

(i) L $\{x \in \{a, b, c\}^*$: x contains exactly one b 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.

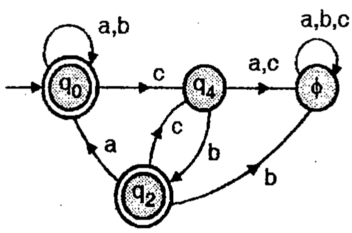


Fig. Ex. 2.6.5(a) : A DFA standing for both DFA and NFA

(ii) L $\{x \in \{0, 1\}^*$: x is starting with 1 and $|x|$ is divisible by 3$\}$

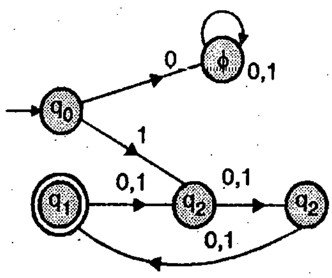


Fig. Ex. 2.6.5(b) : A DFA standing for both DFA and NFA

(iii) L $\{x \in \{a, b\}^*$: x contains any number of a's followed by at least one b$\}$

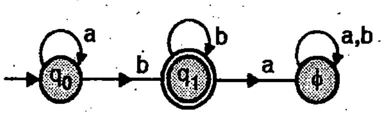


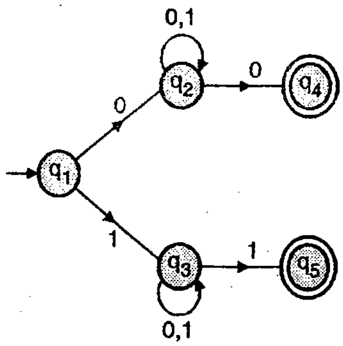
Fig. Ex. 2.6.5(c) : A DFA standing for both DFA and NFA

Note : Technically, a DFA with ϕ transitions omitted may be called an NFA.

Example 2.6.6

Design a NFA for a binary number where the first and the last digits are same.

Solution :



(a) State transition diagram

	a	b
$\rightarrow q_1$	q_2	q_2
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. 2.6.6

- 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 2.6.7 SPPU - May 14, 6 Marks

An NFA with states 1-5 and input alphabet {a, b} has following transition table.

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

(a) Draw a transition diagram

(b) Calculate δ*(1, ab)

(c) Calculate δ*(1, abaab)

Solution :

(a) Transition diagram is given in Fig. Ex. 2.6.7.

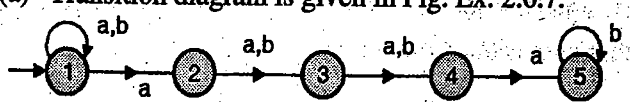


Fig. Ex. 2.6.7 : State transition diagram

(b) $\delta^*(1, ab) = \delta((\delta(1, a)), b)$

$= \delta(\{1, 2\}, b) = \delta(1, b) \cup \delta(2, b)$

$= \{1\} \cup \{3\} = \{1, 3\}$

(c) $\delta^*(1, abaab) = \delta^*(\{1, 2\}, baab)$

$= \delta^*(\{1, 3\}, aab)$

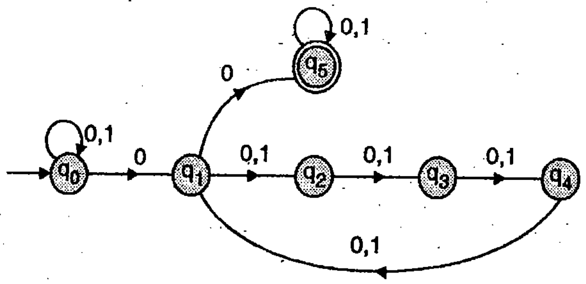
$= \delta^*(\{1, 2, 4\}, ab) = \delta(\{1, 2, 3, 5\}, b)$

$= \{1, 3, 4, 5\}$

Example 2.6.8

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 >=0.

Solution :



(a) State transition diagram

	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. 2.6.8 : NFA for Example 2.6.8

- 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 2.6.9

Design NFA for the set of strings on the alphabet {0, 1} that start with 01 and end with 10.

Solution :

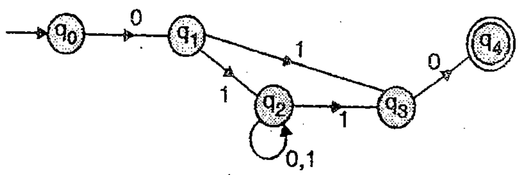


Fig. Ex. 2.6.9 : NFA for example 2.6.9

- 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.

2.6.3 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.

i.e. $Q = (q_0, q_1, q_2)$

The machine could be in any of the following states :

1. ϕ
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)

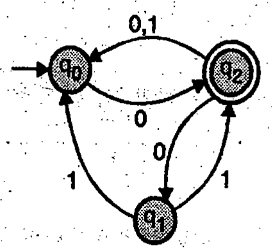
These subsets form a power set on Q, which is given by 2^Q.

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.

$\delta =$



$\delta =$

	0	1
$\rightarrow q_0$	$\{q_2\}$	ϕ
q_1	ϕ	$\{q_0, q_2\}$
q_2^*	$\{q_0, q_1\}$	$\{q_0\}$

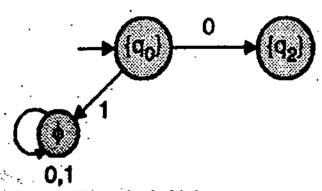
Fig. 2.6.5 : NFA to be converted to DFA

Algorithm for construction of successor table for the NFA of Fig. 2.6.5 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. $\delta(q_0, 1) \Rightarrow \phi$.



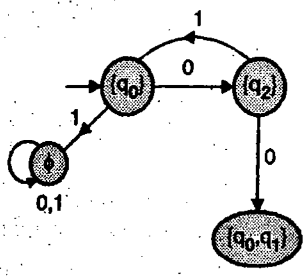
	0	1
$\rightarrow\{q_0\}$	$\{q_2\}$	ϕ

Fig. 2.6.6(a)

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 $\{q_0\}$.



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

Fig. 2.6.6(b)

Step 3 : A new subset $\{q_0, q_1\}$ is generated. Successors of the subset $\{q_0, q_1\}$ are generated.

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

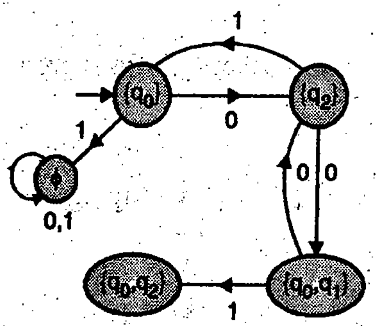
$= \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_2\} \cup \phi = \{q_2\}$

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

$= \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \phi \cup \{q_0, q_2\} = \{q_0, q_2\}$



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

Fig. 2.6.6(c)

Step 4 : A new subset $\{q_0, q_2\}$ is generated successors of the subset $\{q_0, q_2\}$ are generated.

0 – successor of $\{q_0, q_1\} = \delta(\{q_0, q_2\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_2, 0)$

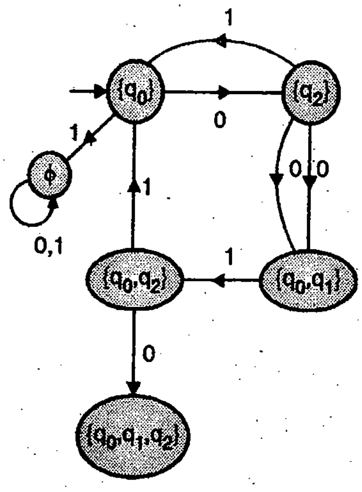
$= \{q_2\} \cup \{q_2, q_1\}$

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

1 – successor of $\{q_0, q_2\} = \delta(\{q_0, q_2\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_2, 1)$

$= \phi \cup \{q_0\} = \{q_0\}$



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

Fig. 2.6.6(d)

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.

0 – successor of $\{q_0, q_1, q_2\} = \delta(\{q_0, q_1, q_2\}, 0)$

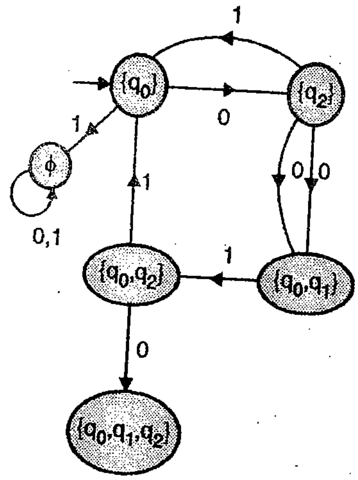
$= \delta(\{q_0, q_1\}\ 0) \cup \delta(q_2, 0)$

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

1 – successor of $\{q_0, q_1, q_2\} = \delta(\{q_0, q_1, q_2\}, 1)$

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

$= \{q_0, q_2\} \cup \{q_0\} = \{q_0, q_2\}$

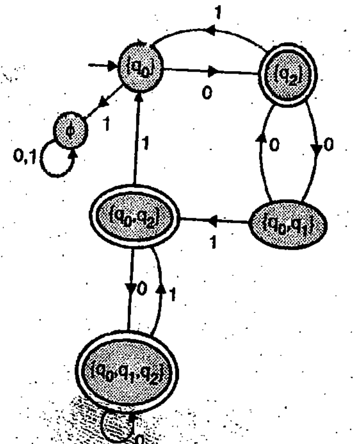


	0	1
$\rightarrow\{q_0\}$	$\{q_2\}$	ϕ
$\{q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_0\}$

Fig. 2.6.6(e)

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. 2.6.6. Every subset containing q_2 should be takes as a final state.



	0	1
$\rightarrow\{q_0\}$	$\{q_2\}$	ϕ
$\{q_2\}^*$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}^*$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
$\{q_0, q_1, q_2\}^*$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$

Fig. 2.6.6(f) : Final DFA after marking of accepting states

Theorem 2.6.1

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 $\omega = \varepsilon$ with $|\omega| = 0$, where $|\omega|$ is length of ω.

Starting state for both NFA and DFA are taken as q_0. When $\omega = \varepsilon$, 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 $\{p_1, p_2, \ldots p_k\}$

$$\text{then, } \delta_2(q_0, \omega_{n+1}) = \bigcup_{i=1}^{k} \delta_2(p_i, a) \quad \ldots\text{(i)}$$

$$\text{or } \delta_2(\{p_1, p_2, \ldots p_k\}, a) = \bigcup_{i=1}^{k} \delta_1(p_i, a) \quad \ldots\text{(ii)}$$

$$\text{also, } \delta_2(q_0, \omega_n) = \{p_1, p_2, \ldots p_k\} \quad \ldots\text{(iii)}$$

from (i), (ii) and (iii) we get,

$$\delta_2(q_0, \omega_{n+1}) = \delta_2(\delta_2(q_0, \omega_n), a)$$
$$= \delta_2(\{p_1, p_2, \ldots p_k\}, a)$$

$$= \bigcup_{i=1}^{k} \delta_i(p_i, a) = \delta_i(q_0, s_{n+1})$$

Thus, the result is true for $|\omega| = n + 1$, hence it is always true.

Example 2.6.10 SPPU - May 13, 8 Marks

Convert to a DFA the following NFA :

	0	1
→p	{p, q}	{p}
q	{r}	{r}
r	{s}	ϕ
s*	{s}	{s}

Solution :

	0	1	
→{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, s}	{p, q, r, s}	(p, r, s)	
{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. 2.6.10 (a) : State transition table of DFA

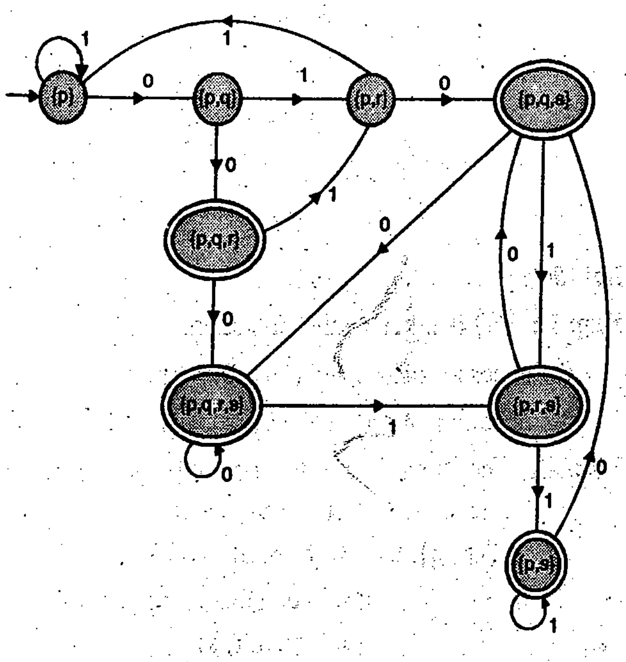


Fig. Ex. 2.6.10 (b) : State transition diagram of DFA

– 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 2.6.11

Convert to a DFA the following NFA.

	0	1
→{p}	{q, s}	{q}
q*	{r}	{q, r}
r	{s}	{p}
s*	ϕ	{p}

Solution :

Step 1 : {p} is taken as the first subset.

0-Successor of $\{p\} = \delta(\{p\}, 0) = \{q, s\}$

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

	0	1
→{p}	{q, s}	{q}

Step 2 : Two new subsets {q, s} and {q} have been generated. Successors of {q, s} and {q} are calculated.

$$\delta(\{q, s\}, 0) = \delta(q, 0) \cup \delta(s, 0) = \{r\} \cup \phi = \{r\}$$

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

$$= \{p, q, r\}$$

$$\delta(\{q\}, 0) = \{r\}$$

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

	0	1
→{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.

$$\delta(\{r\}, 0) = \{s\}$$

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

$$\delta(\{p, q, r\}, 0) = \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0)$$

$$= \{q, s\} \cup \{r\} \cup \{s\}$$

$$= \{q, r, s\}$$

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

$= \{q\} \cup \{q, r\} \cup \{p\}$
$= \{p, q, r\}$

$\delta(\{q, r\}, 0) = \delta(q, 0) \cup \delta(r, 0)$
$= \{r\} \cup \{s\} = \{r, s\}$

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

	0	1
→{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}

Step 4 : Three new subsets {s}, {r, s} and {q, r, s} are generated. Their successors are calculated.

$\delta(\{s\}, 0) = \phi$
$\delta(\{s\}, 1) = p$
$\delta(\{r, s\}, 0) = \{s\}$
$\delta(\{r, s\}, 1) = \{p\}$
$\delta(\{q, r, s\}, 0) = \{r, s\}$
$\delta(\{q, r, s\}, 1) = \{p, q, r\}$

	0	1
→{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}	φ	{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 drawn in Fig. Ex. 2.6.11(b).

	0	1
→{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}*	φ	{p}
{r, s}*	{s}	{p}
{q, r, s}*	{r, s}	{p, q, r}

Fig. Ex. 2.6.11(a) : State transition table of DFA

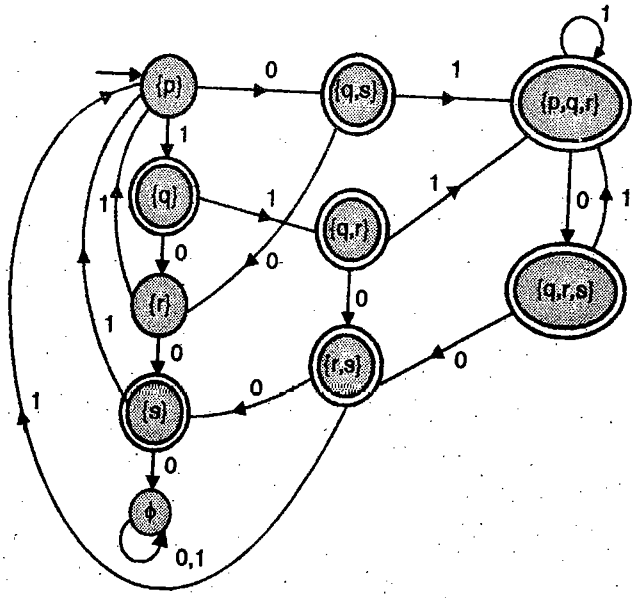


Fig. Ex. 2.6.11(b) : State transition diagram of the DFA

Example 2.6.12

Convert the following NFA to a DFA and informally describe the language it accepts.

	0	1
→p	{p, q}	{p}
q	{r, s}	{t}
r	{p, r}	{t}
s*	φ	φ
t*	φ	φ

Solution :

Step 1 : {p} is taken as the first subset.

0-Successor of $\{p\} = \delta(\{p\}, 0) = \{p, q\}$

1-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.

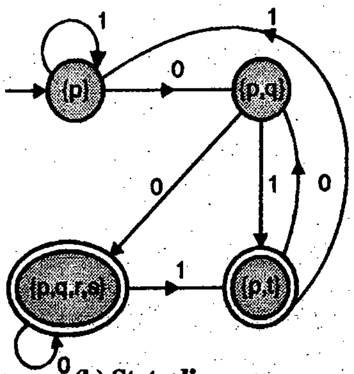
$$\delta(\{p, q, r, s\}, 0) = \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) \cup \delta(s, 0)$$
$$= \{p, q\} \cup \{r, s\} \cup \{p, r\} \cup \phi$$
$$= \{p, q, r, s\}$$
$$\delta(\{p, q, r, s\}, 1) = \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1) \cup \delta(s, 1)$$
$$= \{q\} \cup \{t\} \cup \{t\} \cup \phi = \{p, t\}$$
$$\delta(\{p, t\}, 0) = \delta(p, 0) \cup \delta(t, 0)$$
$$= \{p, q\} \cup \phi = \{p, q\}$$
$$\delta(\{p, t\}, 1) = \delta(p, 1) \cup \delta(t, 1)$$
$$= \{p\} \cup \phi = \{p\}$$

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
→{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. 2.6.12 : Final DFA for example 2.6.12

Example 2.6.13

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 :

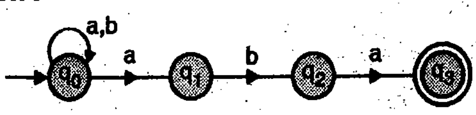


Fig. Ex. 2.6.13(a) : Non-deterministic finite automata for example 2.6.13

Non-determinism should be utilized to full extent while designing a NFA. A string of length n, ending in aba can be recognized by the NFA given in Fig. Ex. 2.6.13(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

a-successor of $\{q_0\} = \delta(q_0, a) = \{q_0, q_1\}$

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.

$$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$$
$$= \{q_0, q_1\} \cup \phi = \{q_0, q_1\}$$
$$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$$
$$= \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$$

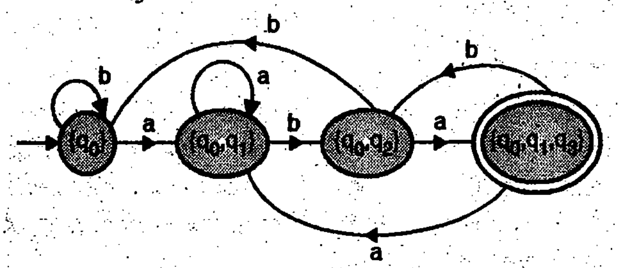
Step 3 : A new subset $\{q_0, q_2\}$ is generated. Successors of $\{q_0, q_2\}$ are calculated.

$$\delta(\{q_0, q_2\}, a) = \delta(q_0, a) \cup \delta(q_2, a)$$
$$= \{q_0, q_1\} \cup \{q_3\}$$
$$= \{q_0, q_1, q_3\}$$
$$\delta(\{q_0, q_2\}, b) = \delta(q_0, b) \cup \delta(q_2, b)$$
$$= \{q_0\} \cup \phi = \{q_0\}$$

Step 4 : A new subset $\{q_0, q_1q_3\}$ is generated. Successors of $\{q_0, q_1q_3\}$ are calculated.

$$\delta(\{q_0, q_1q_3\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a)$$
$$= \{q_0, q_1\} \cup \phi \cup \phi$$
$$= \{q_0, q_1\}$$
$$\delta(\{q_0, q_1q_3\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b)$$
$$= \{q_0\} \cup \{q_2\} \cup \phi = \{q_0, q_2\}$$

No, new subset is generated. Every subset containing q_3 is marked as a final state.



(b) State diagram of the DFA

Fig. Ex. 2.6.13 (Contd....)

	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

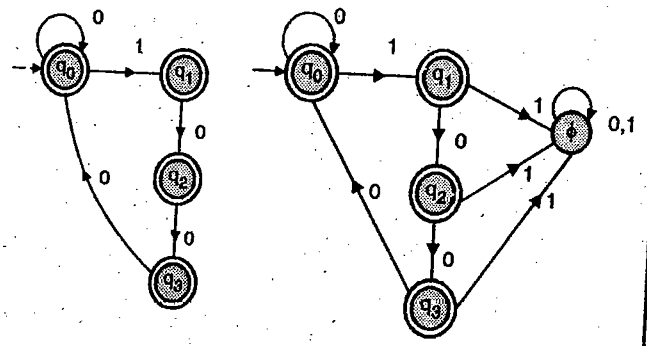
Fig. Ex. 2.6.13

Example 2.6.14

Construct an NFA an 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 as NFA.

NFA is given in Fig. Ex. 2.6.14(a).



(a) A NFA **(b) A DFA**

Fig. Ex. 2.6.14

From the Fig. Ex. 2.6.14(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 be shown.

Above NFA can be converted into a DFA by introducing the dead state. DFA is given in the Fig. Ex. 2.6.14(b).

Example 2.6.15

Construct an 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. 2.6.15(a).

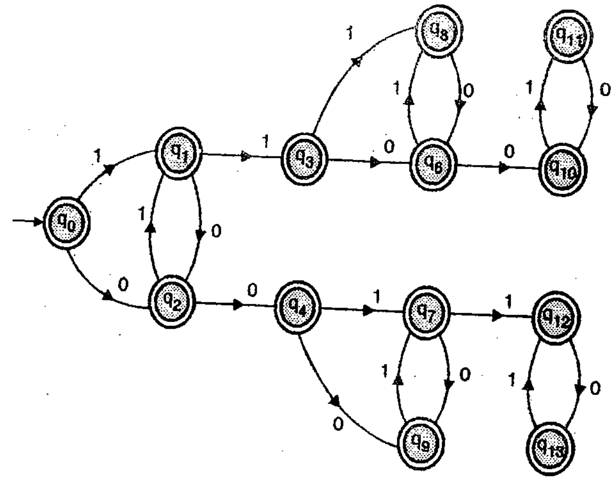


Fig. Ex. 2.6.15(a) : A NFA

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

Above NFA can be converted into a DFA by introducing an explicit dead state.

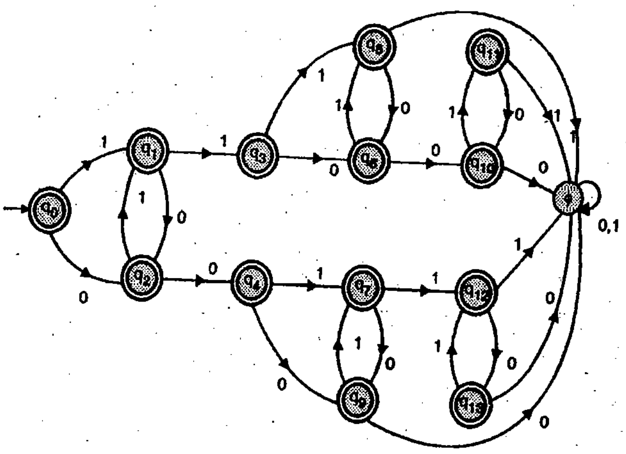


Fig. Ex. 2.6.15(b) : A DFA

Example 2.6.16 SPPU - May 14. 8 Marks

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. Technically, a DFA with ϕ transitions not mentioned can be called a NFA.

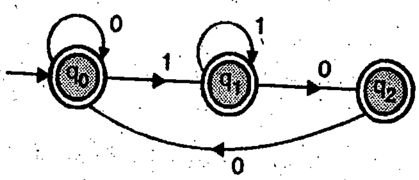


Fig. Ex. 2.6.16(a) : A NFA

- 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 failure Fig. Ex. 2.6.16(b).

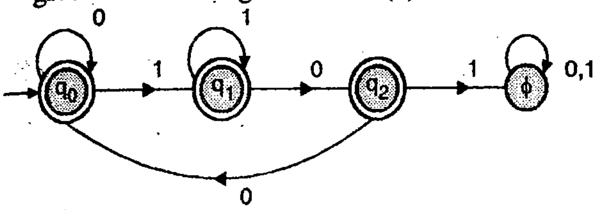


Fig. Ex. 2.6.16(b) : A DFA

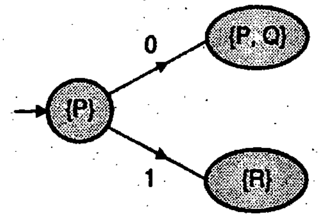
Example 2.6.17 SPPU - Aug. 15. 4 Marks

Convert following NFA into its equivalent DFA.

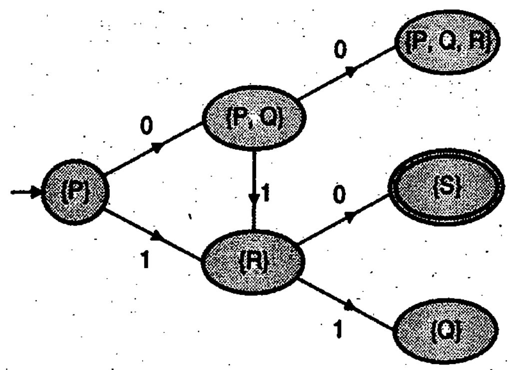
Q \ Σ →	0	1
→ P	P, Q	R
Q	R	R
R	S	Q
*S	S	S

Solution

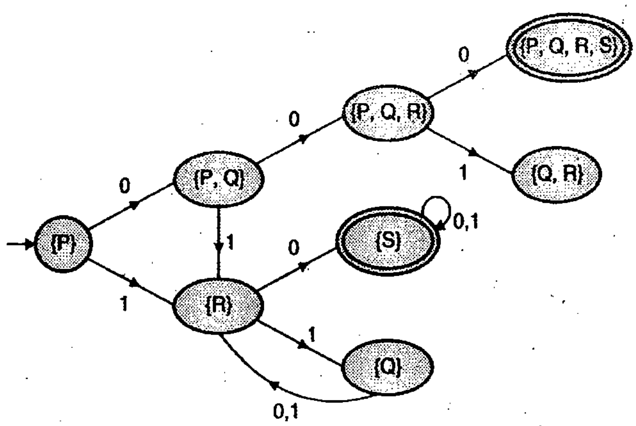
Step 1 : Transitions for state {p} are written



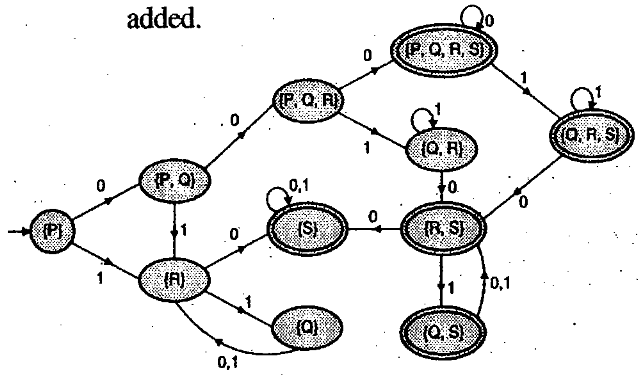
Step 2 : Transitions for states {P,Q } and {R} are



Step 3 : Transitions for {P, Q, R}, {S} and {Q} are written

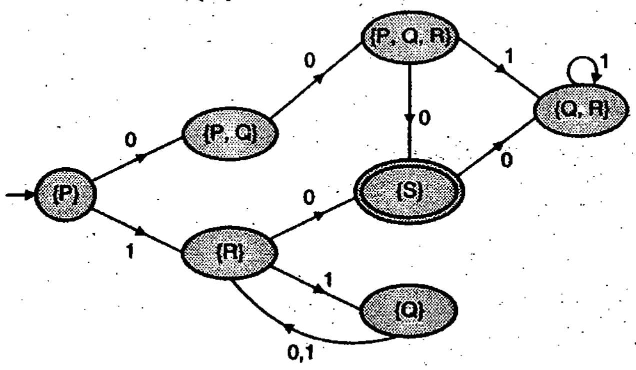


Step 4 : Transition for {Q, R} and {P, Q, R, S} are added.



Step 5 : Further transitions are added in transition graph obtained in step 4.

Step 6 : Final states can be merged into a single final state {S}



Example 2.6.18 SPPU - May 12. 8 Marks

Construct an 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 an NFA.

- State q_1 is for previous character as 0.

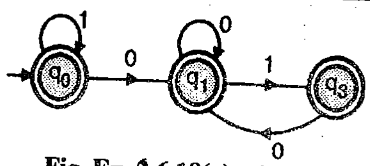


Fig. Ex. 2.6.18(a) : An NFA

- 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.

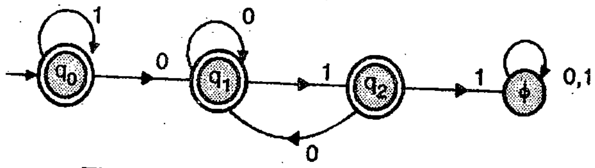


Fig. Ex. 2.6.18(b) : An equivalent DFA

Example 2.6.19

Construct a DFA equivalent to the following NDFA given below :

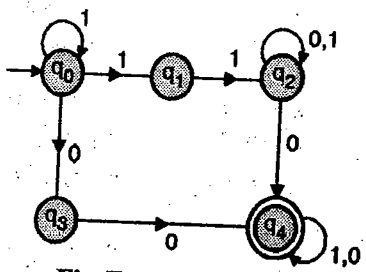


Fig. Ex. 2.6.19

Solution :

Construction of subsets :

Step 1 : $\delta(\{q_0\}, 0) = \{q_3\}$

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

Step 2 : $\delta(\{q_3\}, 0) = \{q_4\}$

$\delta(\{q_3\}, 1) = \phi$

$\delta(\{q_0, q_1\}, 0) = \{q_3\}$

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

Step 3 : $\delta(\{q_4\}, 1) = \{q_4\}$

$\delta(\{q_4\}, 1) = \{q_4\}$

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

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

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 : $\delta(\{q_2, q_4\}, 0) = \{q_2, q_4\}$

$\delta(\{q_2, q_4\}, 1) = \{q_2, q_4\}$

FA is given in Fig. Ex. 2.6.19(a).

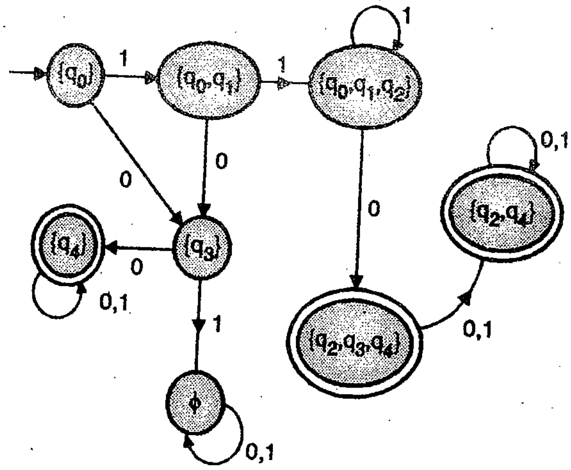


Fig. Ex. 2.6.19(a) : State diagram of the DFA for example 2.6.19

Example 2.6.20

Let m = ($\{q_1, q_3, q_4\}$, {0, 1}, δ, { 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. 2.6.20(a).

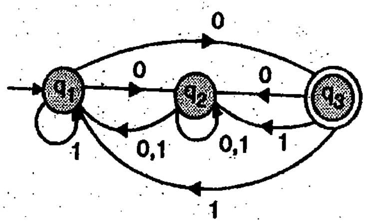


Fig. Ex. 2.6.20(a) : Transition of the given NFA

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. 2.6.20(b).

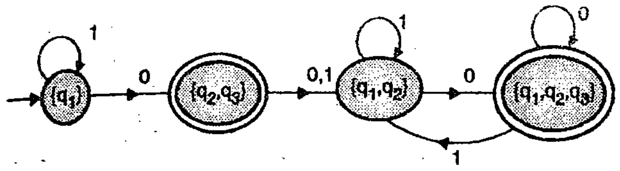


Fig. Ex. 2.6.20(b) : Equivalent DFA

Example 2.6.21

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.

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

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

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

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. 2.6.21.

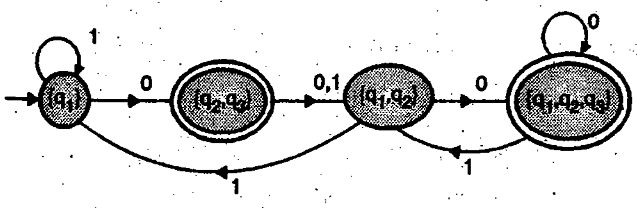


Fig. Ex. 2.6.21

Example 2.6.22

Let M = ($\{q_0, q_1\}$, {0, 1}, δ, 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 : $\delta(\{q_1\}, 0) = \phi$

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

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. 2.6.22.

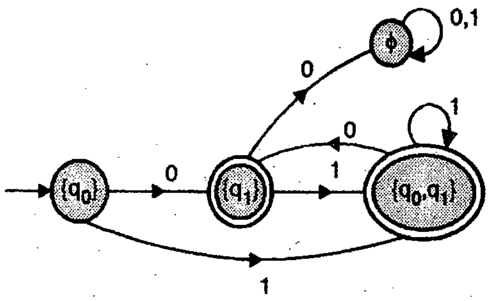


Fig. Ex.2.6.22 : An equivalent DFA

Example 2.6.23

Convert following NFA to corresponding DFA.

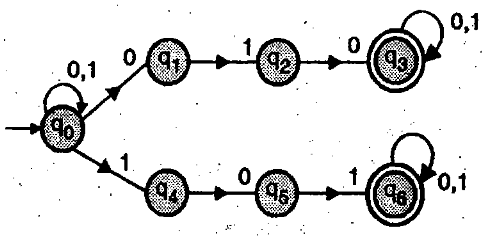


Fig. Ex. 2.6.23

Solution :

Step 1 : q_3 and q_6 have identical NFA and they can be merged into a single state q_3.

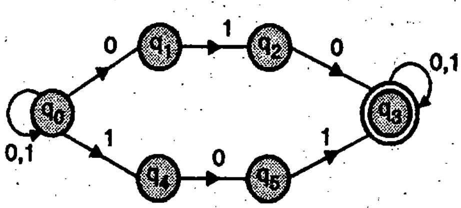


Fig. Ex. 2.6.23(a)

Step 2 :

We start with the subset { q_0}

0-successor of $\{q_0\} = \{q_0, q_1\}$

1-successor of $\{q_1\} = \{q_0, q_4\}$

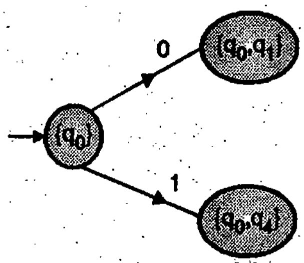


Fig. Ex. 2.6.23(b)

Step 3 : Two new subsets { q_0, q_1} and { q_0, q_4} are generated. Their successors are calculated.

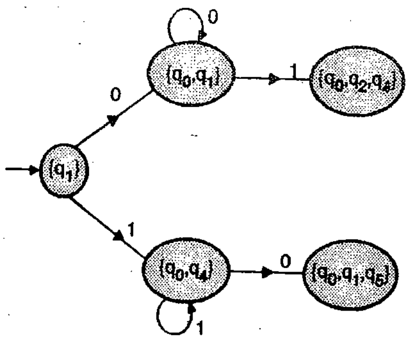


Fig. Ex. 2.6.23(c)

Step 4 : Two new subsets { q_0, q_2, q_4} and {q_0, q_1, q_5} are generated. Their successors are calculated.

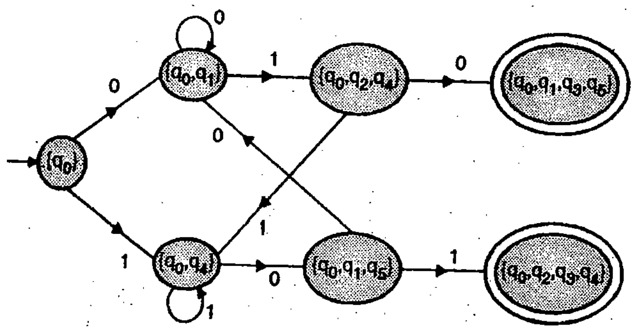


Fig. Ex. 2.6.23(d)

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.

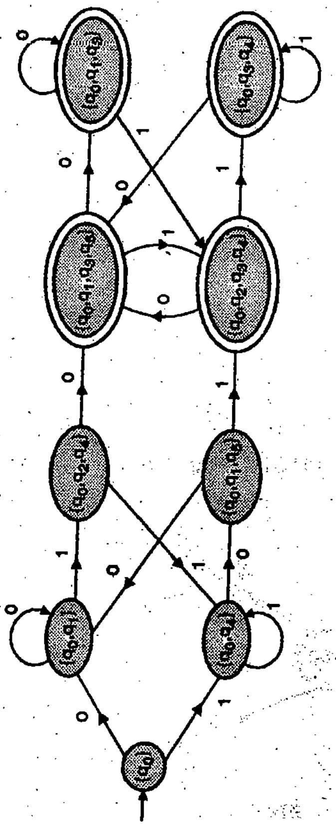


Fig. Ex. 2.6.23(e)

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}

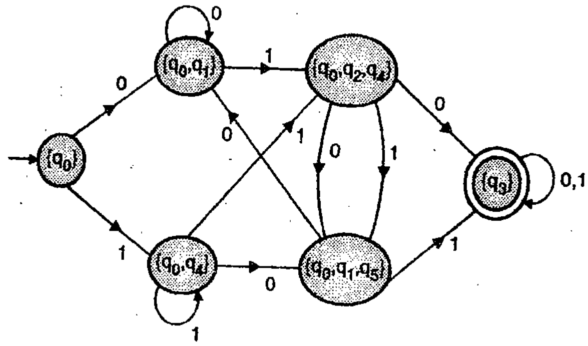


Fig. Ex. 2.6.23(f) : Final DFA

Example 2.6.24

Convert following NFA to DFA.

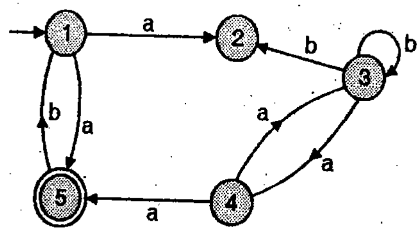


Fig. Ex. 2.6.24

Solution :

Step 1 : Starting state {1} is taken as the first subset. Its successors are calculated.

$$\delta(\{1\}, a) = \{2, 5\}$$
$$\delta(\{1\}, b) = \phi$$

Step 2 : A new subset {2, 5} is generated. Its successors are calculated.

$$\delta(\{2, 5\}, a) = \delta(2, a) \cup \delta(5, a) = \phi \cup \phi = \phi$$
$$\delta(\{2, 5\}, b) = \delta(2, b) \cup \delta(5, b) = \phi \cup \{1\} = \{1\}$$

A subset containing 5 is marked as a final state. The equivalent DFA is given in Fig. Ex. 2.6.24(a).

	a	b
→{1}	{2, 5}	ϕ
{2, 5}*	ϕ	{1}
ϕ	ϕ	ϕ

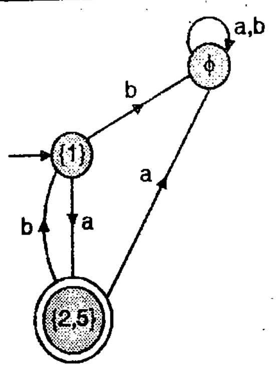


Fig. Ex. 2.6.24(a) : DFA for example 2.6.24

Example 2.6.25

Convert the following NFA to DFA.

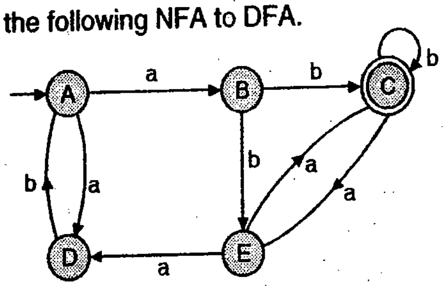


Fig. Ex. 2.6.25

Solution :

In some cases where number of states are large, it is convenient to derive subsets using a state transition table. State transition table for the given NFA is shown in Fig. Ex. 2.6.25(a).

	a	b
$\rightarrow$A	{B, D}	ϕ
B	ϕ	{C, E}
C*	{E}	{C}
D	ϕ	{A}
E	{C, D}	ϕ

Fig. Ex. 2.6.25(a) : State transition table for NFA of example 2.6.25

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.

$$\delta(\{A, C, E\}, a) = \delta(A, a) \cup \delta(C, a) \cup \delta(E, a) = \{B, D\} \cup \{E\} \cup \{C, D\} = \{B, C, D, E\}$$
$$\delta(\{A, C, E\}, b) = \delta(A, b) \cup \delta(C, b) \cup \delta(E, b) = \phi \cup \{C\} \cup \phi = \{C\}$$

Step 4 : Two new subsets {B, C, D, E} and {C} are generated. Their successors are calculated.

$$\delta(\{B, C, D, E\}, a) = \delta(B, a) \cup \delta(C, a) \cup \delta(D, a) \cup \delta(E, a) = \phi \cup \{E\} \cup \phi \cup \{C, D\} = \{C, D, E\}$$
$$\delta(\{B, C, D, E\}, b) = \delta(B, b) \cup \delta(C, b) \cup \delta(D, b) \cup \delta(E, b) = \{C, E\} \cup \{C\} \cup \{A\} \cup \phi = \{A, C, E\}$$
$$\delta(\{C\}, a) = \{E\}$$
$$\delta(\{C\}, b) = \{C\}$$

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. 2.6.25(b).

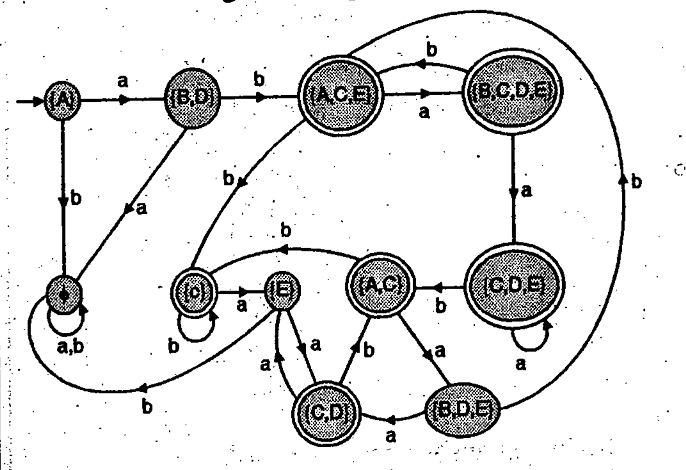


Fig. Ex. 2.6.25(b) : Final DFA for example 2.6.25

Example 2.6.26

The transition table of a NFA 'M' is given below. Construct a DFA equivalent to M. δ is

	0	1	2
q_0	q_1, q_4	q_4	q_2, q_3
q_1	–	q_4	–
q_2	–	–	q_2, q_3
q_3	–	q_4	–
q_4^*	–	–	–

Solution :

Step 1 : Transitions from the start state q_0 are written.

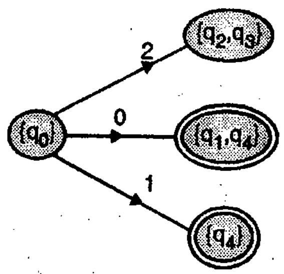


Fig. Ex. 2.6.26

Step 2 : Transitions from $\{q_2, q_3\}$, $\{q_1, q_4\}$ and $\{q_4\}$ are written.

$$\delta(\{q_2, q_3\}, 0) = \delta(q_2, 0) \cup \delta(q_3, 0) = \phi$$

$$\delta(\{q_2, q_3\}, 1) = \delta(q_2, 1) \cup \delta(q_3, 1) = \{q_4\}$$

$$\delta(\{q_2, q_3\}, 2) = \delta(q_2, 2) \cup \delta(q_3, 2) = \{q_2, q_3\}$$

Similarly,

$$\delta(\{q_1, q_4\}, 0) = \phi$$

$$\delta(\{q_1, q_4\}, 1) = \{q_4\}$$

$$\delta(\{q_1, q_4\}, 2) = \phi$$

The DFA, after the above step is

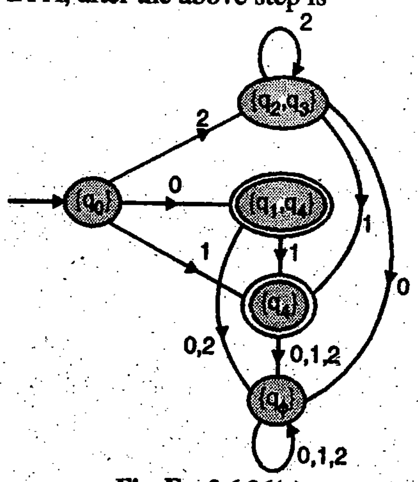


Fig. Ex. 2.6.26(a)

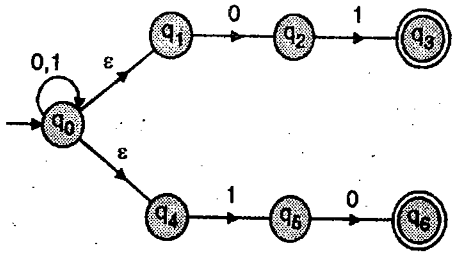
No more transitions can be added.

Syllabus Topic : Epsilon-NFA

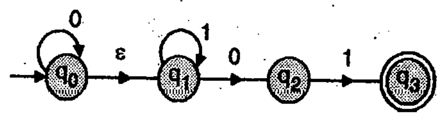
2.6.4 NFA with ε-Transitions

ε stands for a null symbol. An ε transition allows transition on ε (or no input), the empty string. This implies that a machine can make a transition without any input.

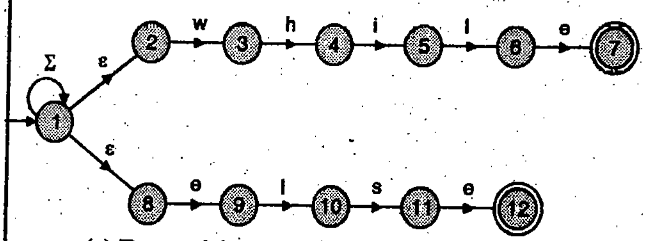
- When finding the string described by a path containing arc with label ε, the ε symbols are discarded.
- The use of ε-transition may simplify a transition graph by reducing number of labelled arcs.



(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. 2.6.7 : Few sample ε-NFAs

2.6.4.1 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 an equivalence between the systems with ε-move and without ε-move. This can be understood with the help of an example.

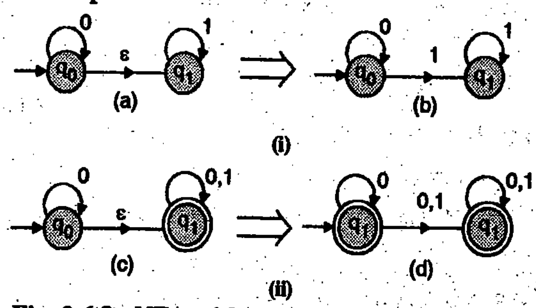


Fig. 2.6.8 : NFA with ε-move converted to NFA without ε-move

If there is an ε-move between any two states q_i and q_j, then ε-move can be removed as follows :

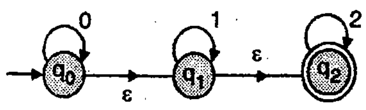
(1) All moves from q_j should also originate from q_i. For example : If there is move from q_j to q_k , then we must add a move on q_i to q_k.

(2) If q_j is a final state then make q_i as a final state. If the ε-closure of q_i contains a final state then q_i should be declared as a final state.

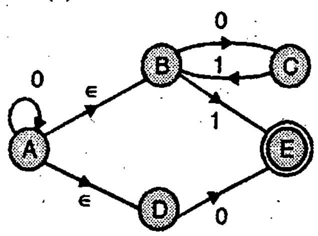
Example 2.6.27 SPPU - Dec. 15. 6 Marks

(a) Use direct method to find an equivalent NFA without ε-move for the NFA given below :

(b) Convert the given NFA – ∈ to an NFA.



(a) NFA with ε-move



(b)

Fig. Ex. 2.6.27

Solution :

(a)

Step 1 : Transitions from q_2 are also duplicated on q_1. q_1 is made a final state.

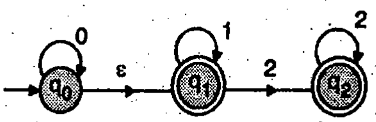


Fig. Ex. 2.6.27(c) : ε-move between q_1 and q_2 is removed

Step 2 : Transition from q_1 are also duplicated on q_0. q_0 is made a final state.

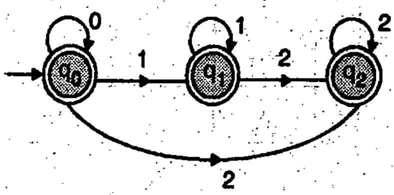


Fig. Ex. 2.6.27(d) : Final NFA without ε-move after removing ε-move between q_0 and q_1

(b)

Outgoing transitions from B and D are duplicated on A as $\delta(A, \in) = B$ and $\delta(A, \in) = D$.

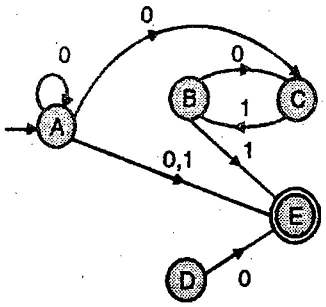


Fig. Ex. 2.6.27(e)

The state D can be deleted, hence the final NFA is drawn as given below.

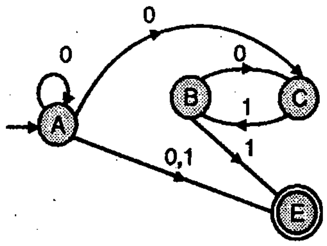


Fig. Ex. 2.6.27(f)

2.6.4.2 The Formal Notation for an ε-NFA

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.

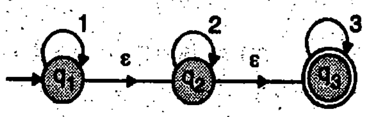
F = $F \subseteq Q$ is a set of final/accepting states

δ = A transition function from $Q \times \{\Sigma \cup \{\varepsilon\})$ 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. Ex. 2.6.27(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. 2.6.9.



	ε	1	2	3
$\rightarrow q_1$	$\{q_2\}$	$\{q_1\}$	ϕ	ϕ
q_2	$\{q_3\}$	ϕ	$\{q_2\}$	ϕ
q_3^*	ϕ	ϕ	ϕ	$\{q_3\}$

Fig. 2.6.9 : An NFA used for formal notation

2.6.4.3 ε-Closures

SPPU - Dec. 13

University Question

Q. Give formal definition of epsilon closure with suitable example. **(SPPU – Dec. 2013, 2 Marks)**

ε-closures 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.

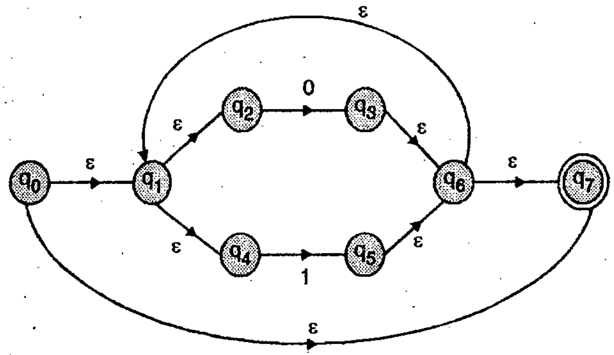


Fig. 2.6.10 : NFA considered for calculation of ε-closures

ε-closures of various states in Fig. 2.6.10 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-closures 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 2.6.28

A transition table is given for another with NULL with seven states.

q	δ(q, a)	δ(q, b)	δ(q, ϕ)
1	{5}	ϕ	{4}
2	{1}	ϕ	ϕ
3	{ϕ}	{2}	ϕ
4	ϕ	{7}	{3}
5	ϕ	ϕ	{1}
6	ϕ	{5}	{4}
7	{6}	ϕ	ϕ

(a) **Draw a transition diagram.**

(b) **Calculate δ*(1, ba)**

Solution :

(a) Transition diagram is in Fig. Ex. 2.6.28.

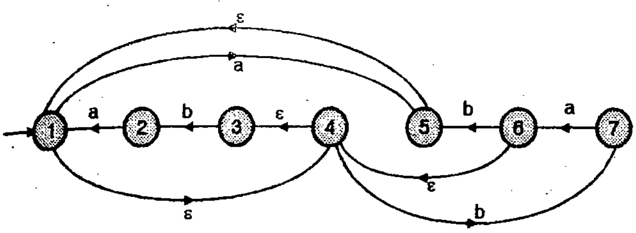


Fig. Ex. 2.6.28 : State transition diagram

(b) Calculate $\delta^*(1,ba)$

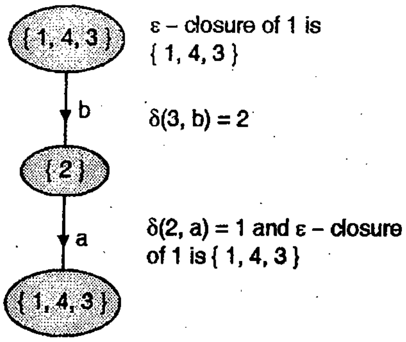


Fig. Ex. 2.6.28(a)

$\therefore \delta^*(1,ba) = \{1,4,3\}$

2.6.4.4 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. 2.6.10.

$$\delta_\varepsilon(q_2,0) = \varepsilon\text{-closure } (q_3) = \{q_3, q_6, q_7, q_1, q_2, q_4\}$$

Example 2.6.29

Find an equivalent DFA for the ε-NFA given in Fig. Ex. 2.6.29(a).

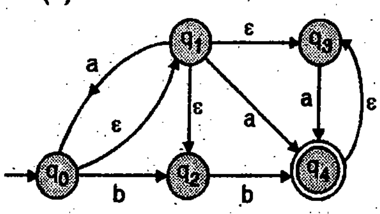


Fig. Ex. 2.6.29(a)

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.

$$\varepsilon\text{-closure } (q_0) = \{q_0, q_1, q_2, q_3\}$$

a-successor of $\{q_0, q_1, q_2, q_3\} = \varepsilon\text{-closure } (\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a))$

$= \varepsilon\text{-closure } (\phi \cup \{q_0, q_4\} \cup \phi \cup \{q_4\})$

$= \varepsilon\text{-closure } (\{q_0, q_4\})$

$= \varepsilon\text{-closure } \{q_0\} \cup \varepsilon\text{-closure } (q_4)$

$= \{q_0, q_1, q_2, q_3\} \cup \{q_3, q_4\}$

$= \{q_0, q_1, q_2, q_3, q_4\}$

b-successor of $\{q_0, q_1, q_2, q_3\} = \varepsilon\text{-closure } (\delta (q_0, b) \cup \delta (q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b))$

$= \varepsilon\text{-closure } (\{q_2\} \cup \phi \{q_4\} \cup \phi)$

$= \varepsilon\text{-closure } (\{q_2, q_4\})$

$= \varepsilon\text{-closure } (q_2) \cup \varepsilon\text{-closure } (q_4)$

$= \{q_2\} \cup \{q_3, q_4\} = \{q_2, q_3, q_4\}$

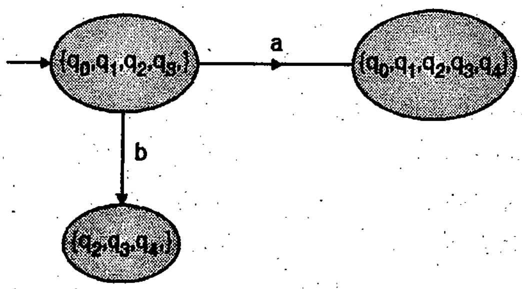


Fig. Ex. 2.6.29(b)

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.

a-successor of $\{q_2, q_3, q_4\} = \varepsilon\text{-closure } (\delta(q_2, a) \cup \delta(q_3, a) \cup (q_4, a))$

$= \varepsilon\text{-closure } (\phi \cup \{q_4\} \cup \phi)$

$= \varepsilon\text{-closure } (q_3, q_4)$

$= \varepsilon\text{-closure } (q_3) \cup \varepsilon\text{-closure } (q_4)$

$= \{q_3\} \cup \{q_3, q_4\} = \{q_3, q_4\}$

b-successor of $\{q_2, q_3, q_4\} = \varepsilon\text{-closure } (\delta(q_2, b) \cup \delta(q_3, b) \cup (q_4, b))$

$= \varepsilon\text{-closure } (\{q_4\} \cup \phi \cup \phi)$

$= \varepsilon\text{-closure } (q_4) = \{q_3, q_4\}$

a-successor of $\{q_0, q_1, q_2, q_3, q_4\}$ = a-successor of $\{q_0, q_1, q_2, q_3\} \cup$ a-successor $\{q_4\}$

$= \{q_0, q_1, q_2, q_3, q_4\} \cup \phi$

$= \{q_0, q_1, q_2, q_3, q_4\}$

b-successor of $\{q_0, q_1, q_2, q_3, q_4\}$ = b-successor of $\{q_0, q_1, q_2, q_3\} \cup$ b-successor of $\{q_4\}$

$= \{q_2, q_3, q_4\} \cup \phi = \{q_2, q_3, q_4\}$

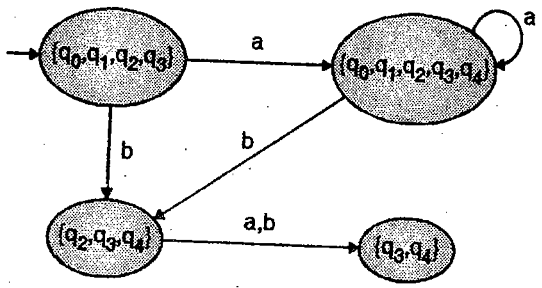


Fig. Ex. 2.6.29(c)

Step 4 : A new subset $\{q_3, q_4\}$ has been generated. Successors of $\{q_3, q_4\}$ are calculated.

a-successor of $\{q_3, q_4\}$ = ε-closure $(\delta(q_3, a) \cup \delta(q_4, b))$

$= \varepsilon\text{-closure}(\{q_4\} \cup \phi) = \varepsilon\text{-closure}(q_4)$

$= \{q_3, q_4\}$

b-successor of $\{q_3, q_4\}$ = ε-closure $(\delta(q_3, b) \cup \delta(q_4, b))$

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

No further subsets are generated. Every subset containing q_4 (final state in Fig. Ex. 2.6.29(d)) is marked as final state.

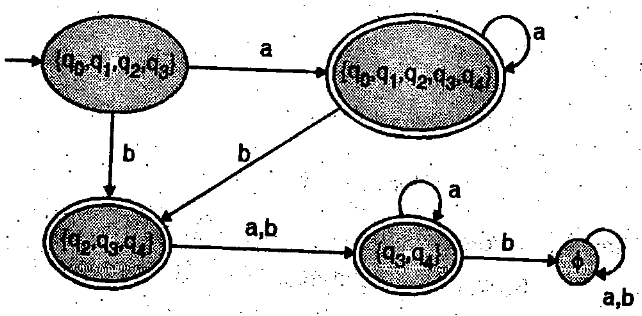


Fig. Ex. 2.6.29(d) : Final DFA

Example 2.6.30

Consider the following ε-NFA

	ε	a	b	c
→p	ϕ	{p}	{q}	{r}
q	{p}	{q}	{r}	ϕ
r*	{q}	{r}	ϕ	{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

ε-closure (p) = {p}

a-successor of {p} = ε-closure (δ(p, a))

= ε-closure (p) = p

b-successor of {p} = ε-closure (δ(p, b))

= ε-closure (q)

= {p, q}

c-successor of {p} = ε-closure (δ(p, c))

= ε-closure (r) = {p, q, r}

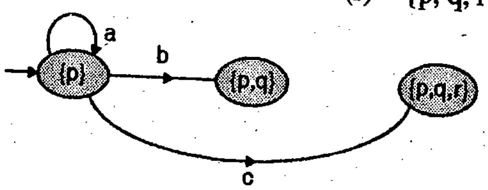


Fig. Ex. 2.6.30(a)

Step 2 : Two new subsets {p, q} and {p, q, r} are generated. Their successors are calculated.

a-successor of {p, q} = ε-closure (δ(p, a) ∪ δ(q, a))

= ε-closure ({p} ∪ {q})

= ε-closure ({p, q})

= ε-closure (p) ∪ ε-closure (q)

= {p} ∪ {p, q} = {p, q}

b-successor of {p, q} = ε-closure (δ(p, b) ∪ δ(q, b))

= ε-closure ({q} ∪ {r})

= ε-closure (q) ∪ ε-closure (r)

= {p, q} ∪ {p, q, r} = {p, q, r}

c-successor {p, q} = ε-closure (δ(p, c) ∪ (q, c))

= ε-closure ({r} ∪ ϕ)

= {p, q, r}

a-successor of {p, q, r} = a-successor of {p, q} ∪ a-successor of {r}

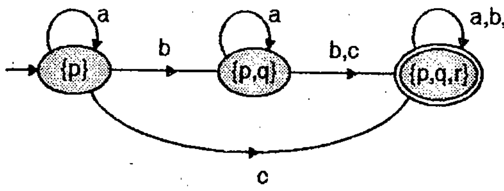
= {p, q} ∪ ε-closure (r)

= {p, q} ∪ {p, q, r} = {p, q, r}

b-successor of $\{p, q, r\}$ = b-successor of $\{p, q\} \cup$ b-successor of $\{r\}$
$= \{p, q, r\}$

c-successor of $\{p, q, r\}$ = c-successor $\{p, q\} \cup$ c-successor of $\{r\}$
$= \{p, q, r\} \cup \varepsilon$-closure (p)
$= \{p, q, r\}$

No new subsets are generated. Each subset containing r (final state in given (ε-NFA) is marked as final state.



(b) State transition diagram of final DFA

	a	b	c
$\rightarrow\{p\}$	$\{p\}$	$\{p, q\}$	$\{p, q, r\}$
$\{p, q\}$	$\{p, q\}$	$\{p, q, r\}$	$\{p, q, r\}$
$\{p, q, r\}^*$	$\{p, q, r\}$	$\{p, q, r\}$	$\{p, q, r\}$

(c) State transition table of final DFA

Fig. Ex. 2.6.30

Example 2.6.31 : Consider the following ε-NFA.

	ε	a	b	c
$\rightarrow$p	{q, r}	ϕ	{q}	{r}
q	ϕ	{p}	{r}	{p, q}
r*	ϕ	ϕ	ϕ	ϕ

(a) Compute the ε-closure of each state.

(b) Convert the automation to 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.

ε-closure (p) $= \{p, q, r\}$

Successor of $\{p, q, r\}$ are given by :

a-successor of $\{p, q, r\}$ = ε-closure ($\delta(p, q, r\}$, a))
$= \varepsilon$-closure $(\delta(p, a) \cup \delta(q, a) \cup \delta(r, a))$
$= \varepsilon$-closure $(\phi \cup \{p\} \cup \phi)$
$= \varepsilon$-closure $(\{p\})$
$= \{p, q, r\}$

b-successor of $\{p, q, r\}$ = ε-closure $[\delta(\{p, q, r,\}, b)]$
$= \varepsilon$-closure $(\delta(p, b) \cup \delta(q, b) \cup \delta(r, b))$
$= \varepsilon$-closure $(\{q\} \cup \{r\} \cup \phi)$
$= \varepsilon$-closure $(\{q, r\})$
$= \varepsilon$-closure (q) $\cup$ ε-closure (r)
$= \{q\} \cup \{r\}$
$= \{q, r\}$

c-successors of $\{p, q, r\}$ = ε-closure $(\delta(\{p, q, r\}, c))$
$= \varepsilon$-closure $(\delta(p, c) \cup (q, c) \cup \delta(r, c))$
$= \varepsilon$-closure $(\{r\} \cup \{p, q\} \cup \phi)$
$= \varepsilon$-closure $(\{p, q, r\})$
$= \varepsilon$-closure (p) $\cup$ ε-closure (q) $\cup$ ε-closure (r)
$= \{p, q, r\} \cup \{q\} \cup \{r\}$
$= \{p, q, r\}$

Step 2 : A new subset $\{q, r\}$ is generated and its successors are calculated.

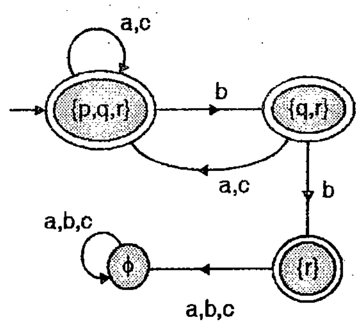
a successor of $\{q, r\}$ = ε-closure $(\delta\{q, r\}, a))$
$= \varepsilon$-closure $(\delta(q, a) \cup \delta(r, a))$
$= \varepsilon$-closure $(\{p\} \cup \phi)$
$= \varepsilon$-closure (p)
$= \{p, q, r\}$

b-successors of $\{q, r\}$ = ε-closure $(\delta(\{q, r\}, b))$
$= \varepsilon$-closure $(\delta(q, b) \cup \delta(r, b))$
$= \varepsilon$-closure $(\{r\} \cup \phi)$
$= \varepsilon$-closure $(\{r\}) = \{r\}$

c-successor of $\{q, r\}$ = ε-closure $(\delta(\{q, r\}, c))$
$= \varepsilon$-closure $(\delta(q, c) \cup \delta(r, c))$
$= \varepsilon$-closure $(\{p, q\} \cup \phi)$
$= \{p, q, r\}$

Step 3 : A new subset $\{r\}$ is generated successor of $\{r\}$ are ϕ.

Every subset containing r is marked as a final state. Final DFA is given Fig. Ex. 2.6.31.



(a) State diagram

	a	b	c
→{p, q, r}*	{p, q, r}	{q, r}	{p, q, r}
{q, r}*	{p, q, r}	{r}	{p, q, r}
{r}*	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ

(b) State table

Fig. Ex. 2.6.31 : Final DFA for example 2.6.31

Example 2.6.32

Design ε-NFA for the following languages. Try to use ε-transitions to simplify the design

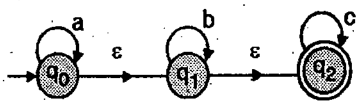
(a) The set of strings consisting of zero or more a's followed by zero or more b's, followed by zero or more c's.

(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
$\rightarrow q_0$	$\{q_1\}$	$\{q_0\}$	ϕ	ϕ
q_1	$\{q_2\}$	ϕ	$\{q_1\}$	ϕ
q_2^*	ϕ	ϕ	ϕ	$\{q_2\}$

Meaning of various states :

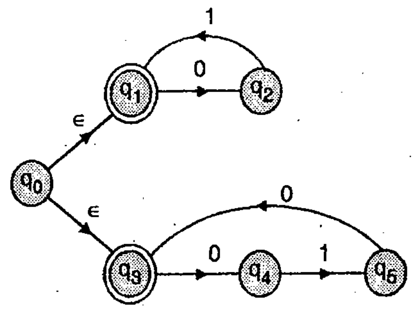
$q_0 \rightarrow q_0$ will generate zero or more a's

$q_1 \rightarrow q_1$ will generate zero or more b's

$q_2 \rightarrow q_2$ will generate zero or more c's

Fig. Ex. 2.6.32(a) : ε-NFA for example 2.6.32(a)

(b) The set of strings that consist of either 01 repeated zero or more times or 010 repeated zero or more times.



(i) 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

(ii) State table

Fig. Ex. 2.6.32(b) : ε-NFA for example 2.6.32(b)

- Path $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_1 \ldots\ q_2 \rightarrow q_1$ is for generation of zero or more representation of 01.
- Path $q_0 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \ldots \rightarrow q_3$ 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.

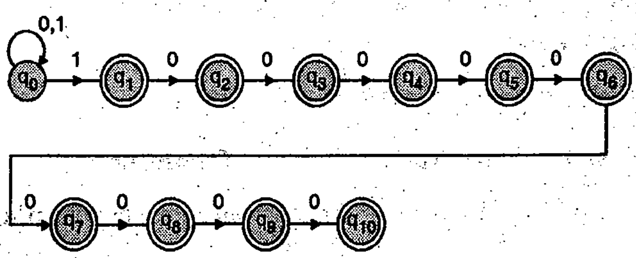


Fig. Ex. 2.6.32(c) : NFA for example 2.6.32(c)

- State q_1 is for accepting strings with last character is 1.
- Similarly, states q_2 to q_{10} are for accepting strings whose second position to 10^{th} position from the end contain 1.

Example 2.6.33

Consider the following NFA with $\in$-transitions. Find an equivalent DFA.

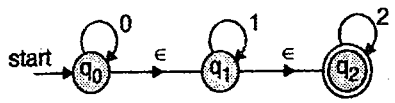


Fig. Ex. 2.6.33

Solution :

DFA

Step 1 : Finding $\in$-closure of each state :

State	$\in$-closure
q_0	$\{q_0, q_1, q_2\}$
q_1	$\{q_1, q_2\}$
q_2	$\{q_2\}$

Step 2 : Construction of DFA using direct approach.

1. $\in$-closure of q_0 is taken as the starting state of the DFA. Its transitions are written :

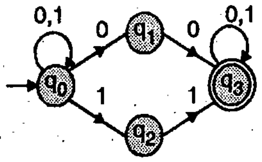


Fig. Ex. 2.6.33(a)

2. Transitions for $\{q_1, q_2\}$ and $\{q_2\}$ are written.

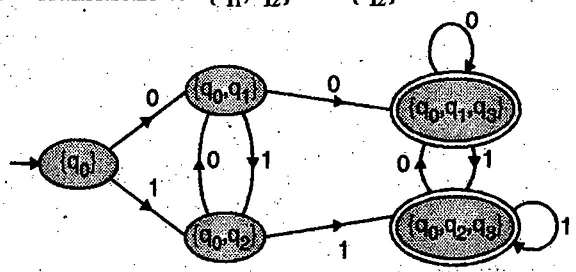


Fig. Ex. 2.6.33(b)

3. Every state (subset) containing the state q_2 is declared as final state.

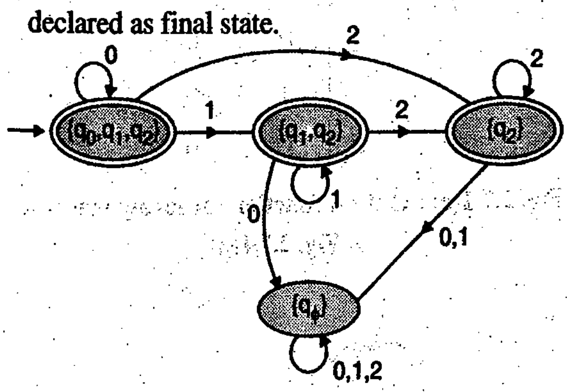


Fig. Ex. 2.6.33(c)

Example 2.6.34 SPPU - Dec. 12, 6 Marks

Construct NFA and then DFA with reduced states, equivalent to $(0 + 1)^* (00 + 11) (0 + 1)^*$

Solution :

Construction of NFA :

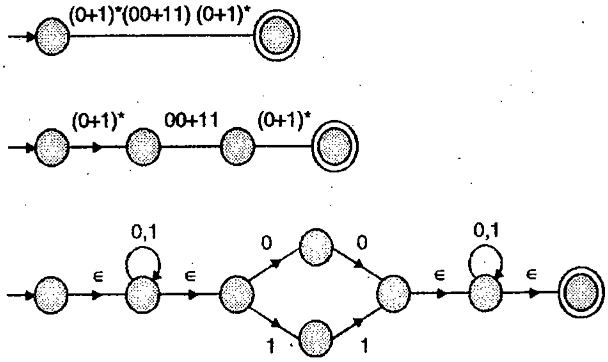


Fig. Ex. 2.6.34

Unnecessary $\in$-moves are recovered.

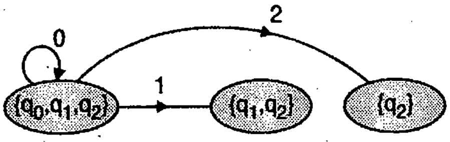
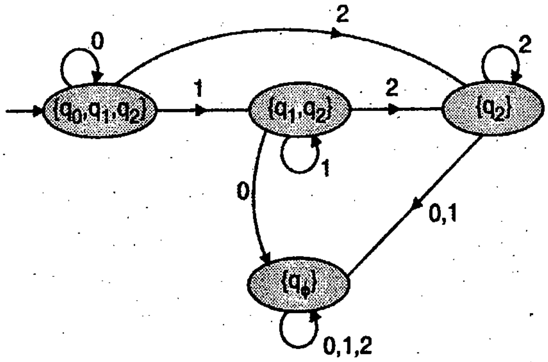


Fig. Ex. 2.6.34(a)

Construction of DFA using the direct method.



The two states $\{q_0, q_1, q_3\}$ and $\{q_0, q_2, q_3\}$ are not separable as they have similar outgoing behaviour. The minimum state DFA is given below. The state $\{q_0, q_3\}$ represents the state $\{q_0, q_1, q_3\}/\{q_0, q_2, q_3\}$

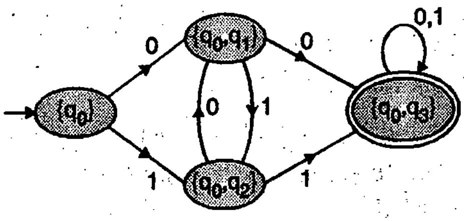


Fig. Ex. 2.6.34(c)

2.6.5 Difference between NFA and DFA

SPPU : May 12, May 14

University Question

Q. Differentiate between NFA and DFA. **(May 2012, May 2014, 4 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.

Syllabus Topic : FA with Output - Moore and Mealy Machines -Definition, Models, Inter-Conversion

2.7 Finite Automata as Output Devices

Finite automata that we have discussed so far has 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).

$\delta : \Sigma \times Q \rightarrow Q$

$\lambda : \Sigma \times Q \rightarrow O$ [for Mealy machine]

$\lambda : Q \rightarrow O$ [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.

   $\lambda : Q \rightarrow O$

2.7.1 A Sample Mealy Machine

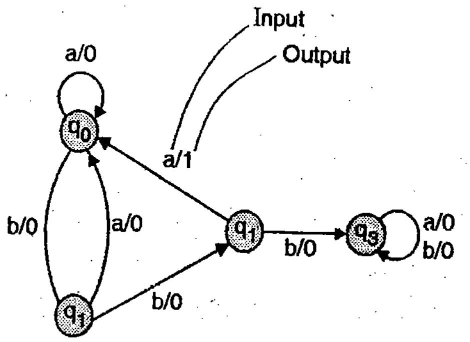


Fig. 2.7.1(a) : State diagram of a Mealy machine

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. 2.7.1(b) : State transition function for Mealy machine of Fig. 2.7.1(a)

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. 2.7.1(c) : Output function for mealy machine of Fig. 2.7.1(a)

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. 2.7.1(d) : State table depicting both transition and output behaviour of mealy machine of Fig. 2.7.1(a)

- 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. 2.7.1(a) 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. 2.7.1(b) and 2.7.1(c); or they can be combined together as in Fig. 2.7.1(d).

2.7.2 Formal Definition of a Mealy Machine

SPPU - Dec. 13. May 14

University Question

Q. Define Mealy Machine. (Dec. 2013, May 2014, 2 Marks)

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.

2.7.3 A Sample Moore Machine

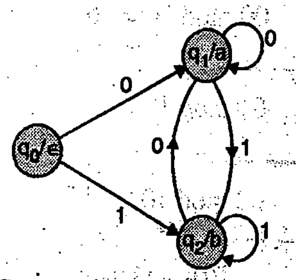


Fig. 2.7.2(a) : State diagram of a Moore machine

State transition function (δ) : Output function (O)

State	Output
$\rightarrow q_0$	ε
q_1	a
q_2	b

Fig. 2.7.2(b) : State transition function for Moore machine of Fig. 2.7.2(a)

	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_2
q_2	q_1	q_2

Fig. 2.7.2(c) : Output function for Moore machine of Fig. 2.7.2(a)

State table for both δ and λ :

State	Input 0	Input 1	Output
$\rightarrow q_0$	q_1	q_2	ε
q_1	q_1	q_2	a
q_2	q_1	q_2	b

Fig. 2.7.2(d) : State table depicting both transition and output behaviour of Moore machine of Fig. 2.7.2(a)

- Output in a Moore machine is associated with a state and not with transition.

2.7.4 Formal Definition of a Moore Machine

SPPU - May 14

University Question

Q. Define the Moore machine (May 2014, 2 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 2.7.1

Give Mealy and Moore machine for the following :

From input Σ^*, where Σ = (0, 1, 2) print the residue modulo 5 of the input treated as ternary (base 3).

Solution :

(a) Mealy machine :

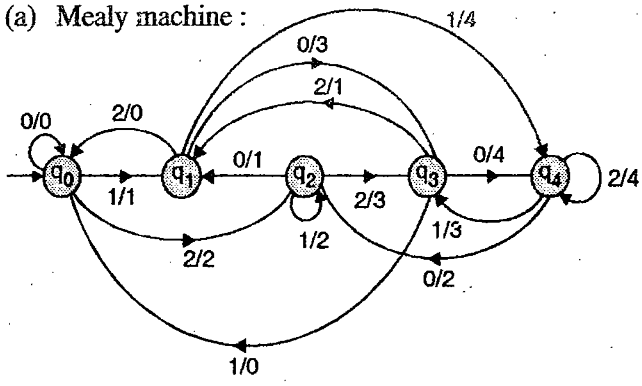


Fig. Ex. 2.7.1(a) : Mealy machine

Meaning of various states is given below :

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 $4 = (11)_3$

In decimal system   In ternary system

(b) Moore machine :

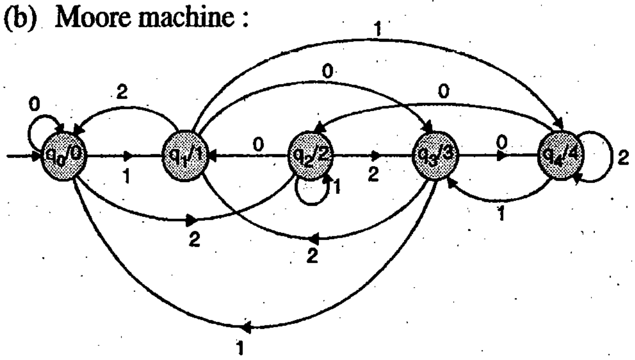


Fig. Ex. 2.7.1(b) : Moore machine

Example 2.7.2

Draw the transition matrix and transition graph for a 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.

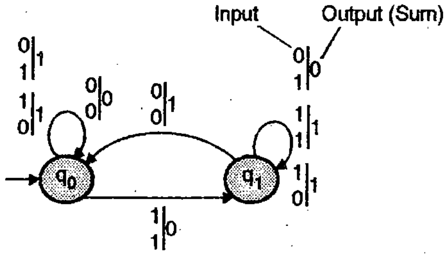


Fig. Ex. 2.7.2(a) : Transition graph

	$\begin{pmatrix}0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	
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

→ Next state

Fig. Ex. 2.7.2(b) : Transition matrix

Example 2.7.3

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	00 mod 10 = 0	q_0
	1	01 mod 10 = 1	q_1
	2	02 mod 10 = 2	q_2

$(10)_3 = (3)_{10}$

Current state (current remainder)	Next input		Next remainder (next state)
1 (q_1)	0	10 mod 10 = 0	q_0
	1	11 mod 10 = 1	q_1
	2	12 mod 10 = 2	q_2
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.

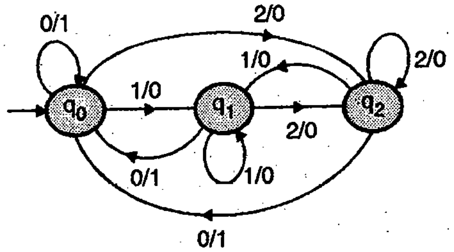


Fig. Ex. 2.7.3(a) : State transition diagram

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	$q_0/1$	$q_1/0$	$q_2/0$

Fig. Ex. 2.7.3(b) : State transition table

	0	1	2
$\rightarrow q_0$	1	0	0
q_1	1	0	0
q_2	1	0	0

Fig. Ex. 2.7.3(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

Fig. Ex. 2.7.3(d) : STF

Example 2.7.4

Design a Mealy machine that gives an output of 1 if the input string ends in bab.

Solution :

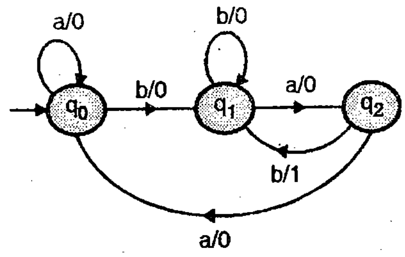


Fig. Ex. 2.7.4(a) : State diagram

	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

Fig. Ex. 2.7.4(b) : State table

Meaning of various states :

q_0 – Start state

q_1 – Previous character is b

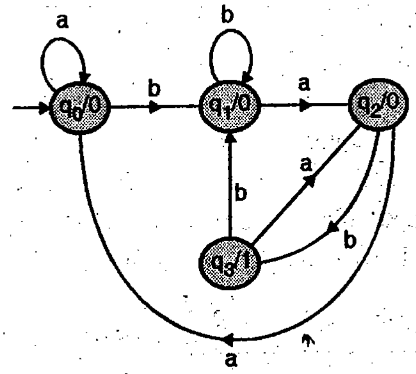
q_2 – Preceding two character are ba

A transition from q_2 to q_1 will make the preceding three characters as bab and hence the output 1.

Example 2.7.5

Design a Moore machine that gives an output of 1 if the input string ends in bab.

Solution :



(a) State diagram

Present state	Input a	b	Output
$\rightarrow q_0$	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. 2.7.5 : Moore machine for the example 2.7.5

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 b

A transition from q_2 to q_3 will make the preceding three characters as bab hence the output 1.

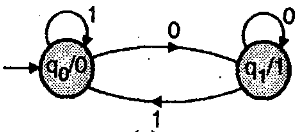
Example 2.7.6 SPPU - May 15. 6 Marks

(a) Design a Moore machine for the 1's complement of binary number.

(b) Design a mealy machine to find out 2's complement of a given binary number.

Solution :

(a) :



(a)

	0	1	Output
$\rightarrow q_0$	q_1	q_0	0
q_1	q_1	q_0	1

(b)

Fig. Ex. 2.7.6

– 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.

(b) 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 :

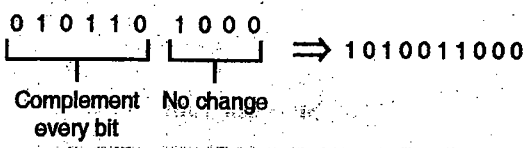


Fig. Ex. 2.7.6(c)

The required mealy machine is given below.

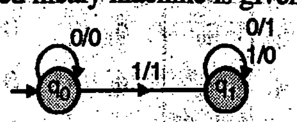
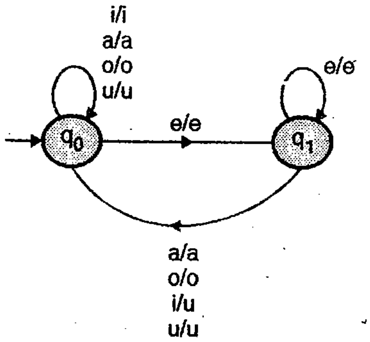


Fig. Ex. 2.7.6(d)

Example 2.7.7 SPPU - May 13. 8 Marks

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. 2.7.7

– 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 2.7.8 SPPU - May 14. 6 Marks

Give the Mealy and Moore machine for the following processes. "For input from $(0 + 1)^*$, if inputs ends in 101, output x; if input ends in 110, output y; otherwise output z".

Solution :

(a) Design of Mealy machine

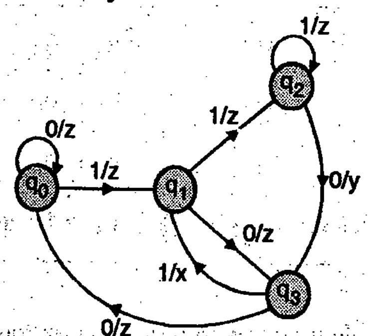


Fig. Ex. 2.7.8 : State diagram of Mealy machine

- 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

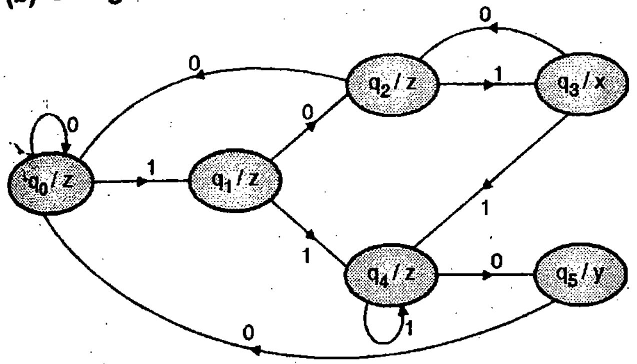


Fig. Ex. 2.7.8(a) : State diagram of the Moore machine

Example 2.7.9

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

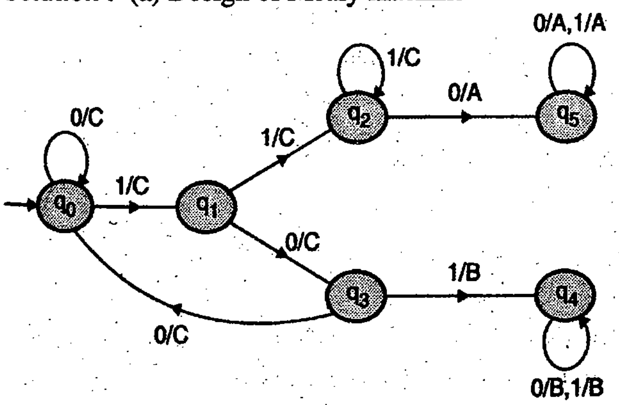


Fig. Ex. 2.7.9(a) : State diagram of the Mealy machine

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.

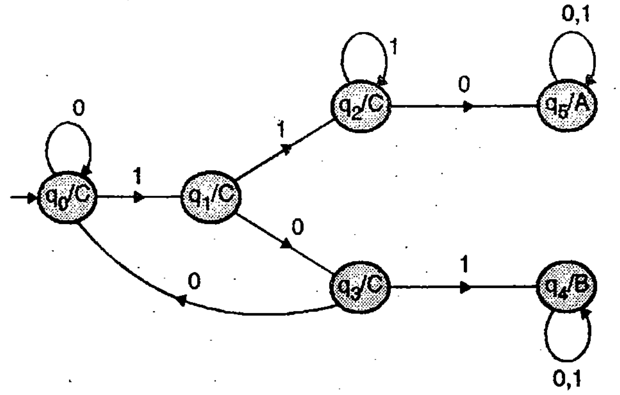


Fig. Ex. 2.7.9(b) : State diagram of the Moore machine

Example 2.7.10

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.

where, $A = a_n, a_{n-1} \ldots a_1$

$B = b_n, b_{n-1} \ldots b_1$

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.

$A > B$ if $a_i > b_i$

$A < B$ if $a_i < b_i$

Machine generates the following output :

$A > B$ ... +

$A < B$ ... –

$A = B$ ... 0

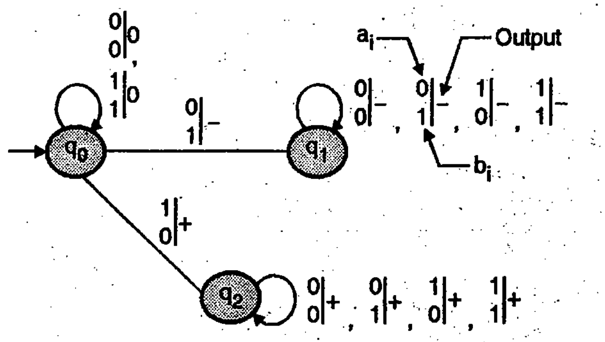


Fig. Ex. 2.7.10 : State diagram of the Mealy machine

Example 2.7.11 SPPU - Dec. 12. 6 Marks

Construct a Moore machine equivalent to the Mealy machine M given as :

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 : The transition diagram for the referenced Moore machine is given below.

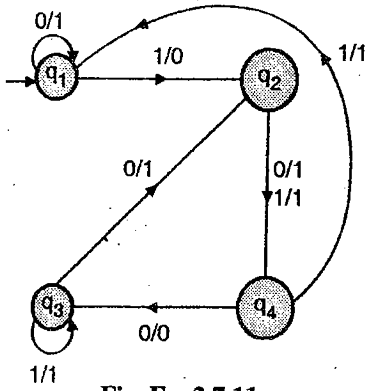


Fig. Ex. 2.7.11

Step 2 : Splitting of states on the basis of incoming states is carried out starting state is divided into two states.

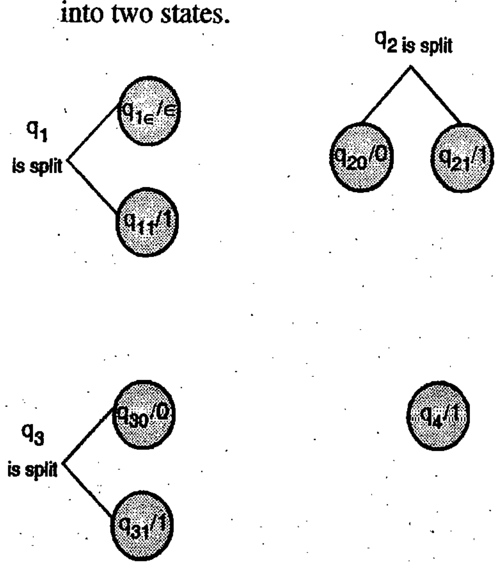


Fig. Ex. 2.7.11(a)

Step 3 : Writing transitions on the basis of outgoing transitions.

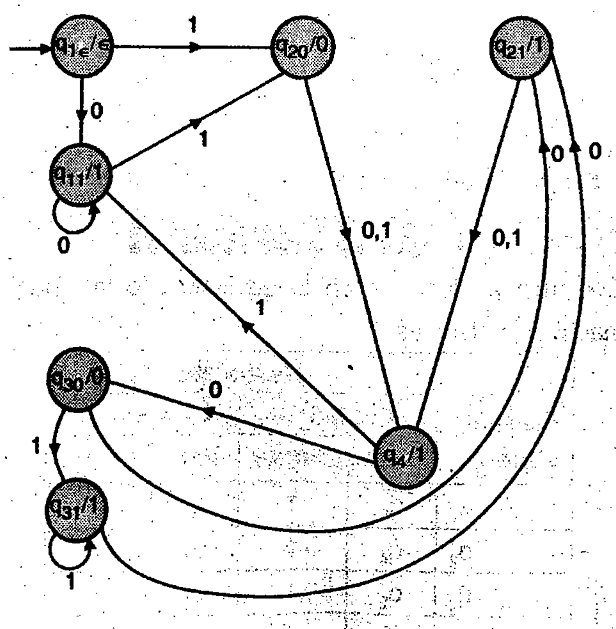


Fig. Ex. 2.7.11(b)

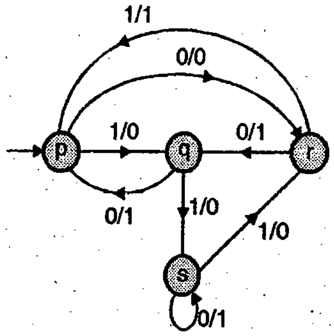
The machine, as given above is the mealy machine.

2.7.5 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. 2.7.3(a).

Present state	(Next state, output)	
	Input 0	Input 1
→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. 2.7.3 : 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.

(i) From q to p on input 0, giving an output of 1.

(ii) 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.

(i) From p to q on input 1, giving an output of 0.

(ii) 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.

$$q \overset{\text{split}}{\Longrightarrow} \begin{array}{l} q_0/0 - \text{output } 0 \\ q_1/1 - \text{output } 1 \end{array}$$

- There are two incoming lines into the state r.
  (i) From p to r on input 0, giving an output of 0.
  (ii) 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.
  (i) From q to s an input 1, giving an output of 0.
  (ii) 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.

$$s \overset{\text{split}}{\Longrightarrow} \begin{array}{l} s_0/0 - \text{output } 0 \\ s_1/1 - \text{output } 1 \end{array}$$

Step 2 : Drawing of an equivalent Moore machine.

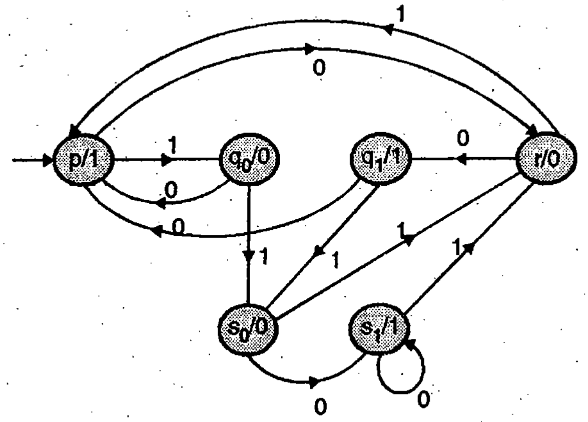


Fig. 2.7.3(c) : State diagram of Moore machine

- 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. 2.7.3(d).

Present state	Next state 0	Next state 1	Output
$\rightarrow p$	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. 2.7.3(d) : State transition table

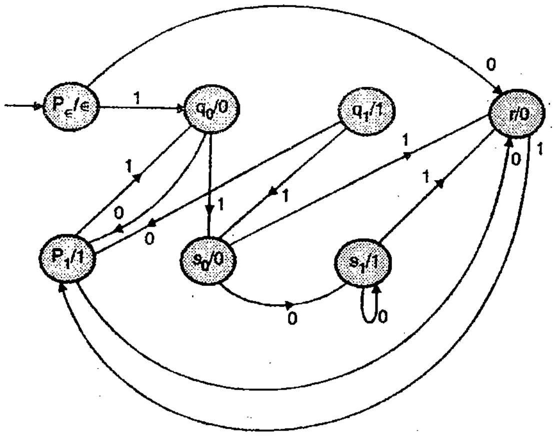
Step 3 : Moore machine of Fig. 2.7.3(c) produces an output 1 without any input. Thus, the Moore machine gives an output 1 on a zero length sequence. But the Mealy machine will produce no output for a zero length sequence.

To handle the above problem, we must split the start state p into two states :

p_ε – Giving no output

p_1 – Giving an output 1.

Final Moore machine is drawn in Fig. 2.7.4.



(a) State diagram

Present state	Next state		Output
	0	1	
$\rightarrow p_\varepsilon$	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

(b) State table

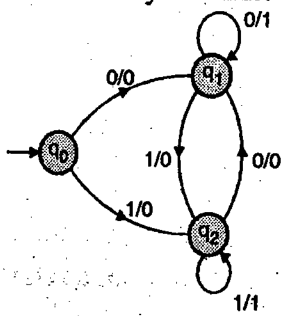
Fig. 2.7.4 : Final Moore machine

Example 2.7.12 SPPU - May 14. 4 Marks

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_1, 0	q_2, 1

(b) State transition table of the Mealy machine

Fig. Ex. 2.7.12

The Mealy machine is shown in the Fig. Ex. 2.7.12. 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.

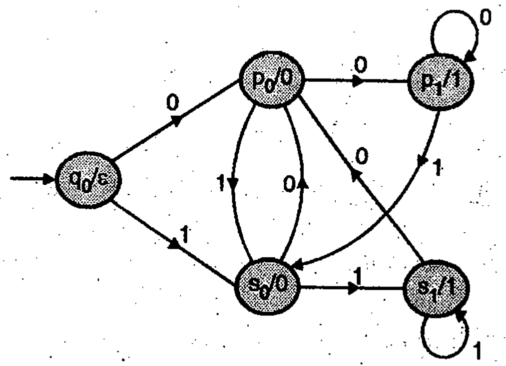
(b) q_1 to q_1 on input 0, with output 1.

(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.



(c) State transition diagram of the 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

(d) State transition table of the Moore machine

Fig. Ex. 2.7.12

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.
$\delta(q_2, 0) = q_1, 0$	$\delta(s_0, 0) = p_0$ $\delta(s, 0) = p_0$ Transition is duplicated for both s_0 and s_1. p_0 is associated with output 0.
$\delta(q_2, 1) = q_2, 1$	$\delta(s_0, 1) = s_1$ $\delta(s_1, 1) = s_1$ Transition is duplicated for both s_0 and s_1. s_1 is associated with output 1.

Example 2.7.13 SPPU - Dec. 13, 6 Marks

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 so 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.

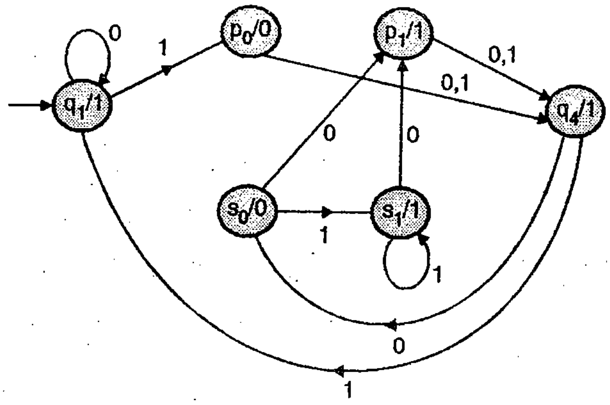


Fig. Ex. 2.7.13(a) : Moore machine

Step 3 : Moore machine of Fig. Ex. 2.7.13(a) produces an output of 1 without any input. But the Mealy machine given in example 2.7.9, 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_ε – Giving no output

r_1 – Giving an output 1.

Final Moore machine is drawn in Fig. Ex. 2.7.13(b) and 2.7.13(c).

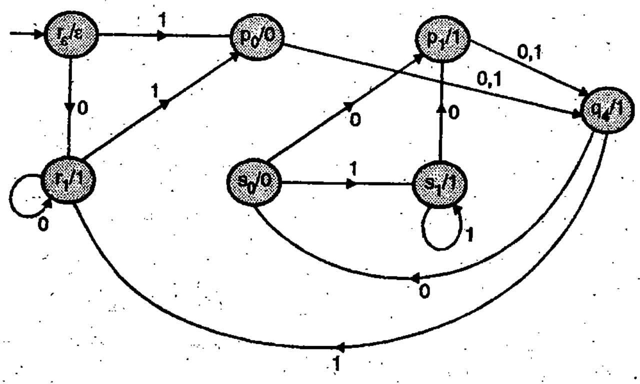


Fig. Ex. 2.7.13(b) : State transition diagram for an equivalent Moore machine

Current state	0	1	Output
$\rightarrow r_\varepsilon$	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. 2.7.13(c) : State transition table for an equivalent Moore machine

Example 2.7.14

Convert the following Mealy machine to Moore machine.

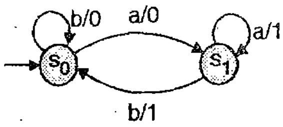


Fig. Ex. 2.7.14

Solution :

Step 1 : Splitting of states.

(i) Incoming lines into s_0 give both 0 and 1 as output. In addition, s_0 in an initial state and hence an ε-output condition should also be considered. s_0 is split into three states :

p_ε with output ε,

p_0 with output 0,

p_1 with output 1.

(ii) 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.

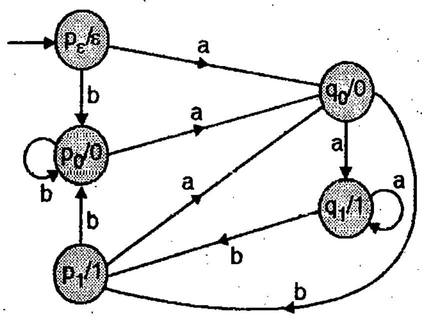


Fig. Ex. 2.7.14(a) : Moore machine for example 2.7.14

– Transitions for s_0 are duplicated for p_ε, p_0 and p_1.

– Transitions for s_1 are duplicated for q_0 and q_1.

Example 2.7.15

Convert the following Mealy into equivalent Moore machine.

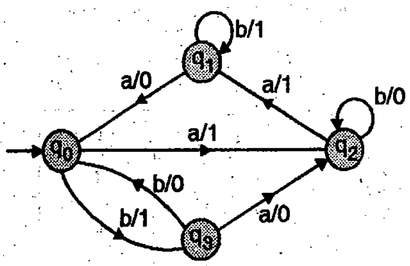


Fig. Ex. 2.7.15

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_1 .... (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.

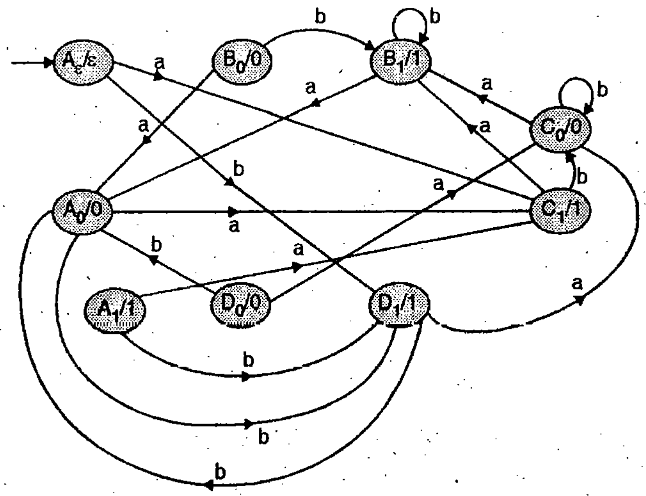


Fig. Ex. 2.7.15(a) : Mealy machine for example 2.7.15

Example 2.7.16 SPPU - Dec. 12, 6 Marks

Construct Mealy machine equivalent to Moore machine given as ;

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

Solution :

Step 1 : A trivial mealy machine is constructed by moving output associate with states to their corresponding incoming transitions

Present State	NS/output	
	a = 0	a = 1
q_0	$q_1/0$	$q_2/1$
q_1	$q_3/1$	$q_2/1$
q_2	$q_2/1$	$q_1/0$
q_3	$q_0/1$	$q_3/1$

Step 2 : Minimization of the trivial mealy machine

1. Initial grouping of states based on outputs (q_0) (q_1, q_3) (q_2)
2. The state q_1 is separable from the state q_3 on the input '0'. Therefore, the trivial mealy machine is the minimum state mealy machine

2.8 Minimization of a Mealy Machine

Minimization of a Mealy machine is almost similar to minimization of a DFA (Discussed in Section 2.5).

- 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. 2.8.1 : Mealy machine being considered for minimization

Minimization process is being explained step wise for the Mealy machine given in Fig. 2.8.1.

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.

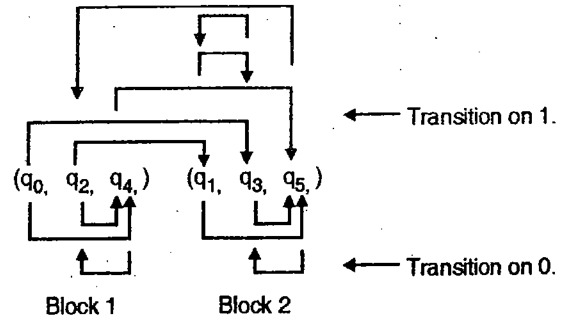


Fig. 2.8.1(a)

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.

$P_2 = (q_0, q_2, q_4)\quad (q_1, q_3)\quad (q_5)$

Block 1 Block 2 Block 3

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. 2.8.1(b) and 2.8.1(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. 2.8.1(b) : State table for minimum state Mealy machine

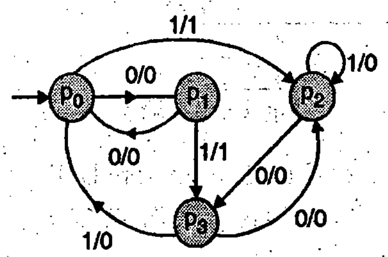


Fig. 2.8.1(c) : State diagram for minimum state Mealy machine

Example 2.8.1

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.

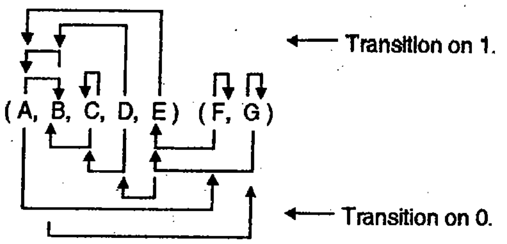


Fig. Ex. 2.8.1(a)

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)

∴ Behaviour 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.

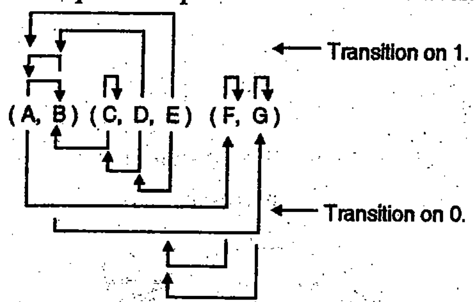


Fig. Ex. 2.8.1(b)

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.

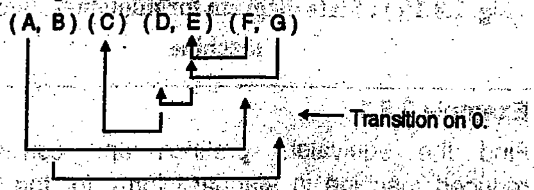


Fig. Ex. 2.8.1(c)

On input 0, D is mapped to block (C)

On input 0, E is mapped to block (D, E)

∴ Behaviour of D is different from E.

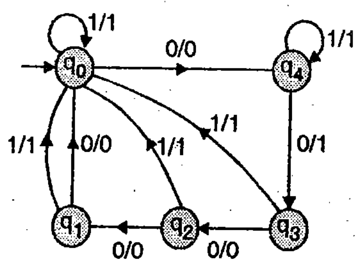
$$P_4 = (A, B)\ (C)\ (D)\ (E)\ (F, G)$$

Further division is not possible.

The minimum state Mealy machine is drawn in Fig. Ex. 2.8.1(e).

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. 2.8.1 : Minimum state Mealy machine for example 2.8.1

2.8.1 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. 2.8.2(a)

Present state	Next state		Output
	0	1	
→p	s	q	0
q	q	r	1
r	r	s	0
s	s	p	0

Fig. 2.8.2(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	
→p	s	q	–
q	q	r	1
r	r	s	0
s	s	p, 0	0

Fig. 2.8.2(b)

(ii) Moving the output associated with q, which is 1 to transitions entering into state q, we get :

	0	1	
→p	s	q, 1	–
q	q, 1	r	–
r	r	s	0
s	s	p, 0	0

Fig. 2.8.2(c)

(iii) Moving the output associated with r, which is 0 to transitions entering into the state r, we get :

	0	1	
→p	s	q, 1	–
q	q, 1	r, 0	–
r	r, 0	s	–
s	s	p, 0	0

Fig. 2.8.2(d)

(iv) Moving the output associated with s, which is 0 to transitions entering the state s, we get :

	0	1
→p	s, 0	q, 1
q	q, 1	r, 0
r	r, 0	s, 0
s	s, 0	p, 0

Fig. 2.8.2(e) : A trivial Mealy machine

Step 2 : The trivial Mealy machine obtained in Fig. 2.8.2(e) can not be minimized further.

Final Mealy machine is given in Fig. 2.8.2(f) and 2.8.2(g).

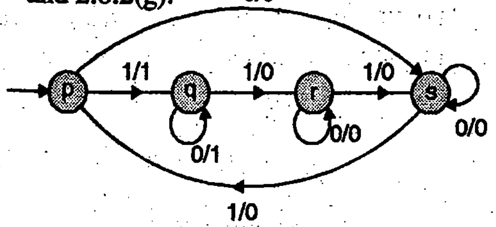


Fig. 2.8.2(g) : Transition diagram for final Mealy machine

	0	1
→p	s, 0	q, 1
q	q, 1	r, 0
r	r, 0	s, 0
s	s, 0	p, 0

Fig. 2.8.2(f) : Transition table for final Mealy machine

Example 2.8.2

Change the given Moore machine into Mealy machine.

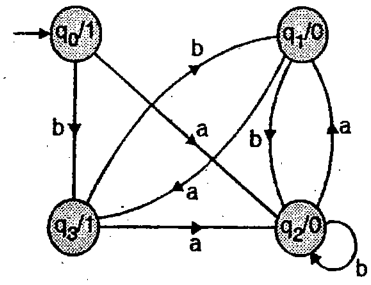


Fig. Ex. 2.8.2

Solution :

Step 1 : Construction of a trivial Mealy machine by moving output associated with a state to transitions entering that state, we get :

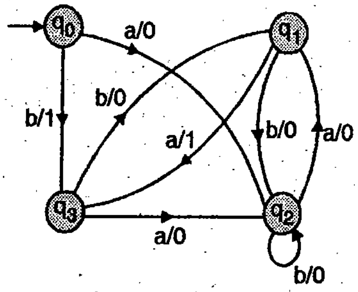


Fig. Ex. 2.8.2(a)

	a	b
$\rightarrow q_0$	q_2, 0	q_3, 1
q_1	q_3, 1	q_2, 0
q_2	q_1, 0	q_2, 0
q_3	q_2, 0	q_1, 0

Fig. Ex. 2.8.2(b)

Step 2 : Minimization of Mealy machine obtained in Fig. Ex. 2.8.2(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.

$\therefore$ Behaviour of q_2 is different from q_3.

$$P_2 = (q_0)(q_1)(q_2)(q_3)$$

Mealy machine given in Fig. Ex. 2.8.2(a) is the final Mealy machine.

Example 2.8.3

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

Fig. Ex. 2.8.3

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. 2.8.3(a) : A trivial Mealy machine

Step 2 : Minimization of the trivial Mealy machine obtained in Fig. Ex. 2.8.3(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.

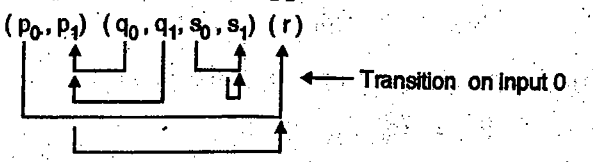


Fig. Ex. 2.8.3(b)

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)

$\therefore$ Behaviour 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. 2.8.3(c) and Fig. Ex. 2.8.3(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

Fig. Ex. 2.8.3(c) : State transition table of Mealy machine

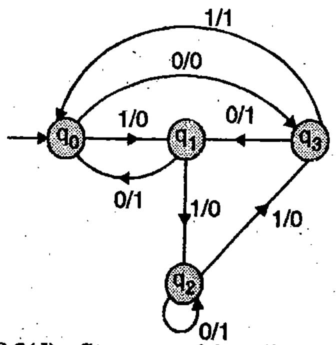


Fig. Ex. 2.8.3(d) : State transition diagram of Mealy machine

Example 2.8.4 SPPU - May 12, 8 Marks

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
$\rightarrow q_1$	$q_1,0$	$q_2,0$
$(q_2, q_3) = q_2$	$q_1,0$	$q_2,1$

Syllabus Topic : FSM for Vending Machine

2.9 FSM for Vending Machine

An schematic of a typical vending machine is shown in the Fig. 2.9.1.

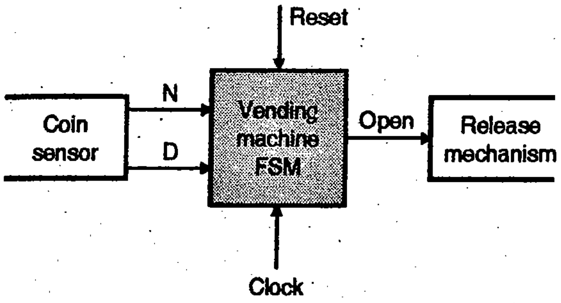


Fig. 2.9.1 : A typical vending machine

Nowadays, vending machines are well known among developed countries. A vending machine can be designed using a finite state machine model with auto-billing. FSM model reduces the hardware. A typical vending machine her the following steps :

1. User selection of a product.
2. Waiting for money insertion
3. Product delivery and servicing.

Entire thing can be modelling around a mealy machine.

A vending machine can count the value of inserted coins and take a decision regarding when to dispense a product.

Syllabus Topic : FSM for Spell Checker

2.10 FSM for Spell Checker

A spell checker is a design feature of a software proper to verify the spelling of words in a document. It helps user in ensuring correct spelling checkers are most often implemented as a feature of a larger application, such as a word processor. A set of words can be encoded a an FSM. FSM can detect whether a word is a valid word and it can also flash a list of probable words. The lexical analysis phase of a complier uses a similar concept.

❑❑❑

CHAPTER 3 Regular Expressions (RE)

Syllabus

Introduction, Operators of RE, Building RE, Precedence of operators, Algebraic laws for RE, Conversions: NFA to DFA, RE to DFA Conversions: RE to DFA, Language accepted by FA, Definition of Regular Language, DFA to RE Conversions: State/loop elimination, Arden's theorem. Properties of Regular Languages: Pumping Lemma for Regular languages, Closure and Decision properties. Case Study: RE in text search and replace

Syllabus Topic : Introduction, Operators of RE

3.1 Introduction

SPPU - Dec. 12, May 13, Dec. 13

University Questions

Q. Define regular set and explain with example. **(Dec. 2012, 4 Marks)**

Q. Define the following with example : Regular expression. **(May 2013, 2 Marks)**

Q. Give formal definition of regular expression and regular set. **(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) . , concatenation operator

(3) * , star or closure operator.

An expression written using the set of operators (+, · , *) and describing a regular language is known as regular expression. Regular expressions for some basic automata are given in Fig. 3.1.1.

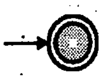
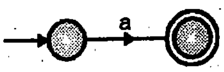
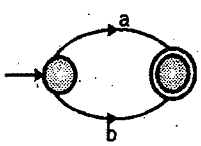
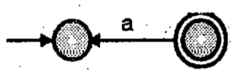
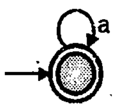
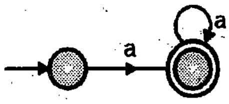
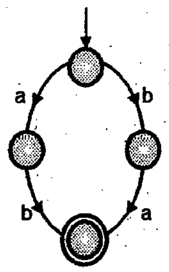
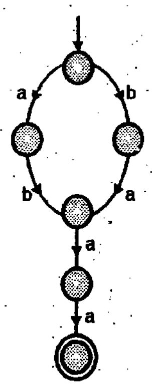
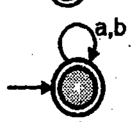
Automata	Language	Regular expression
	{ε}	R.E. = ε
	{a}	R.E. = a
	{a, b}	R.E = a + b
	{ab}	R.E. = a · b or simply ab
	φ	R.E. = φ
	{ε, a, aa, aaa, ...}	R.E. = a*
	{a, aa, aaa, ...}	R.E. = aa* or a+
	{ab, ba}	R.E. = ab + ba
	{abaa, baaa}	R.E. = (ab + ba) aa
	{ε, a, b, aa, ab, ba, bb, ...}	R.E. = (a + b)*

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.

(a) The set {1010}

(b) The set {10, 1010}

(c) The set {ε, 10, 01}

(d) The set {ε, 0, 00, 000, ...}

(e) The set {0, 00, 000, ...}

(f) The set of strings over alphabet {0, 1} starting with 0.

(g) The set of strings over alphabet {0, 1} ending in 1.

(h) The set of strings over alphabet (a, b} starting with a and ending in b.

(i) 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.

Syllabus Topic : Precedence of Operators

3.1.1 Precedence of Operators

Similar to algebraic operators, operators of regular expression follow a predefined precedence. Operators are associated with operands in a particular order. The precedence of operators for regular expressions is as follows :

1. The star operator has the highest precedence.
2. The concatenation or "dot" operators comes next in precedence.
3. Finally, union or "+" operators comes last in the precedence.

Syllabus Topic : RE to DFA Conversion, Language Accepted by FA

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^*

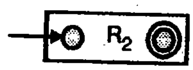
Then we can construct finite state automata (FA) for any regular expression.

3.2.1 Composite Finite State Automata

Let us assume the existence of FAs for regular expressions R_1 and R_2.

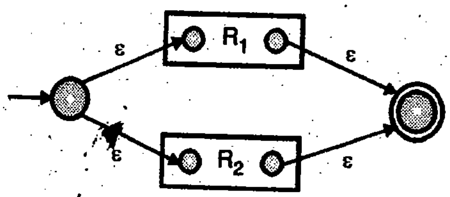
FA for R_1 is given by

FA for R_2 is given by

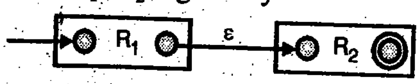


Each FA is assumed to have one final state.

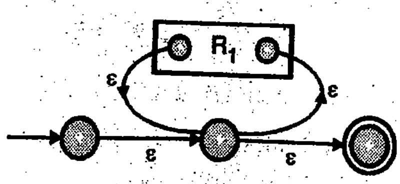
1. FA for $R_1 + R_2$ is given by



2. FA for $R_1 \cdot R_2$ is given by

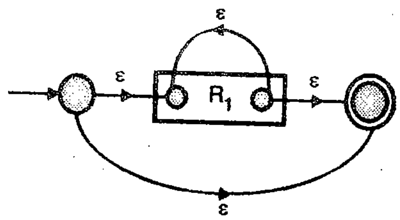


3. FA for R_1^* is given by



OR

FA for R_1^* can also be shown as



Finite automata for some primitive strings are given below

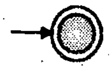
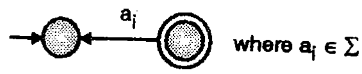
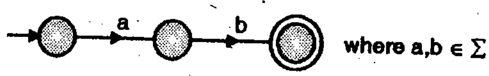
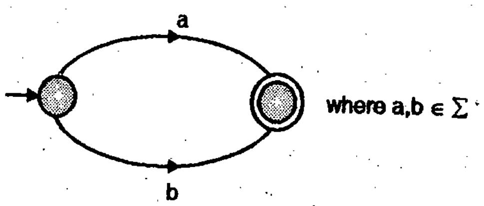
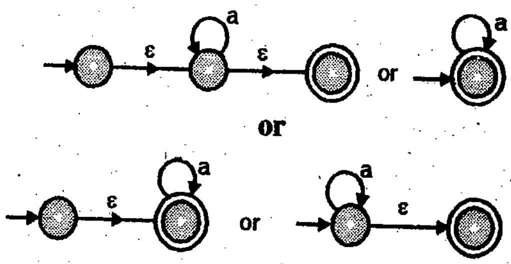
String	Finite Automata
ε	
φ	
a	
ab	
a + b	
a*	

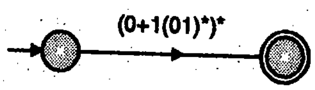
Fig. 3.2.1

Example 3.2.1

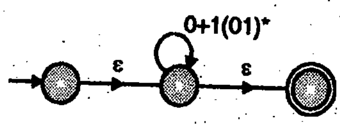
Construct a finite automata for the regular expression $R = (0 + 1(01)^*)^*$

Solution :

Step 1 :



Step 2 :



Step 3 :

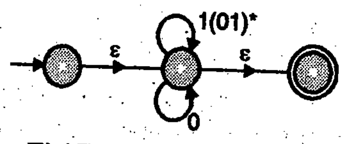
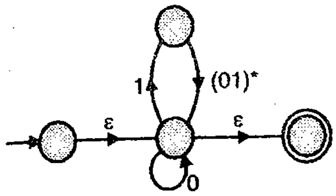


Fig. Ex. 3.2.1 Contd...

Step 4 :



Step 5 :

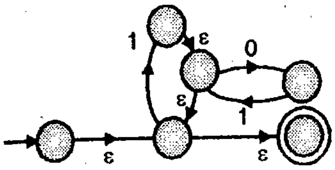
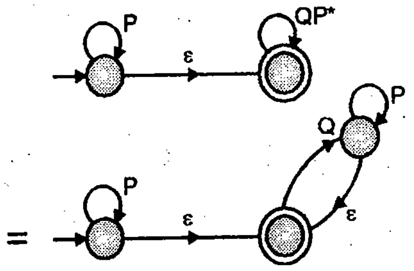


Fig. Ex. 3.2.1

Example 3.2.2 SPPU - May 14. 4 Marks

Show that $P^*(QP^*)^* = (P + Q)^*$

Solution : A finite automata recognizing $P^*(QP^*)^*$ is given by :



Naming the states, we get the Fig. Ex. 3.2.2(a).

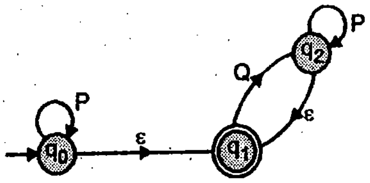


Fig. Ex. 3.2.2(a) : A finite automata for $P^*(QP^*)^*$

ε move from q_0 to q_1 can be removed by of

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.

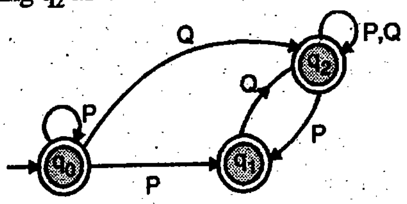


Fig. Ex. 3.2.2(b) : An equivalent finite automata without ε-moves

Since all the three states q_0, q_1 and q_2 are final states (in Fig. Ex. 3.2.2(b)), they can be merged into a single state.

$= (P + Q)^*$

Fig. Ex. 3.2.2(c) : 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 3.2.3 SPPU - Dec. 15. 2 Marks

Prove or disprove the following for a regular expression. $(r^*s^*)^* = (r + s)^*$

Solution :

Step 1 : ε-NFA for $(r^*s^*)^*$

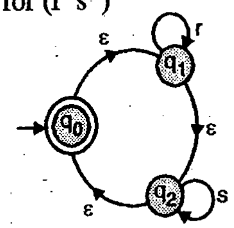


Fig. Ex. 3.2.3(a)

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 below

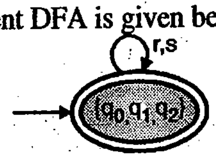


Fig. Ex. 3.2.3(b)

Step 3 : The regular expression for the DFA

$= (r + s)^*$

= R.H.S.

Example 3.2.4

Check the equivalence of the regular expressions.

(1) $(a^* bbb)^*a^*$ AND $a^* (bbb\, a^*)^*$

(2) $((a + bb)^* aa)^*$ AND $\in + (a + bb)^*.aa$

Solution :

(1) $(a^* bbb)^*a^*$ AND $a^* (bbb\, a^*)^*$

Substituting P for a^* and Q for bbb, we get,

$(PQ)^*P$ and $P(QP)^*$ are equivalent.

We, know form the basic lemma on R.E.'s that the above statement is true.

(2) $((a + bb)^* aa)^*$ AND $\in + (a + bb)^*.aa$

Substituting P for a + bb and Q for aa, we get, (P*Q)* and $\in$ + P*Q are equivalent.

They are not equivalent, as form straight reading it appears that (P*Q)*can have multiple occurrences of Q, whereas $\in$ + P*Q can have only one occurrence of Q.

Example 3.2.5 SPPU - Dec. 15 .4 Marks

Prove the formula : $(ab)^* \neq a^*b^*$

Solution :

The string abab $\in (ab)^*$

but abab $\notin a^*b^*$

and hence $(ab)^* \neq a^*b^*$

Example 3.4.6 SPPU - Dec. 13, 4 Marks

Prove or disprove each of the following about regular expression :

(1) $(RS + R)^*RS = (RR^*S)^*$

(2) $(R + S)^* = R^* + S^*$.

Solution :

(1) $(RS + R)^*RS = (RR^*S)^*$

$\in (RR^*S)^*$ but $\in \notin (RS + R)^* RS$

Hence disproved.

(2) $(R + S)^* = R^* + S^*$

$RS \in (R + S)^*$ but

$RS \notin R^* + S^*$

Hence disproved.

Syllabus Topic : Algebraic Laws for RE

3.2.2 Algebraic Laws for Regular Expressions

There are a number of laws for algebraic laws, including :

1. Assciativity and commutativity.
2. Identities and annihilators
4. The idempotent law
5. Laws involving closures.

1. Assocoativity and commutativity

- The commutative law for union says that the union of two regular languages can be taken in any order.
- For any two languages L and M,

$L + M = M + L$

- The associativity law holds for union of regular languages.

or, $(L + M) + N = L + (M + N)$

2. Identities and annihilators

$\in$ is the identity and ϕ is the annihilator. There are three laws for regular expressions involving $\in$ and ϕ :

1. $\phi + L = L + \phi = L$, ϕ is the identity for +
2. $\in \cdot L = L \cdot \in = L$, $\in$ is the identity for concatenation
3. $\phi \cdot L = L \cdot \phi = \phi$, ϕ is annihilator for concatenation

3. Distributive laws

The left distributive law of concatenation over union holds for regular languages.

$L (M + N) = LM + LN$

The right distributive law of concatenation over union holds for regular languages.

$(M + N) L = ML + NL$

4. The idempotent law

It says that the union of two identical expression can be replaced by one copy of the expression.

$L + L = L$

5. Laws involving closures

These laws include :

1. $(L^*)^* = L$, closure of the closure does not changes the language
2. $\phi^* = \in$
3. $\in^* = \in$
4. $L^* = LL^* = L^*L$
5. $L^* = L^* + \in$

Syllabus Topic : Building RE

3.3 Determination of Regular Expression

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.

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 $L(r_2) = \{\varepsilon, 0, 00, 000, \ldots\}$

If $r_3 = (0 + 1)^*$ then $L(r_3) = \{\varepsilon, 0, 1, 00, 01, 10, 11, ..\}$

3.3.2 Basic Properties of Regular Expressions

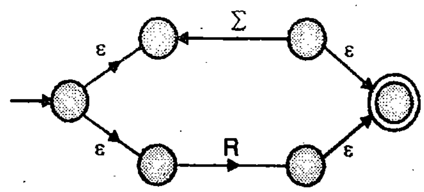
Some of the basic identities for regular expressions are given below. P, Q and R are regular expressions.

$$\phi + R = R \quad \ldots(3.3.1)$$
$$\phi \cdot R = R \cdot \phi = \phi \quad \ldots(3.3.2)$$
$$\varepsilon \cdot R = R \cdot \varepsilon = R \quad \ldots(3.3.3)$$
$$\varepsilon^* = \varepsilon \quad \ldots(3.3.4)$$
$$\phi^* = \varepsilon \quad \ldots(3.3.5)$$
$$R + R = R \quad \ldots(3.3.6)$$
$$PQ + PR = P(Q + R) \quad \ldots(3.3.7)$$
$$QP + RP = (Q + R)P \quad \ldots(3.3.8)$$
$$R^*R^* = R^* \quad \ldots(3.3.9)$$
$$RR^* = R^*R \quad \ldots(3.3.10)$$
$$(R^*)^* = R^* \quad \ldots(3.3.11)$$
$$\varepsilon + RR^* = R^* \quad \ldots(3.3.12)$$
$$(PQ)^*P = P(QP)^* \quad \ldots(3.3.13)$$
$$(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^* \quad \ldots(3.3.14)$$
$$(P^* Q)^* = \varepsilon + (P + Q)^*Q \quad \ldots(3.3.15)$$

Proof for basic properties

1. $\phi + R = R$

Machine for ϕ + R is given by

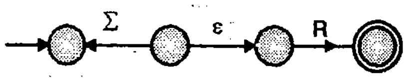


There is only one path from the start state to the final state, recognizing $\varepsilon \cdot R \cdot \varepsilon = R$.

Hence, $\phi + R = R$

2. $\phi \cdot R = \phi$

A composite machine for $\phi \cdot R$ is given by

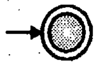


There is no way of reaching the final state from the start state.

$\therefore \ \phi \cdot R = \phi$

3. $\varepsilon \cdot R = R$

FA for ε is given by



FA for R is given by

Composite FA for $\varepsilon \cdot R$ is given by

This machines accepts R and only R.

$\therefore \ \varepsilon \cdot R = R.$

4. $\varepsilon + RR^* = R^*$

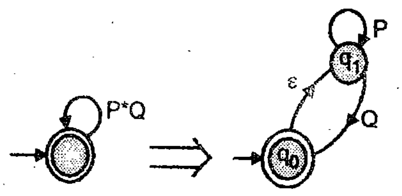
$$\begin{aligned} \text{L.H.S.} &= \varepsilon + RR^* \\ &= \varepsilon + R\{\varepsilon + R + RR + RRR + \ldots\} \\ &= \varepsilon + R + RR + RRR + \ldots \\ &= R^* = \text{R.H.S.} \end{aligned}$$

5. $(PQ)^*P = P(QP)^*$

$$\begin{aligned} \text{L.H.S.} &= (PQ)^* P \\ &= \{\varepsilon + PQ + (PQ)^2 + (PQ)^3 + \ldots\} P \\ &= \{\varepsilon + PQ + PQPQ + PQPQPQ + \ldots\}P \\ &= \{P + PQP + PQPQP + \ldots\} \\ &= P\{\varepsilon + QP + QPQP + \ldots\} \\ &= P(QP)^* = \text{R.H.S.} \end{aligned}$$

6. $(P^*Q)^* = \varepsilon + (P + Q)^*Q$

Finite automata for $(P^*Q)^*$ is given by :



ε-move can be removed by duplicating transitions of q_1 on q_0.

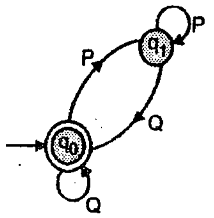
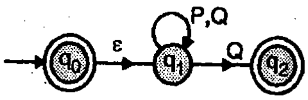
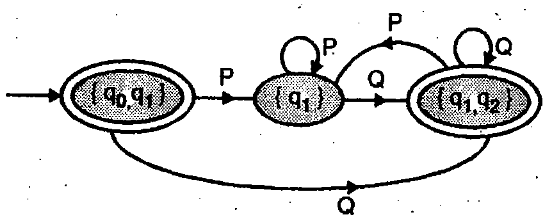


Fig. 3.3.1 : FA for (P*Q)*

Finite automata for $\varepsilon + (P+Q)^*Q$ 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.

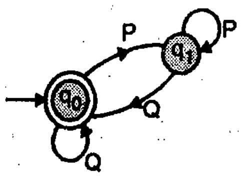


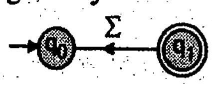
Fig. 3.3.2 : FA for ε + (P + Q)*Q

DFAs for $(P + Q)^*$ given in Fig. 3.3.1 and $\varepsilon + (P+Q)^*Q$ given in Fig. 3.3.2 are same.

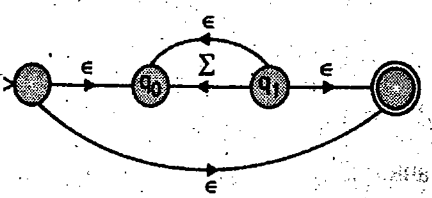
$\therefore \ (P^*Q)^* = \varepsilon + (P+Q)^*Q.$

7. $\phi^* = \epsilon$

An FA for ϕ is given by



This FA can be extended for ϕ^*.



The above FA accepts the only string $\in$.

$\therefore \ \phi^* = \in$

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) $r_1 + \varepsilon \neq r_1$ (2) $r_1 \cdot r_2 \neq r_2 \cdot r_1$

(3) $r_1 \cdot r_1 \neq r_1$ (4) $r_1 + (r_2 \cdot r_3) \neq (r_1 + r_2) \cdot (r_1 \cdot r_3)$

Solution :

Let us make the following assumptions

$$r_1 = 0, \quad r_2 = 1, \quad r_3 = 1$$

R.E.	LHS of RE	RHS of RE	Remark
$r_1 + \varepsilon \neq r_1$	$0 + \varepsilon$	0	L.H.S. ≠ R.H.S.
$r_1 \cdot r_1 \neq r_1$	0.0	0	L.H.S. ≠ R.H.S.
$r_1 \cdot r_2 \neq r_2 \cdot r_1$	0.1	1.0	L.H.S. ≠ R.H.S.
$r_1 (r_2 \cdot r_3) \neq (r_1 + r_2) (r_1 \cdot r_3)$	0 + 10	$(0+1) \cdot 00 = 000 + 100$	L.H.S. ≠ R.H.S.

Example 3.3.2

Prove that : $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$

Solution :

$$\begin{aligned} \text{L.H.S.} &= \varepsilon + 1^*(011)^*(1^*(011)^*)^* \\ &= \varepsilon + RR^*, \text{ where R is taken as } 1^*(011)^* \\ &= R^* \quad \text{...[By Equation (3.3.12)]} \\ &= (1^*(011)^*)^* \\ &= (1+011)^* \quad \text{...[By Equation (3.3.14)]} \\ &= \text{R.H.S.} \end{aligned}$$

Example 3.3.3 SPPU - Dec. 12. 4 Marks

Prove that

$(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1)$
$= 0^*1(0 + 10^*1)^*$

Solution :

$$\begin{aligned} \text{L.H.S.} &= (1+00^*1) + (1+00^*1)(0+10^*1)^* \\ &\quad (0+10^*1) \\ &= (1+00^*1)[\varepsilon + (0+10^*1)^*(0+10^*1)] \\ &\quad \text{...[By Equation (3.3.7)]} \\ &= (1+00^*1)(0+10^*1)^* \\ &\quad \text{...[By Equation (3.3.12)]} \\ &= [(\varepsilon + 00^*)1](0+10^*1)^* \\ &\quad \text{...[By Equation (3.3.8)]} \end{aligned}$$

$= 0^*1(0 + 10^*1)^*$ ...[By Equation (3.3.12)]

= R.H.S.

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 = {ε}

(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, baa, 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.

∴ Required strings are {ε, 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 SPPU - May 15. 4 Marks

If S = {a, bb}, find the set of all strings in S^* with string length less than or equal to 5. Also for given S, prove whether the following is true or false.

$(S^*)^+ = (S^+)^*$.

Solution :

Strings of length 0 = $\in$

Strings of length 1 = a

String of length 2 = aa, bb

Strings of length 3 = aaa, abb, bba

String of length 4 = aaaa, bbbb, aabb, bbaa, abba,

String of length 5 = aaaaa, abbbb, bbabb, bbbba, aaabb, aabba, abbaa, bbaaa

$(S^*)^+ = (S^+)^*$ is true as $\in$ belongs to both $(S^*)^+$ and $(S^+)^*$

Example 3.3.7

Find all possible regular expression $L \subseteq \{a, b\}^*$

(a) the set of all strings ending in b.

(b) the set of all strings ending in ba.

(c) the set of all strings ending neither in b nor in ba.

(d) the set of all strings ending in ab

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

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

Example 3.3.8 SPPU - May 16. 6 Marks

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

(1) The language of all strings containing exactly two 0's.

(2) The language of all strings containing at least two 0's.

(3) The language of all strings that do not end with 01.

(4) The language of all string starting with 11.

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) All string starting with 11.

R.E. $= 11\,(0+1)^*$

Example 3.3.9

Construct NFA with ε-moves for

R.E. $= 0.1[(10^* + 111)^* + 0\,]^*1$. 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 below.

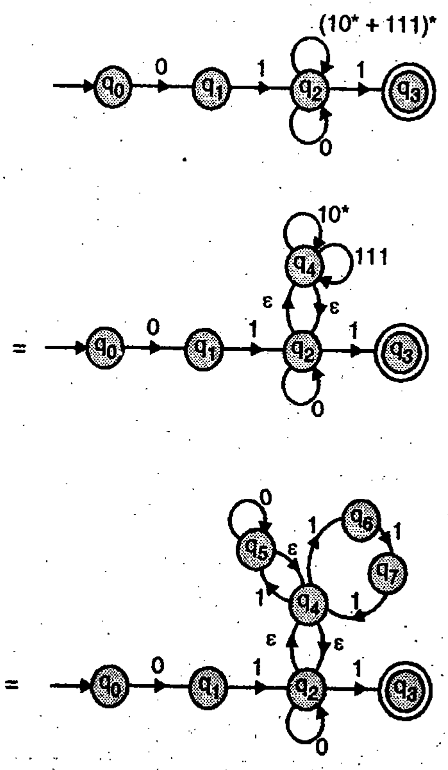


Fig. Ex. 3.3.9(a) : ε-NFA for example 3.3.9

DFA is being constructed from ε-NFA. It is shown in Fig. Ex. 3.3.9(b).

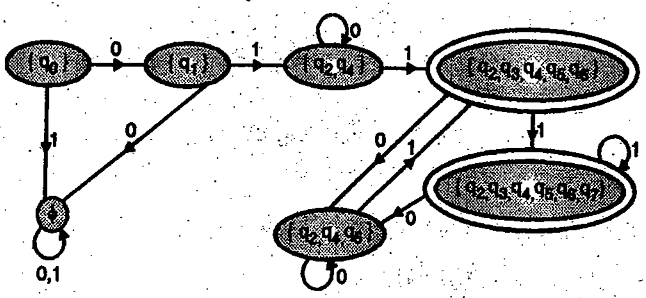


Fig. 3.3.9(b) : DFA for example 3.3.9

Example 3.3.10

Which of the following are true ? Explain.

(i) baa $\in a^*b^*a^*b^*$

(ii) $b^*a^* \cap a^*b^* = a^* \cup b^*$

(iii) $a^*b^* \cap b^*c^* = \phi$

(iv) abcd $\in (a\,(cd)^*b)^*$

Solution :

(i) baa $\in a^*b^*a^*b^*$

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

$(\varepsilon + a + aa + \ldots)(\varepsilon + b + bb + \ldots)$

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) $b^*a^* \cap a^*b^* = a^* \cup b^*$

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

...[From Equation (3.3.12)]

$= \varepsilon + aa^* + bb^* + bb^*aa^*$

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

$= a^* + b^* + bb^*aa^*$ ...(1)

Similarly,

$a^*b^* = a^* + b^* + aa^*bb^*$ ...(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) $a^*b^* \cap b^*c^* = \phi$

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 3.3.11 SPPU - May 14. 8 Marks

For each of the following draw DFA of following regular expressions :

(a) $(11+00)^*$ (b) $(111+100)^*0$

(c) $0 + 10^* + 01^*0$

Solution :

(a) DFA for (11 + 00)*

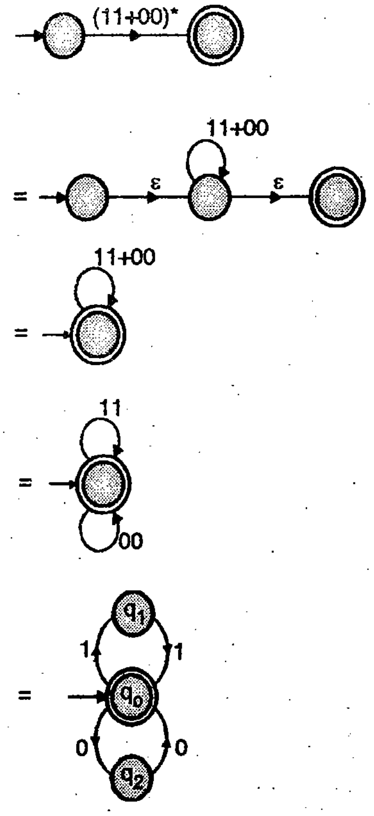


Fig. Ex. 3.3.11

(b) DFA for (111 + 100)*0

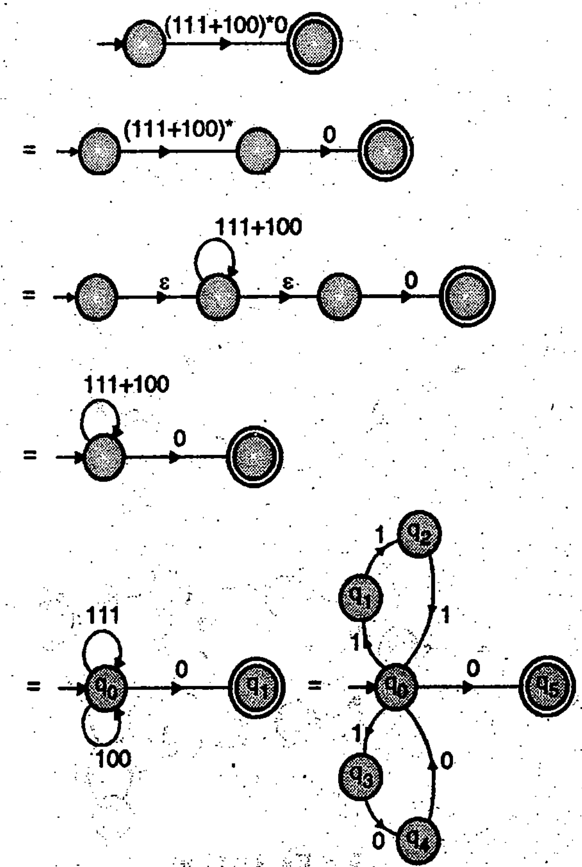


Fig. Ex. 3.3.11(a)

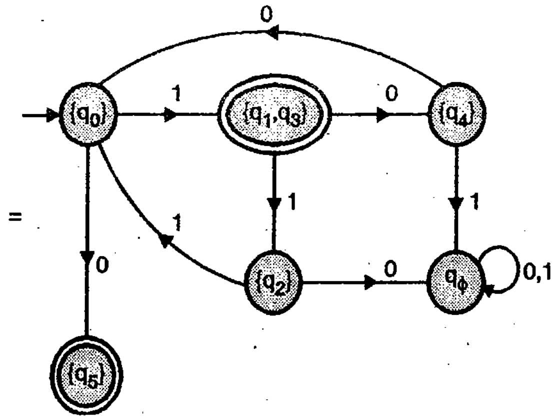


Fig. Ex. 3.3.11(b)

(c) DFA for 0 + 10* + 01* 0

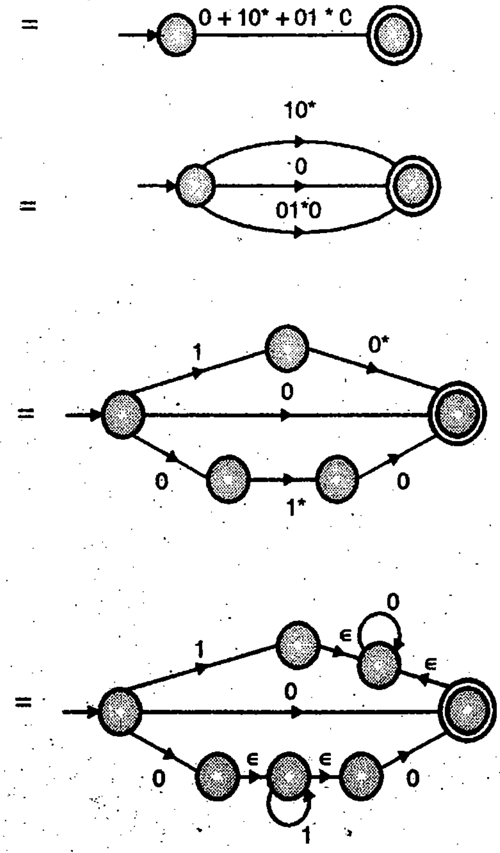


Fig. Ex. 3.3.11(c)

Example 3.3.12 SPPU- Dec. 16. 6 Marks

Construct finite automata equivalent to the following regular sets.

(a) $10 + (0 + 11)0^*1$

(b) $01[((10)^* + 111)^* + 0]^*1$

(c) $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

(d) $1(1 + 10)^* + 10(0 + 01)^*$

Solution :

(a) FA for 10 + (0 + 11) 0*1

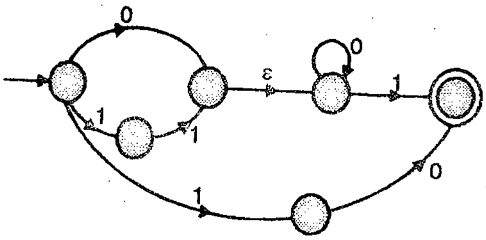


Fig. Ex. 3.3.12(a)

(b) FA for $01[((10)^* + 111)^* + 0]^*1$

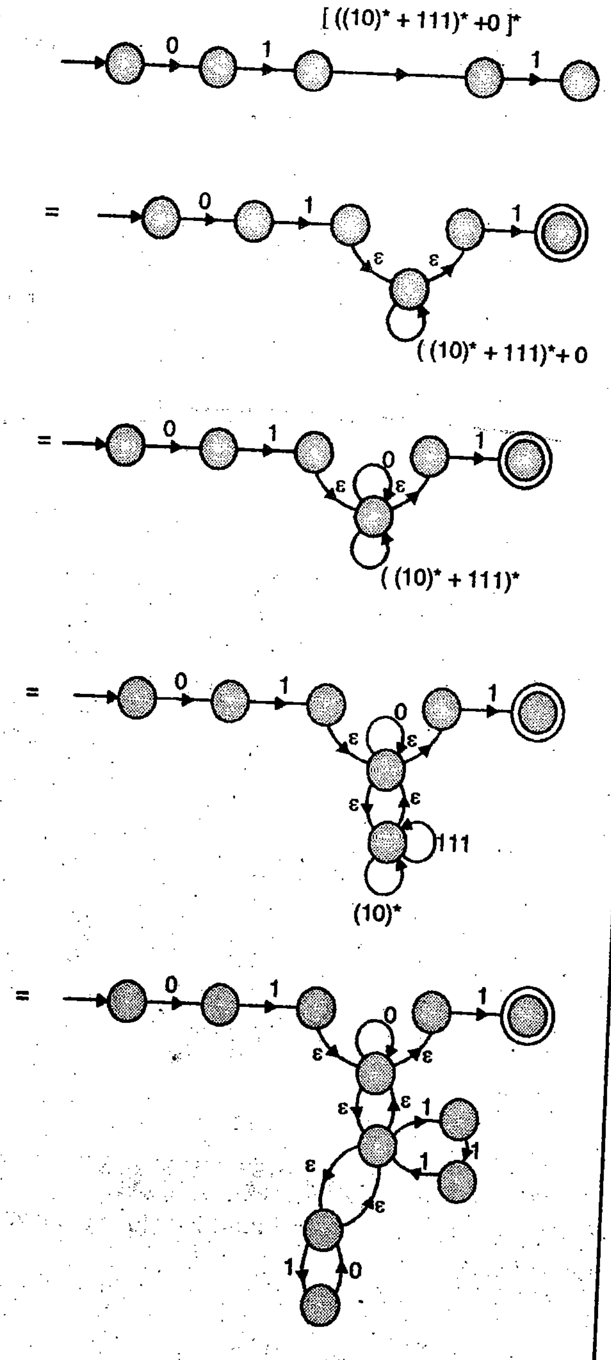


Fig. Ex. 3.3.12(b)

(c) FA for $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

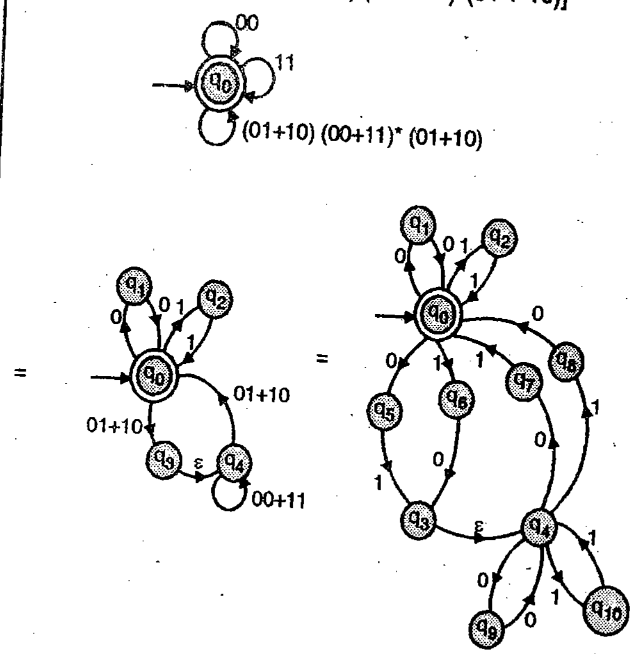


Fig. Ex. 3.3.12(c)

(d) FA for $1(1 + 10)^* + 10(0 + 01)^*$

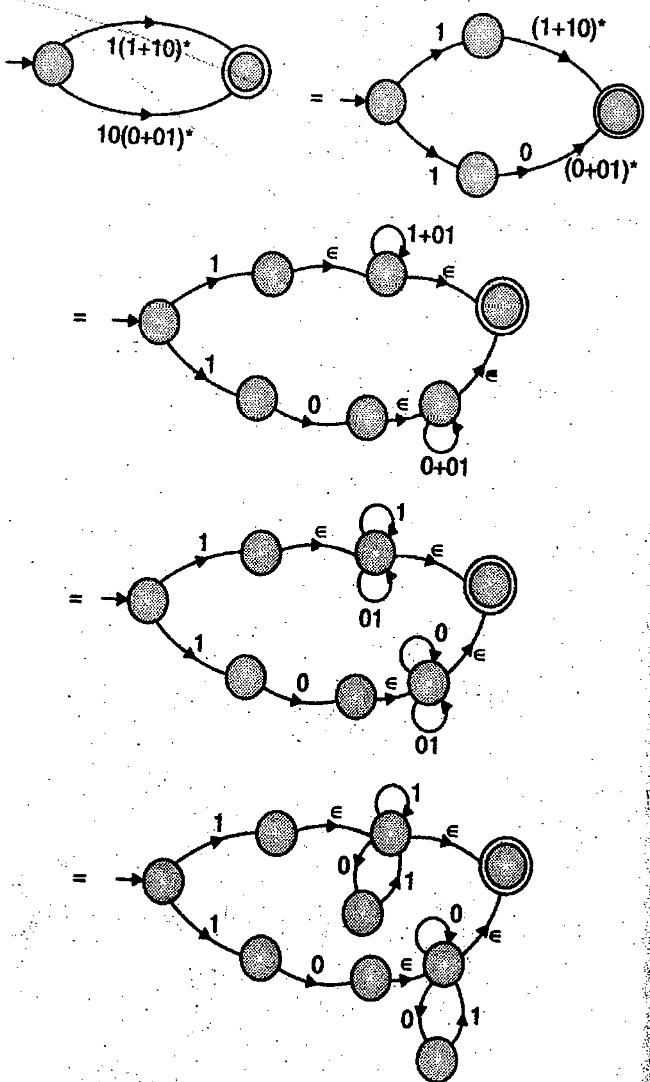


Fig. Ex. 3.3.12(d)

Example 3.3.13

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 $(1 + 10 + 110)^*0$

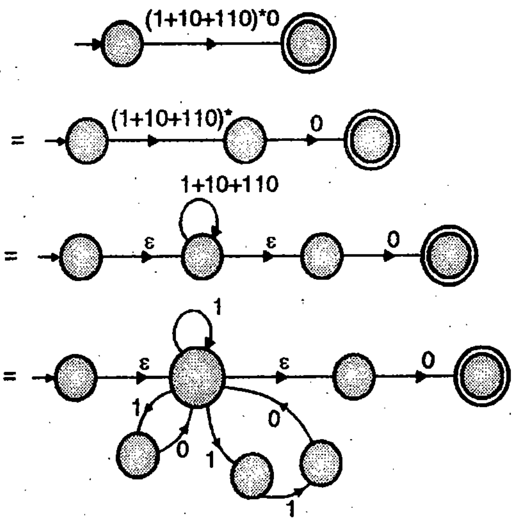


Fig. Ex. 3.3.13

2. F.A. for $1(01 + 10)^* + 0(11 + 10)^*$

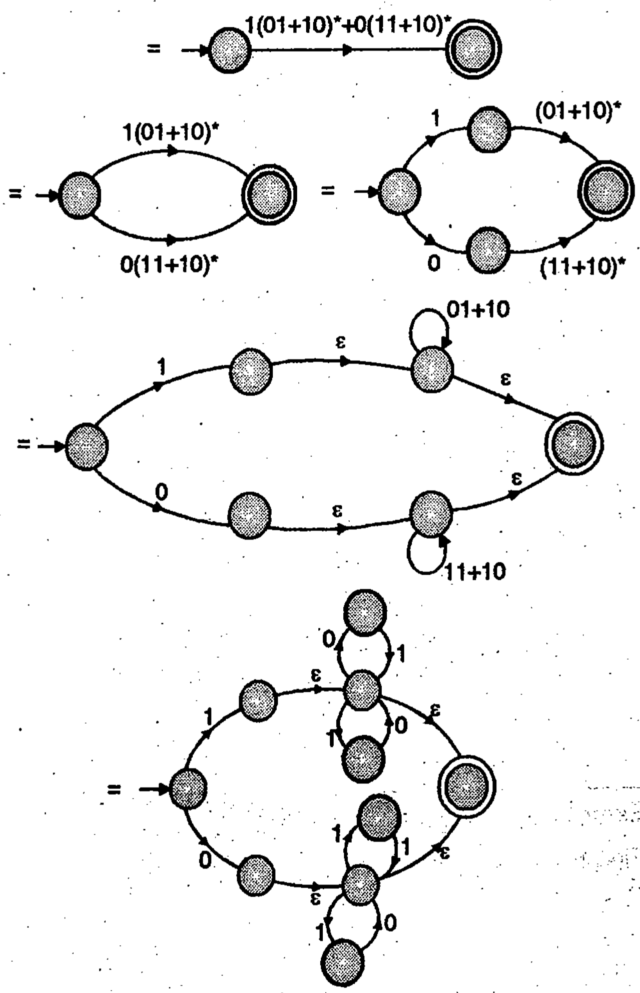


Fig. Ex. 3.3.13(a)

3. F.A. for $(010 + 00)^*(10)$

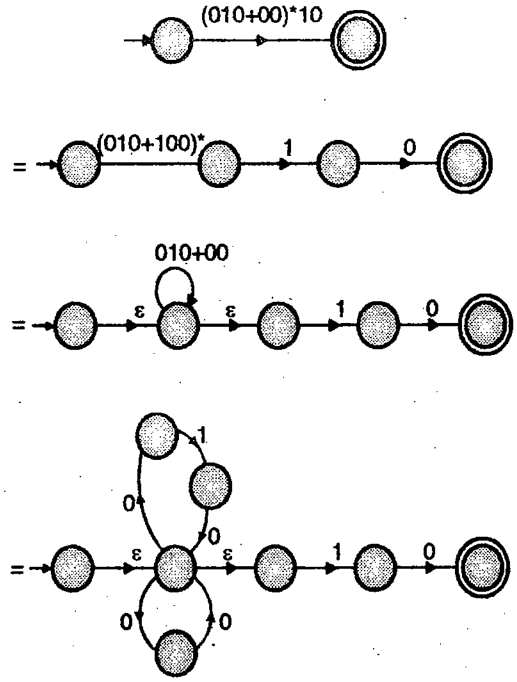


Fig. Ex. 3.3.13(b)

4. F.A. for $1(1 + 10)^* + 10(0 + 01)^*$

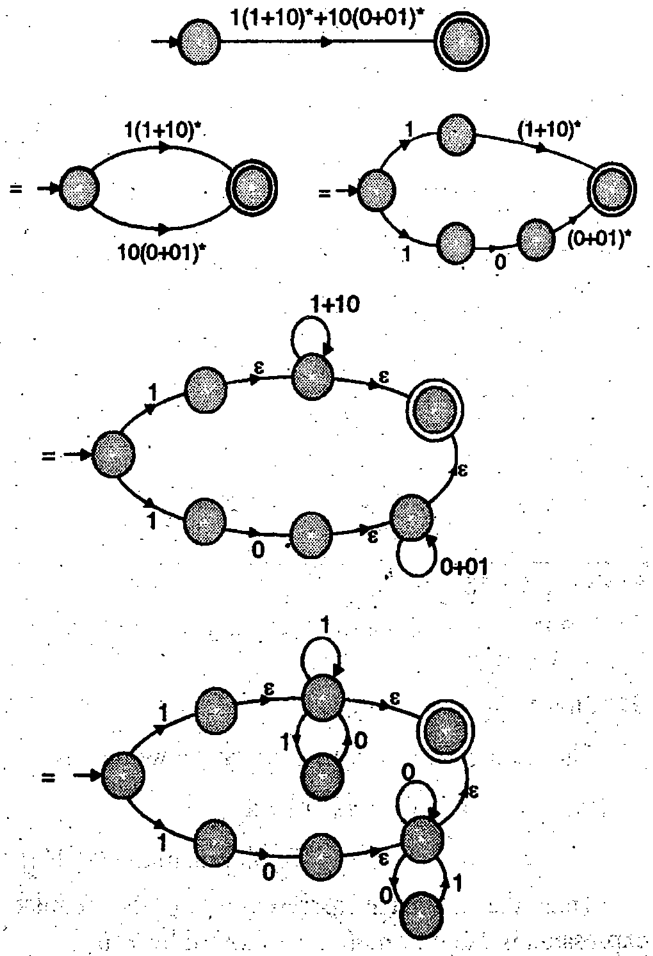


Fig. Ex. 3.3.13(c)

Example 3.3.14

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

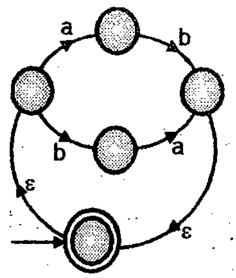


Fig. Ex. 3.3.14

Thus the ε-NFA for $(ab + ba)^*aa(ab + ba)^*$ can be drawn as :

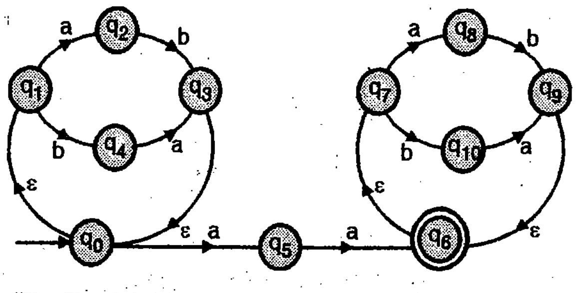


Fig. Ex. 3.3.14(a) : ε-NFA for the given example

DFA using the direct approach is shown below,

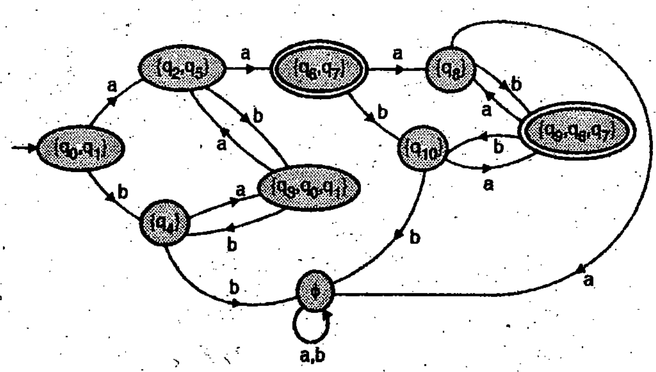


Fig. Ex. 3.3.14(b) : An equivalent DFA

Example 3.3.15

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

$$\text{R.E.} = (a^*a)b + b = (a^*a + \varepsilon)b$$

$$= a^*b \quad \text{[From Equation (3.3.12)]}$$

Thus the language represented by the regular expression is "zero or more a's followed by a 'b' ".

Example 3.3.16

Consider the two regular expression

$r = 0^* + 1^*$, $s = 01^* + 10^* + 1^*0 + (0^*1)^*$

a) Find the string corresponding to r but not to s.
b) Find the string corresponding to s but not to r.
c) Find the string corresponding to both r and s.
d) Find the string corresponding to neither r nor s.

Solution :

a) Find the string corresponding to r but not to s.

00 is in r but not in s

b) Find the string corresponding to s but not to r.

01 is in s but not in r

c) Find the string corresponding to both r and s.

11 is both in r and s.

d) The string 010 is neither in r nor in s.

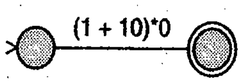
Example 3.3.17

For the following regular expression, draw an FA recognizing the corresponding language.

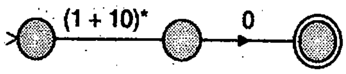
$r = (1 + 10)^*\ 0$.

Solution :

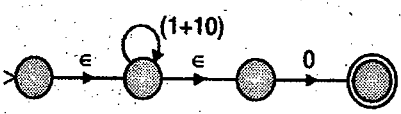
1.



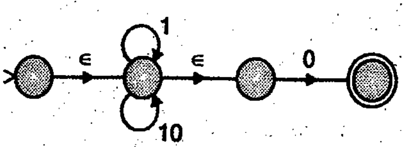
2.



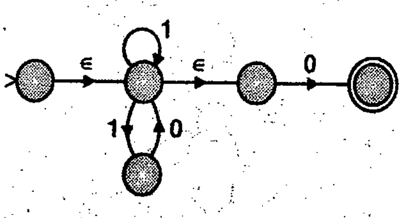
3.



4.



5.



Example 3.3.18 SPPU - Dec. 14, 4 Marks

Find FA for a given RE : $(0 + 1)^*.\ (010 + 101).\ (0 + 1)^*$.

Solution :

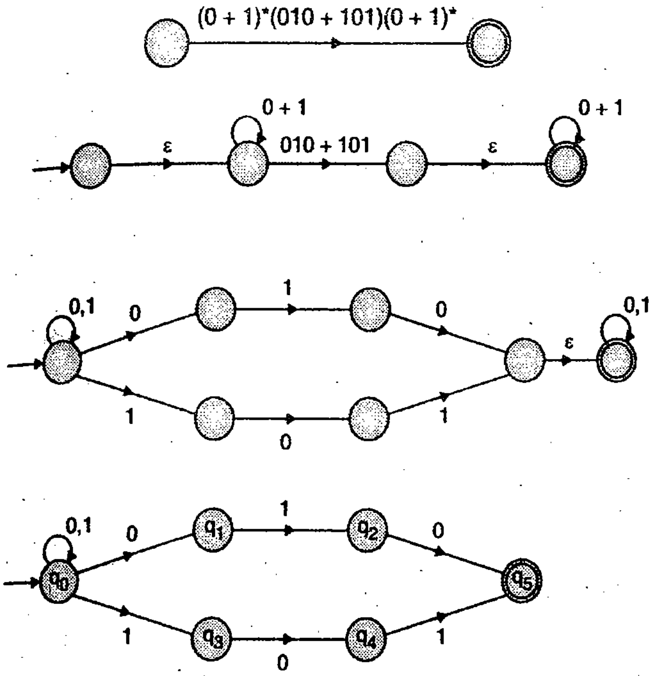


Fig. Ex. 3.3.18

Example 3.3.19 SPPU - May 12, May 15. 6 Marks

Construct DFA for the Regular Expression

(i) (a + b)* abb (ii) (11)*.010.(11)*

Solution :

(i) Regular expression: (a + b)* abb

Step 1 : R.E. to ∈ – NFA

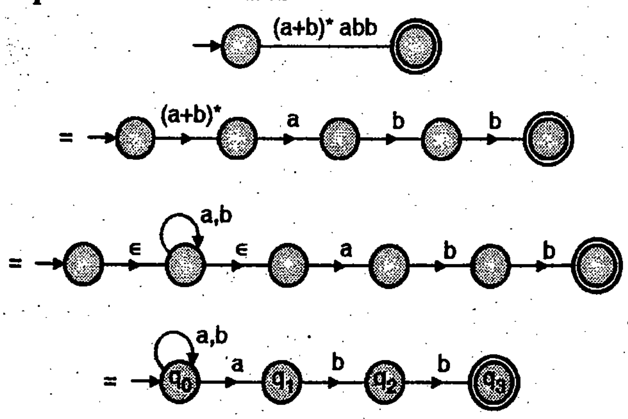


Fig. Ex. 3.3.19(a)

Step 2 : NFA to DFA

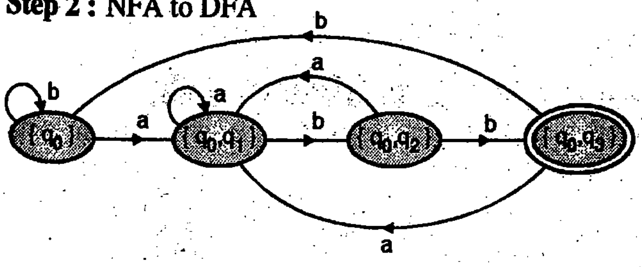
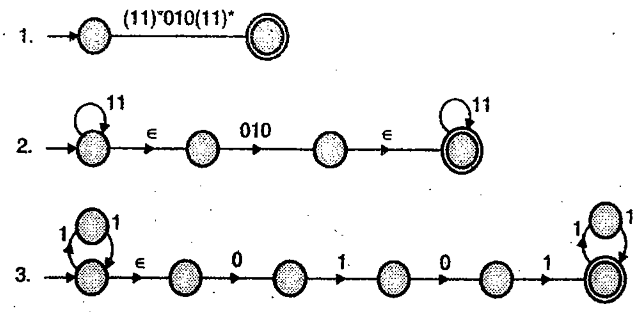


Fig. Ex. 3.3.19(b)

(ii) Regular Expression : (11)*.010.(11)*



Syllabus Topic : DFA to RE Conversions

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 : State/Loop Elimination

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,

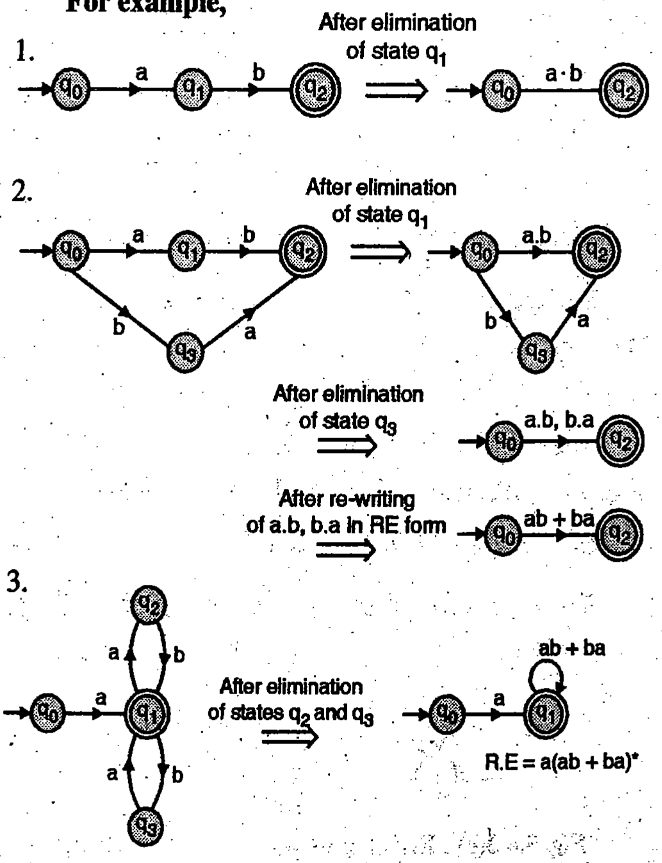


Fig. 3.4.1

3.4.1.1 A Generic One State Machine

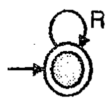


Fig. 3.4.2 : A generic one-state machine

The regular expression for strings accepted by a one state generic machine is **R***.

3.4.1.2 A Generic Two State Machine

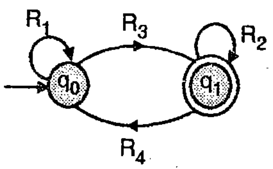


Fig. 3.4.3(a) : A generic two-state machine

A regular expression for a two sate 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

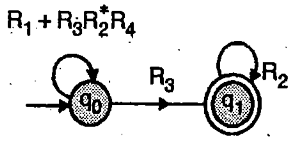


Fig. 3.4.3(b) : Effect of loop is moved to state q_0

Starting from the start state q_0, one can come back to q_0 either on R_1,

or $R_3R_2^*R_4$

The equivalent R.E. for machine shown in Fig. 3.4.3(b) can be written easily. It is given by :

$$\text{R.E.} = (R_1 + R_3R_2^*R_4)^*R_3R_2^*$$

Machine after moving the effect of loop to state q_1

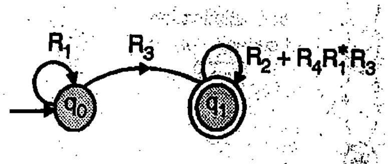


Fig. 3.4.3(c) : Effect of loop is moved to state q_1

Starting from the state q_1, one can come back to q_1 either on R_2.

or $R_4R_1^*R_3$

The equivalent R.E. for machine shown in Fig. 3.4.3(c) can be written easily. It is given by :

$$\text{R.E.} = R_1^*R_3(R_2 + R_4R_1^*R_3)^*$$

Example 3.4.1

Find a regular expression for the given two state generic machine.

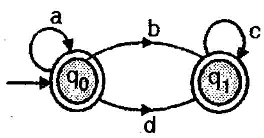


Fig. Ex. 3.4.1(a) : A two state generic machine

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).

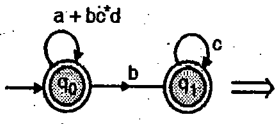


Fig. Ex. 3.4.1(b)

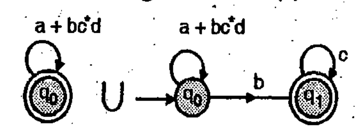


Fig. Ex. 3.4.1(c) : Union of two machines

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.

$$\therefore \text{Final R.E.} = (a + bc^*d)^* + (a + bc^*d)^*bc^*$$
$$= (a + bc^*d)^*[\varepsilon + bc^*]$$

Example 3.4.2 SPPU - May 12. 6 Marks

Find a regular expression for the given three state machine.

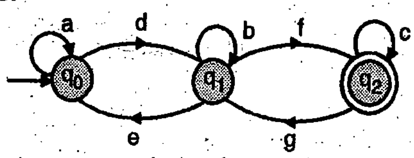


Fig. Ex. 3.4.2(a)

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. 3.4.2(b).

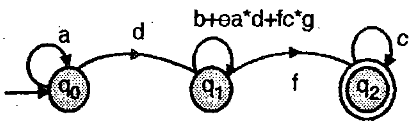


Fig. Ex. 3.4.2(b) : Moving the effect of loops on state q_1

The state q_1 can be eliminated.

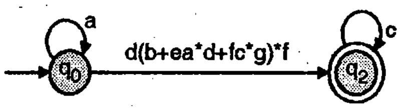


Fig. Ex. 3.4.2(c) : After elimination of state q_1

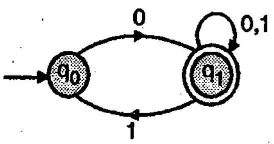
Now, the regular expression can be written as,

$$\text{R.E.} = a^*d(b + ea^*d + fc^*g)^*fc^*$$

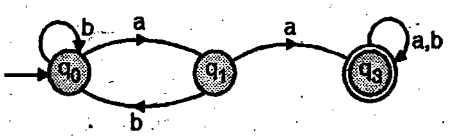
Example 3.4.3

Find the regular expression for the following DFA's.

(i)



(ii)



(iii)

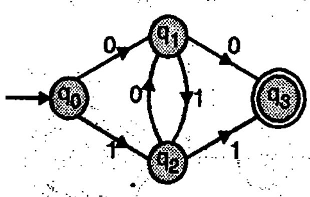


Fig. Ex. 3.4.3

Solution :

(i) Machine is redrawn after the effect of loop between q_0 and q_1 is transferred to q_0.

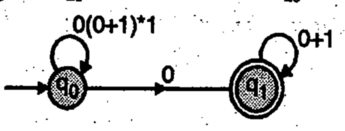


Fig. Ex. 3.4.3(a)

$$\text{R.E.} = (0(0 + 1)^*1)^*0(0 + 1)^*$$

(ii) Machine is re-drawn after the effect of loop between q_0 and q_1 is transferred to q_0

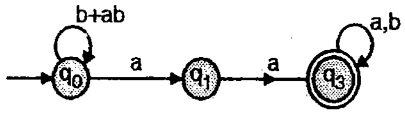


Fig. Ex. 3.4.3(b)

$$\text{R.E.} = (b + ab)^*aa(a + b)^*$$

(iii) Machine is redrawn after the effect of loop between q_1 and q_2 is transferred to both q_1 and q_2.

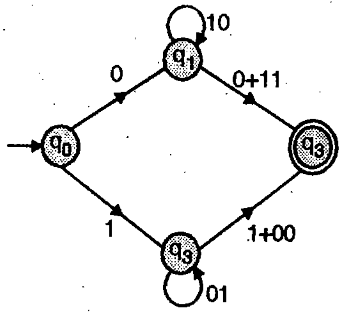


Fig. Ex. 3.4.3(c)

There are two paths from q_1 and q_3 [0 + 11]. There are two paths from q_2 to q_3 [1 + 00].

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

Example 3.4.4

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. 3.4.4(a).

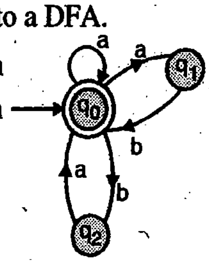


Fig. Ex. 3.4.4(a) : An equivalent NFA

Step 2 : NFA to DFA : An equivalent DFA is given in Fig. Ex.3.4.4(b).

	a	b
$\rightarrow\{q_0\}^*$	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_2\}$	$\{q_0\}$	ϕ
$\{q_0, q_1\}^*$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}^*$	$\{q_0, q_1\}$	$\{q_2\}$

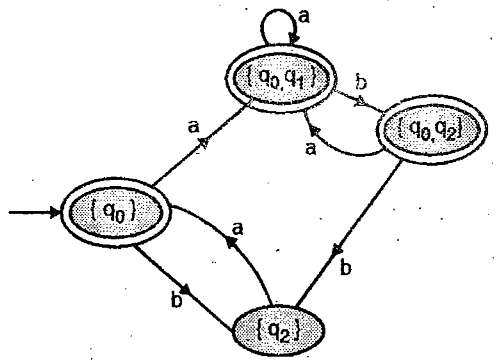


Fig. Ex. 3.4.4(b) : An equivalent DFA

Step 3 : Minimization of DFA obtained in Fig. Ex. 3.4.4(b).

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. 3.4.4(c).

	a	b
$\{q_0, q_2\} = \rightarrow A^*$	B	C
$\{q_0, q_1\} = B^*$	B	A
$\{q_2\} = C$	A	ϕ

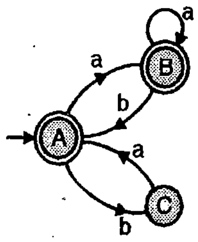


Fig. Ex. 3.4.4(c) : An equivalent minimal DFA

Step 4 : Writing of regular expression for DFA derived in Fig. Ex. 3.4.4(c).

- There are two loops on A.
  1. Loop through A → C → A on input ba
  2. Loop through A → B → B → .... → B → A on input aa*b
  3. There are two final states.

An equivalent machine without a loop is shown in Fig. Ex. 3.4.4(d).

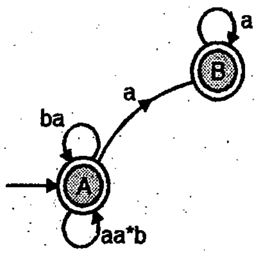


Fig. Ex. 3.4.4(d) : An equivalent machine without a loop

Regular expression for the machine of Fig. Ex. 3.4.4(d) is given by :

$$\text{R.E.} = \underbrace{(aa^*b + ba)^*}_{\text{Corresponding to state A}} + \underbrace{(aa^*b + ba)^*aa^*}_{\text{Corresponding to state B}}$$

$= (aa^*b + ba)^*[\varepsilon + aa^*]$

$= (aa^*b + ba)^*a^*$ ...[By Equation (3.3.12)]

$= (a^*ab + ba)^*a^*$ ...(By Equation (3.3.10)]

$=$ L.H.S.

Example 3.4.5

Prove or disprove the following for regular expression r, s and t.

(a) $(rs + r)^*r = r\,(sr + r)^*$

(b) $s(rs + s)^*r = rr^*s\,(rr^*s)^*r$

(c) $(r + s)^* = r^* + s^*$

Solution :

(a) $(rs + r)^*r = r(sr + r)^*$

This can be proved by drawing a F.A. for $(rs + r)^*r$

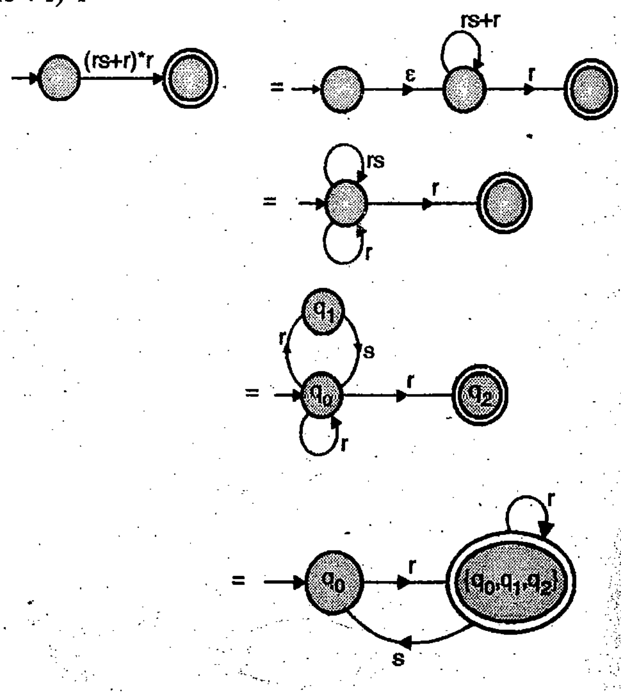


Fig. Ex. 3.4.5

Moving the effect of the loop on $\{q_0, q_1, q_2\}$, we get,

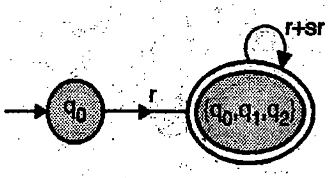


Fig. Ex. 3.4.5(a)

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 3.4.6 SPPU - May 14, 3 Marks

Find the regular expression corresponding to each of the following subset of $\{0, 1\}^*$

(a) The language of all strings not containing the substring 000.

(b) The language of all strings that do not contain the substring 110.

(c) The language of all strings containing both 101 and 010 as substring.

(d) Represent following formal Languages using regular expressions.

 (1) All string's of a's and b's without any combination of double letters.

 (2) All string's of 0's and 1's with even number of 0's.

 (3) All string's of a's and b's containing at least two a's.

Solution :

(a) 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,

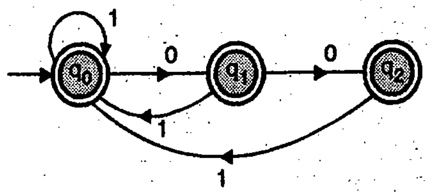


Fig. Ex. 3.4.6

Moving the effect of loops on state q_0 we get,

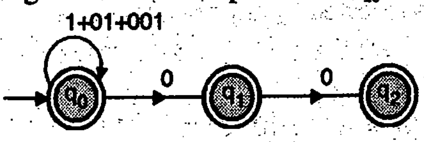


Fig. Ex. 3.4.6(a)

$\therefore$ R.E. $= (1 + 01 + 001)^*(\varepsilon + 0 + 00)$

(b) 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,

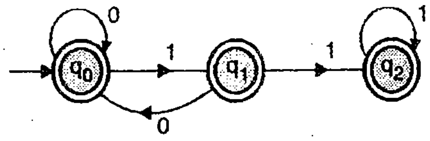


Fig. Ex. 3.4.6(b)

Moving the effect of loop on state q_0, we get,

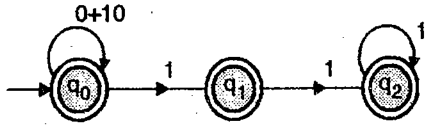


Fig. Ex. 3.4.6(c)

$\therefore$ R.E. $= (0 + 10)^* + (0 + 10)^*1 + (0 + 10)^*111^*$

(c) The language of all strings containing both 101 and 010 as substring.

$\therefore$ R.E. $= (0 + 1)^*101(0 + 1)^*010(0 + 1)^* + (0 + 1)^*010(0 + 1)^*101(0 + 1)^*$

(d) Representing following formal languages using regular expressions.

(1) All string's of a's and b's without any combination of double letters.

The DFA for the given language is

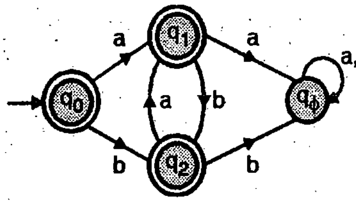


Fig. Ex. 3.4.6(d)

The R.E. can be written form the above DFA.

The R.E. $= \in + a(ba)^* (\in + b) + b\ (ab)^* (\in + a)$

(2) All string's of 0's and 1's with even number of 0's.

The DFA for the given language is

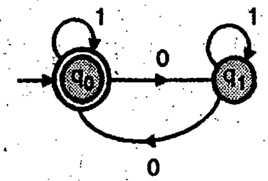


Fig. Ex. 3.4.6(e)

which can be written as

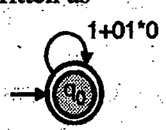


Fig. Ex. 3.4.6(f)

∴ R.E. $= (1 + 01^*0)^*$

(3) All string's of a's and b's containing at least two a's.

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

Example 3.4.7 SPPU - May 14, 3 Marks

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) Identities in C language.

(iv) Obtain in plain English, the language represented by following regular expression

(a) $0^*\,(10^*\,10^*)^*1\,(0^*\,10^*\,1)^*\,0^*$

(b) $0^*\,(0^*\,10^*\,1)^*\,0^*$

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

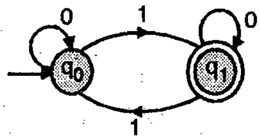


Fig. Ex. 3.4.7(a) : DFA for strings having odd number of 1's

An equivalent FA with effect of loop between q_0 and q_1 moved to q_0.

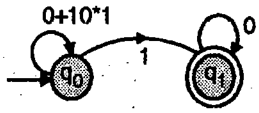


Fig. Ex. 3.4.7(b) : An equivalent machine of DFA given in Fig. Ex. 3.4.7(a)

Step 2 : Writing of R.E. from Fig. Ex. 3.4.7(b).

R.E. $= (0 + 10^*1)^*10^*$

(ii) $\Sigma = \{0, 1\}$ Triple 0 must never appear in string.

Step 1 : Drawing of an equivalent DFA.

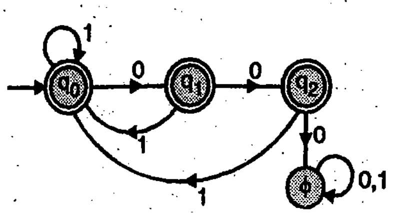


Fig. Ex. 3.4.7(c) : DFA for strings not having 000

An equivalent FA with effect of loops moved to q_0 is shown in Fig. Ex. 3.4.7(d).

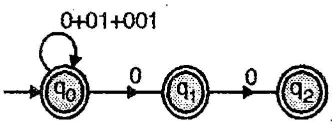


Fig. Ex. 3.4.7(d) : An equivalent FA for DFA of Fig. Ex. 3.4.7(c)

Step 2 : Writing of R.E.

R.E. $= (1 + 01 + 001)^*(\varepsilon + 0 + 00)$

(iii) Identifiers in C language.

R.E. $= (A + B + \ldots + Z + a + b + z)$
$(A + B \ldots + Z + a + \ldots z + 0 + \ldots 9)^*$

(iv) (a) The string contains odd number of 1's.

(b) The string contains even number of 1's.

Example 3.4.8 SPPU - Dec. 14. 4 Marks

Construct RE for given FSM in Fig. Ex. 3.4.8.

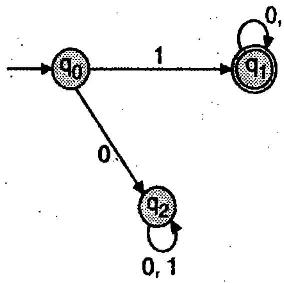


Fig. Ex. 3.4.8

Solution :

Strings accepted by the state $q_0 = \in$

Strings accepted by the state $q_1 = 1\,(0 + 1)^*$

Strings accepted by the state $q_2 = 0\,(0 + 1)^*$

∴ Required R.E. $= \in + 1\,(0 + 1)^* + 0\,(0 + 1)^*$

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

Example 3.4.9

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, ...} then what is r ?

(iii) If L (r) = {00, 010, 0110, 01110, ...} 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.

(v) $L_1 = \{b^2, b^5, b^8, b^{11}, b^{14} \ldots\}$

(vi) $L_2 = \{a^{2n+1} \mid n > 0\}$

Solution :

(i) R.E. $= a^*c^*$

(ii) $L(r) = \{0, 2, 01, 21, 011, 211, 0111, \ldots\}$

$$r = 0 + 2 + 01 + 21 + 011 + 211 + 0111 + \ldots$$
$$= 0(\varepsilon + 1 + 11 + 111 + \ldots) + 2(\varepsilon + 1 + 11 + \ldots) = 01^* + 21^*$$
$$\mathbf{r = (0 + 2)1^*}$$

(iii) $L(r) = \{00, 010, 0110, 01110, \ldots\}$

$$\textbf{R.E.} = \mathbf{0(1)^*0}$$

(iv) DFA for the given language can be drawn as given below.

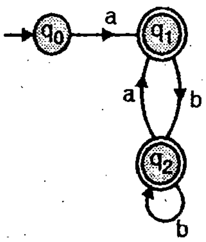


Fig. Ex. 3.4.9

Moving the loop between q_1 and q_2 and q_1 we get :

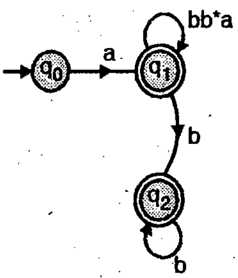


Fig. Ex. 3.4.9(a)

R.E. corresponding to $q_1 = a(bb^*a)^*$

R.E. corresponding to $q_2 = a(bb^*a)^*bb^*$

$\therefore$ Combined R.E. $= a(bb^*a)^* + a(bb^*a)^*bb^*$

$$\mathbf{= a(bb^*a^*) + [\varepsilon + bb^*]}$$

(v) The required R.E. $= bb(bbb)^*$

(vi) The required R.E. $= a(aa)^*$

Example 3.4.10

Let $\Sigma = \{0, 1\}$, construct NFA and hence regular expression for each of the following :

(a) $L_1 = \{\omega \in \Sigma^* \mid \omega$ has at least one pair of consecutive 0's$\}$

(b) $L_2 = \{\omega \in \Sigma^* \mid \omega$ has no pair of consecutive zero$\}$

(c) $L_3 = \{\omega \in \Sigma^* \mid \omega$ is a string with either 1 preceding a 0 or 0 preceding 1$\}$

(d) $L_4 = \{\omega \in \Sigma^* \mid$ w consists of an even number of 0's followed by odd number of 1's$\}$.

Solution :

(a) $L_1 = \{\omega \in \Sigma^* \mid \omega$ has at least one pair of consecutive 0's$\}$

The equivalent FA is shown in Fig. Ex. 3.4.10(a).

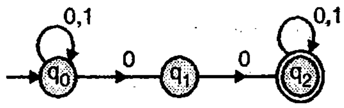


Fig. Ex. 3.4.10(a) : FA for example (a)

$$\textbf{R.E.} = \mathbf{(0 + 1)^*00(0 + 1)^*}$$

(b) $L_2 = \{\omega \in \Sigma^* \mid \omega$ has no pair of consecutive zero$\}$

The equivalent FA is shown in Fig. Ex. 3.4.10(b)

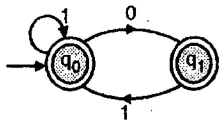


Fig. Ex. 3.4.10(b) : FA for example (b)

The effect of loop between q_0 and q_1 can be move to q_0. A new machine is obtained and it is shown in Fig. Ex. 3.4.10(c).

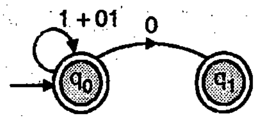


Fig. Ex. 3.4.10(c) : FA after the effect of loop is moved to q_0

R.E. for state $q_0 = (1 + 01)^*$

R.E. for state $q_1 = (1 + 01)^*0$

$\therefore$ The final R.E. $= (1 + 01)^* + (1 + 01)^*0$

$$\mathbf{= (1 + 01)^*[\varepsilon + 0]}$$

(c) $L_3 = \{\omega \in \Sigma^* \mid \omega$ is a string with either 1 preceding a 0 or 0 preceding 1$\}$

The equivalent FA is shown in Fig. Ex. 3.4.10(d).

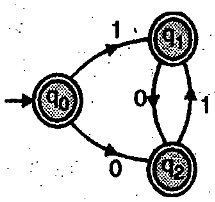


Fig. Ex. 3.4.10(d) : FA for example (c)

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.3.4.10(e).

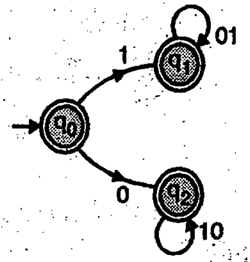


Fig. Ex. 3.4.10(e) : FA after elimination of loop

$$\therefore \textbf{R.E.} = \mathbf{\varepsilon + 1(01)^* + 0(10)^*}$$

(d) $L_4 = \{\omega \in \Sigma^* \mid \omega$ consists of an even number of 0's followed by an odd number of 1's.$\}$

The equivalent FA is shown in Fig. Ex. 3.4.10(f)

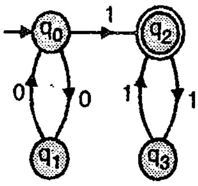


Fig. Ex. 3.4.10(f) : FA for example (d)

The effect of loop between q_0 and q_1, between q_2 and q_3 are removed and FA redrawn as shown in Fig. Ex. 3.4.10(g).

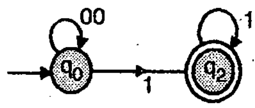


Fig. Ex. 3.4.10(g)

$\therefore$ **R.E.** $= \mathbf{(00)^*1(11)^*}$

Example 3.4.11 SPPU - Dec. 12, 6 Marks

Find regular expressions representing the following sets :

(i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.

(ii) The set of all strings over {a, b} in which the number of occurrences of a is divisible by 3.

(iii) The set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a.

Solution :

(i) FA for strings containing at most one pair of a's is given by :

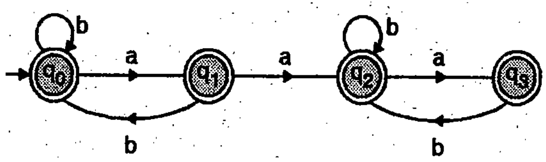


Fig. Ex. 3.4.11(a)

The above FA can be written as

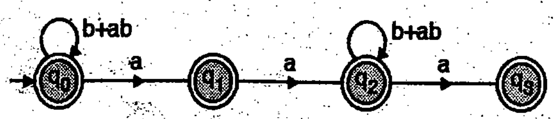


Fig. Ex. 3.4.11 (b)

The R.E. for the above transition graph is given by:

$(b + ab)^* [\in + a + aa(b + ab)^* [\in + a]]$

Extending the above R.E. to represent strings containing at most one pair of a's or atmost one pair of b's, we get

$(b + ab)^* (\in + a + aa (b + ab)^* (\in + a)) + (a + ba)^* (\in + b + bb(a + ba)^* (\in + b))$

(ii) $(b^* ab^* ab^* a)^*$

(iii) The DFA for the referenced language is given in Fig. Ex. 3.4.11 (c).

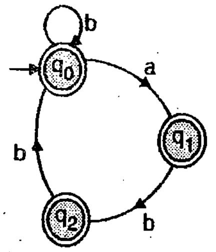


Fig. Ex. 3.4.11 (c)

The equivalent transition diagram can be drawn as given in Fig. Ex. 3.4.11(d):

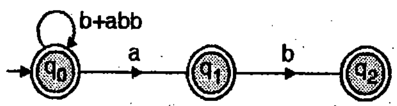


Fig. Ex. 3.4.11(d)

$\therefore$ R.E. $= (b + abb)^* [\in + a + ab]$

Example 3.4.12

Prove the formula $(00^*1)^*1 = 1 + 0(0 + 10)^*11$

Solution : The formula can be proved by drawing an equivalent DFA.

DFA for $(00^*1)^*1$

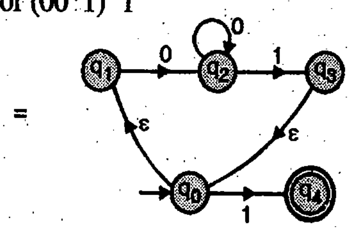


Fig. Ex. 3.4.12 (a) : An equivalent ε-NFA

DFA corresponding to ε-NFA of Fig. Ex. 3.4.12(a) is given Fig. Ex. 3.4.12 (b).

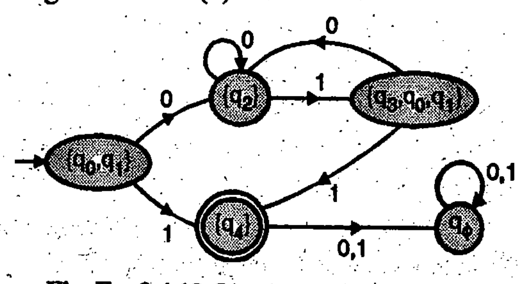


Fig. Ex. 3.4.12 (b) : An equivalent DFA

FA after moving the effect of loop between $\{q_2\}$ and $\{q_3, q_0, q_1\}$ is redrawn in Fig. Ex. 3.4.12 (c).

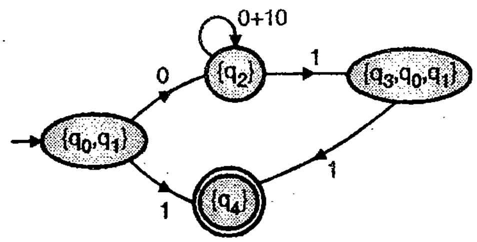


Fig. Ex. 3.4.12 (c) : Effect of loop is moved to $\{q_2\}$

There are two paths between the start $\{q_0, q_1\}$ and the final state $\{q_4\}$

1. First path is represented by

   R.E. $= 1$

2. Second path is represented by

   R.E. $= 0(0 + 10)^*11$

$\therefore$ The equivalent regular expression for FA Fig. Ex. 3.4.12(c) is,

R.E. $= 0(0 + 10)^*11 + 1$

Hence the formula is proved.

Example 3.4.13 SPPU – Dec. 13, 6 Marks

(i) Write regular expression for following languages over $\{0, 1\}^*$:
- The set of strings that begin with 110
- The set of all strings not containing 101 as a substring.

(ii) Give English description of the language of the following regular expression $(1 + \varepsilon)(00^*1)^*0^*$.

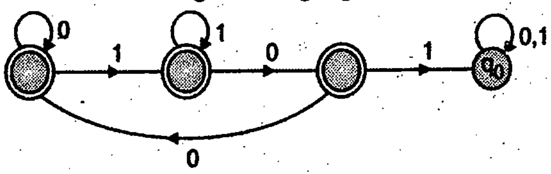
Solution :

(i) The set of strings that begin with 110

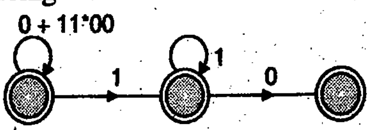
R.E. $= 110(1 + 0)^*$

The set of strings not containing 101 as a substring.

Step 1 : DFA for the given language



Step 2 : Reducing the machine



$\therefore$ R.E. $= (0 + 11^*00)^*[\in + 11^* + 11^*0]$

(ii) A language over Σ (1, 0)* in which two 1's are never together.

Example 3.4.14 SPPU – Dec. 15, 6 Marks

Write regular expressions for the following languages over the alphabet $\Sigma = \{a, b\}$

i) All strings that do not end with 'aa'.

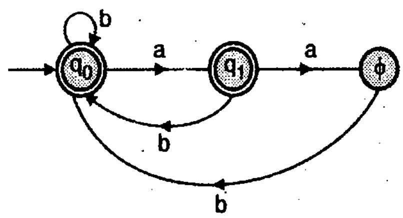
ii) All strings that contain an even number of 'b' s.

iii) All strings which do not contain the substring 'ba'.

Solution :

(i) We can write the R.E. from the DFA for the given language.

DFA for strings not ending in 'aa' :



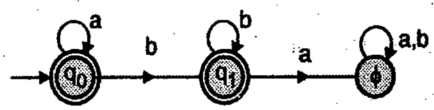
Strings represented by $q_0 = (b + ab + aab)^*$

String represented by $q_1 = (b + ab + aab)^*a$

$\therefore$ Required R.E. $= (b + ab + aab)^* + (b + ab + aab)^*a$

(ii) $(a^*ba^*ba^*)^*$

(iii) DFA for strings not containing the substring 'ba'.



$\therefore$ R.E. $= a^* + a^*bb^*$

Syllabus Topic : 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

$$R = Q + RP \quad \ldots(3.4.1)$$

has a unique solution given by,

$$R = QP^* \quad \ldots(3.4.2)$$

Proof : Let us verify whether QP^* is a solution to the equation.

$$R = Q + RP$$

On substitution of QP^* for R in Equation (3.4.1),

$$R = Q + RP = Q + QP^*P$$

$$= Q^*(\varepsilon + P^*P) = QP^*$$

Thus the Equation (3.4.1) is satisfied by $R = QP^*$. We still do not know whether QP^* is a unique solution or not.

To establish uniqueness, we expand

$$\begin{aligned} R &= Q + RP \\ &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \\ &= Q + QP + (Q + RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &\vdots \\ &= Q + QP + QP^2 + \ldots + QP^i + RP^{i+1} \\ &= Q(\varepsilon + P + P^2 + \ldots + P^i) + RP^{i+1} \quad \ldots(3.4.3) \end{aligned}$$

here, i is any arbitrary integer.

Let us take a string $\omega \in R$ | length of $\omega = k$.

Substituting k for i in Equation (3.4.3)

$$R = Q(\varepsilon + P + P^2 + \ldots + P^k) + RP^{k+1}$$

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 w is contained in R, it must be contained in $Q(\varepsilon + P + P^2 + \ldots + P^k)$.

Conversely, if ω is contained in QP* then for some integer k it must be in $Q\{\varepsilon + P + P^2 + \ldots + P^k\}$, and hence in $R = Q + RP$.

3.4.2.1 Application of Arden's Theorem

SPPU - Dec. 12

University Question

Q. Consider the transition system given in Fig. Q. 1 :

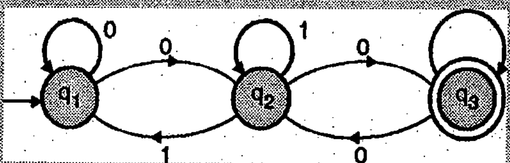


Fig. Q. 1

Prove that the strings recognized are :

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

(Dec. 2012, 8 Marks)

A finite automata can be represented using a system of equations :

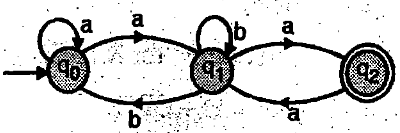


Fig. 3.4.4 : A transition diagram to be converted into a set of equations

Let us consider the FA of Fig. 3.4.4. 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 $\mathbf{q_1}$ and the set of strings generated by coming to q_2 from q_0 is $\mathbf{q_2}$.

The set $\mathbf{q_2}$ can be written as

$$\mathbf{q_2} = \mathbf{q_1} a \quad \ldots(3.4.4)$$

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 $\mathbf{q_1}$ can be expressed in terms of $\mathbf{q_0}$ and $\mathbf{q_2}$.

$$\mathbf{q_1} = \mathbf{q_0} a + \mathbf{q_1} b + \mathbf{q_2} a \quad \ldots(3.4.5)$$

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 $\mathbf{q_0}$ can be expressed in terms of $\mathbf{q_0}$, $\mathbf{q_1}$ and ε.

$$\mathbf{q_0} = \mathbf{q_0} a + \mathbf{q_1} b + \varepsilon \quad \ldots(3.4.6)$$

- These three Equations (3.4.4, 3.4.5, 3.4.6) can be solved with the help of Arden's theorem.
- The set represented by $\mathbf{q_2}$ will be the solution for the FA of Fig. 3.4.4. q_2 is a final state.

Solving of equations using Arden's theorem

1. Substituting for $\mathbf{q_2}$ in Equation (3.4.4). $\mathbf{q_2}$ is given by Equation (3.4.4)

$$\begin{aligned} \mathbf{q_1} &= \mathbf{q_0} a + \mathbf{q_1} b + \mathbf{q_1} aa \\ &= \mathbf{q_0} a + \mathbf{q_1}(b + aa) \quad \ldots(3.4.7) \end{aligned}$$

Equation (3.4.7) is of the form

$$R = Q + RP \text{ with } Q = \mathbf{q_0} a,$$
$$R = \mathbf{q_1} \text{ and } P = (b + aa)$$

and its solution is given by,

$$R = QP^*$$

Thus, $\mathbf{q_1} = \mathbf{q_0} a(b + aa)^* \quad \ldots(3.4.8)$

2. Substituting the value of $\mathbf{q_1}$ from Equation (3.4.8) into Equation (3.4.6)

$$\begin{aligned} \mathbf{q_0} &= \mathbf{q_0} a + \mathbf{q_0} a(b + aa)^* b + \varepsilon \\ &= \varepsilon + \mathbf{q_0}(a + a(b + aa)^* b) \quad \ldots(3.4.9) \end{aligned}$$

The Equation (3.4.9) is of the form

$$R = Q + RP \text{ from } Q = \varepsilon, R = q_0$$

$$\text{and } P = (a + a(b + aa^*)^*b)$$

and its solution is given by,

$$R = QP^*$$

$$\text{Thus, } q_0 = \varepsilon \cdot (a + a(b + aa)^*b)^*$$

$$= (a + a(b + aa)^*b)^* \quad \text{...(3.4.10)}$$

3. Substituting the value of q_0 from Equation (3.4.10) into the Equation (3.4.8).

$$q_1 = (a + a(b + aa)^*)^*a(b + aa)^* \quad \text{...(3.4.11)}$$

4. Substituting the value of q_1 from Equation (3.4.11), into the Equation (3.4.4).

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

Thus the R.E. representing the FA of Fig. 3.4.10 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 3.4.14 SPPU - May 13, 8 Marks

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.

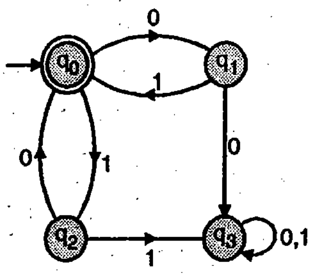


Fig. Ex. 3.4.14

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.

$$q_0 = \varepsilon + q_1 \cdot 1 + q_2 \cdot 0 \quad \text{...(1)}$$

$$q_1 = q_0 \cdot 0 \quad \text{...(2)}$$

$$q_2 = q_0 \cdot 1 \quad \text{...(3)}$$

Substituting values of q_1 and q_2 from Equation (2) and (3) in Equation (1)

$$q_0 = \varepsilon + q_0 01 + q_0 10$$

$$= \varepsilon + q_0(01 + 10) \quad \text{...(4)}$$

The Equation (4) is of the form $R = Q + RP$ with $Q = \varepsilon$, $R = q_0$ and $P = (01 + 10)$

and its solution is given by $R = QP^*$

$$\therefore q_0 = \varepsilon \cdot (01 + 10)^* = (01 + 10)^*$$

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 3.4.15

Consider the following transition diagram, convert it to an equivalent regular expression using Arden's theorem.

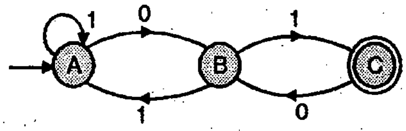


Fig. Ex. 3.4.15

Solution : The set of equations for regular sets represented by the states A, B and C are given below :

$$A = \varepsilon + A1 + B1 \quad \text{...(1)}$$

$$B = A0 + C0 \quad \text{...(2)}$$

$$C = B1 \quad \text{...(3)}$$

Rewriting Equation (1),

$A = (\varepsilon + B1) + A1$, it is of the from $R = Q + RP$ with $R = A$; $Q = (\varepsilon + B1)$ and $P = 1$ and its solution is given by $R = QP^*$

$$\therefore A = (\varepsilon + B1)1^* \quad \text{...(4)}$$

Substituting the value of A from Equation (4) and the value of C from Equation (3) in Equation (2).

$$B = (\varepsilon + B1)1^*0 + B10$$

$$= 1^*0 + B11^*0 + B10$$

$$= 1^*0 + B(11^*0 + 10)$$

From Arden's theorem,

$$B = 1^*0(11^*0 + 10)^* \quad \text{...(5)}$$

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 get represented by C describes the FA given in the example.

$\therefore$ R.E. $= 1^*0(11^*0 + 10)^*1$

Example 3.4.16 SPPU – May 16, 6 Marks

Find the regular expression for the set strings recognized by the given FA. Use Arden's theorem.

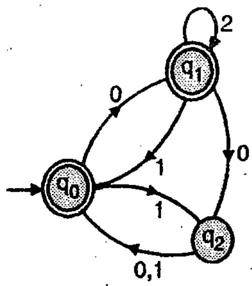


Fig. Ex. 3.4.16

Solution : The set of equations for regular sets represented by the state q_0, q_1 and q_2 are given below.

$$q_0 = \varepsilon + q_1 1 + q_2(0 + 1) \quad \ldots(1)$$
$$q_1 = q_1 2 + q_0 0 \quad \ldots(2)$$
$$q_2 = q_1 0 + q_0 1 \quad \ldots(3)$$

Substituting q_2 from Equation (3) in Equation (1)

$$q_0 = \varepsilon + q_1 1 + (q_1 0 + q_0 1)(0 + 1)$$
$$= \varepsilon + q_1 1 + q_1 00 + q_1 01 + q_0 1\,(0 + 1)$$
$$= \varepsilon + q_1(1 + 00 + 01) + q_0 1\,(0 + 1)$$

From Arden's theorem,

$$q_0 = [\varepsilon + q_1(1 + 00 + 01)]\,(1(0 + 1))^* \quad \ldots(4)$$

Putting the value of q_0 from Equation (4) in Equation (2).

$$q_1 = q_1 2 + (\varepsilon + q_1(1 + 00 + 01)]\,(1(0 + 1))^*0$$
$$= (1(0 + 1))^*0 + q_1[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]$$

From Arden's theorem,

$$q_1 = (1 + (0 + 1))^*0[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]^* \quad \ldots(5)$$

Putting the value of q_1 from Equation (5) in Equation (4)

$$q_0 = [\varepsilon + (1\,(0 + 1))^*0[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]^*(1 + 00 + 01)]\,(1(0 + 1))^* \quad \ldots(6)$$

The R.E. describing the machine is given by,

R.E. $= q_0 + q_1$ [both q_0 and q_1 are final states]

$$= [\varepsilon + (1\,(0 + 1))^*0[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]^*\,(1 + 00 + 01)]\,(1(0 + 1))^* + (1(0 + 1))^*\,0\,[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]$$

Example 3.4.17

Consider the following transition diagram. Convert it to the equivalent regular expression.

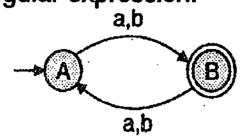
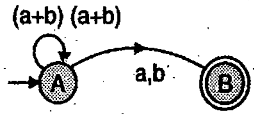


Fig. Ex. 3.4.17

Solution : Method 1

We can use the direct method to write the equivalent regular expression

- The effect of loop between the states **A** and **B** can be transferred to the state **A**.



- The required regular expression is

$[(a + b)\,(a + b)]^*(a + b)$

Method 2

We can use the Arden's theorem to write the equivalent expression.

Step 1 : The set of equations for regular sets represented by the states **A** and **B** are given below.

$$A = \epsilon + B\,(a + b) \quad \ldots(1)$$
$$B = A\,(a + b) \quad \ldots(2)$$

Step 2 : Substituting B form equation (2) in Equation (1),

$$A = \epsilon + A\,(a + b)\,(a + b)$$

From Arden's theorem,

$$A = \epsilon\,[(a + b)\,(a + b)]^*$$
$$= [(a + b)\,(a + b)]^* \quad \ldots(3)$$

Form the equation (2) and (3),

$$B = [(a + b)\,(a + b)]^*\,(a + b)$$

Required R.E. $= [(a + b)\,(a + b)]^*\,(a + b)$

Example 3.4.18 SPPU - May 15, 6 Marks

Convert the following finite automation into its equivalent regular expression using Arden's Theorem.

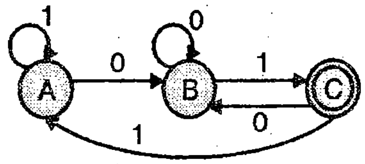


Fig. Ex. 3.4.18(a)

Solution : The set of equations for regular sets represented by the states A, B and C are given below.

$$A = \in + A1 + C1 \quad \ldots(1)$$
$$B = A0 + B0 + C0 \quad \ldots(2)$$
$$C = B1 \quad \ldots(3)$$

Replacing **C** with **B1** in Equations (1) and (2), we get

$$A = \in + A1 + B11 \quad \ldots(4)$$
$$B = A0 + B0 + B10$$
$$= A0 + B\,(0 + 10) \quad \ldots(5)$$

Applying Arden's theorem on Equation (5), we get

$$B = A0\,(0 + 10)^* \quad \ldots(6)$$

Substituting the value of B in Equation (4) we get

$$A = \in + A1 + A0\,(0 + 10)^*11$$
$$= \in + A\,[1 + 0\,(0 + 10)^*11]$$

Applying Arden's theorem, we get

$$A = [1 + 0\,(0 + 10)^*\,11]^* \quad \ldots(7)$$

Hence, **B** from equation (6) will be

$$B = [1 + 0\,(0 + 10)^*\,11]^*\,0\,(0 + 10)^*$$

∴ **C** from Equation (3) will be

$$C = [1 + 0\,(0 + 10)^*\,11]^*\,0(0 + 10)^*\,1$$

∴ Required regular expression

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

Example 3.4.19 SPPU - Dec. 15. 6 Marks

Find the regular expression for the set of strings recognized by the given FA. Use Arden's theorem.

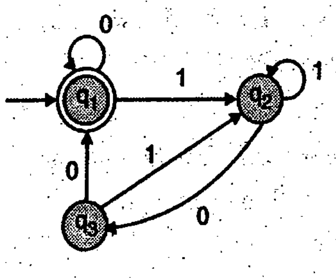


Fig. Ex. 3.4.19

Solution : The set of equations for regular sets represented by the states q_0, q_1 and q_2 are given below:

$$q_1 = \in + q_1 0 + q_3 0 \quad \ldots(1)$$
$$q_2 = q_1 1 + q_2 1 + q_3 1 \quad \ldots(2)$$
$$q_3 = q_2 0 \quad \ldots(3)$$

Substituting the value of q_3 in Equations (1) and (2), we get

$$q_1 = \in + q_0 + q_2 00 \quad \ldots(4)$$
$$q_2 = q_1 1 + q_2 1 + q_2 01$$
$$= q_1 1 + q_2\,(1 + 01) \quad \ldots(5)$$

∴ From Arden's theorem

$$q_2 = q_1\,(1 + 01)^* \quad \ldots(6)$$

Substituting the value of q_1 from the Equation (6) in the Equation (4), we get

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

From Arden's theorem

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

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 Regular Languages

3.6 Pumping Lemma for Regular Languages

SPPU - May 13. Dec. 13. May 14

University Questions

Q. State the pumping lemma for regular sets. (SPPU - May 2013, Dec. 2013, 2 Marks)

Q. Write a short note on pumping lemma. (SPPU - May 2014, 4 Marks)

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 a 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

SPPU - May 16

University Question

Q. Define Pumping Lemma. (May 2016, 3 Marks)

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

(i) $|y| > 0$ (ii) $|xy| \leq n$

(iii) $xy^i z \in L$ for all $i \geq 0$ here y^i denotes that y is repeated or pumped i times.

3.6.2 Interpretation of Pumping Lemma

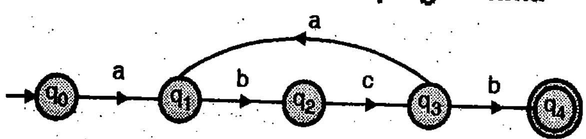


Fig. 3.6.1(a) : FA considered for interpretation of pumping lemma

Let us consider the FA of Fig. 3.6.1(a)

No. of states $= 5$ (q_0 to q_4)

Let us take a string ω with $|\omega| \geq 5$, recognized by the FA.

ω = abcabcb

To recognize the string ω = abcabcb, the machine will transit through various states as given in Fig. 3.6.1(b).

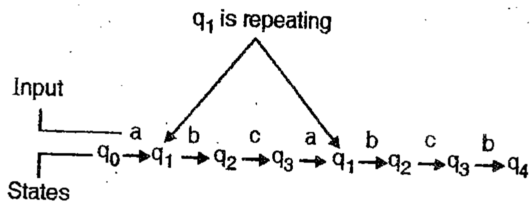


Fig. 3.6.1(b) : Transitions of FA on input abcabcb

As the input abcabcb takes the machine through the loop $q_1 \to q_2 \to q_3 \to q_1$, 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(a). The portion bca is input during the loop.

$q_1 \to q_2 \to q_3 \to q_4$.

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^iz 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(a).

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 $\omega \in L \mid |\omega| \geq n$.

   String ω can be written as

   $a_1 a_2 a_3 \ldots a_m$ with $m \geq n$.
4. Let us assume that states of M are given by $q_0, q_1, q_2, \ldots, q_{n-1}$ 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 $w = a_1 a_2 \ldots a_m$, maching will be in a state r_i.

   $\delta^*(q_0, a_1 a_2, \ldots a_i) = r_i$
6. As the word w is fed through the machine M, the machine will go through the various states as given below :

q_0	r_1	r_2	$\ldots r_{m-1}$	q_{n-1}	$\leftarrow$ state
a_1	a_2	$\ldots a_{m-1}$	a_m		$\leftarrow$ 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 w can be divided into three parts.
   1. x = Portion before loop = $a_1 a_2 \dots a_i$
   2. y = Portion of loop = $a_{i+1} a_{i+2} \dots a_j$
   3. z = 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. 3.6.2.

$$x = a_1 a_2 \dots a_i$$
$$y = a_{i+1} \dots a_j$$
$$z = a_{j+1} \dots a_m$$

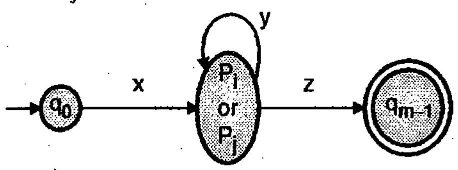


Fig. 3.6.2 : Behaviours of machine M on input xy^iz for $i \geq 0$

10. From the Fig. 3.6.2 it is clear that if $xyz \in L$ then xy^iz will also the accepted by the machine for every $i \geq 0$

3.6.4 Applications 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 $|\omega| \geq n$.

— **which 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 $xy^iz \notin L$. This will contradict the assumption that L is regular and hence we can conclude that L is not regular.

— **which i to select such that $xy^iz \notin L$ 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

$$|y| > 0$$

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 the form a^s.

Now, $a^n b^n$ can be written as

$$\underbrace{a^s}_{x}\underbrace{a^r}_{y}\underbrace{a^{n-s-r} b^n}_{z}$$

Step 3 : Let us check whether xy^iz for i = 2 belongs to L.

$$xy^2z = a^s (a^r)^2 a^{n-s-r} b^n$$
$$= a^s a^{2r} a^{n-s-r} b^n$$
$$= a^{s+2r+n-s-r} b^n$$
$$= a^{n+r} b^n$$

Since r > 0, number of a's in $a^{n+r} b^n$ is greater than number of b's. Therefore $xy^2z \notin L$. 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

$$p + q > n \quad \text{and} \quad p = q + 1$$

Let us write w as xyz with

$$|y| > 0$$

and $|xy| \leq n$

y could take any of the given forms :

1. a^i .... case I
2. $a^i b^j$ ... case II
3. b^j ... case III

Step 3 : We want to find i so that $xy^iz \notin L$ 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.

$xy^0z \notin L$

Case II

xy^2z for $i = 2$ is given by

$$\begin{aligned} & a^{p-i}(a^i b^j)^2 b^{q-j} \\ &= a^{p-i} a^i b^j a^i b^j b^{q-j} \\ &= a^p b^j a^i b^q \notin L \end{aligned}$$

Case III

$$\begin{aligned} xy^nz &= a^p (b^j)^n b^{q-j} \\ &= a^p b^{q-j+nj} \\ &= a^p b^{q+(n-1)j} \end{aligned}$$

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

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

$|y| > 0$

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

$$\underbrace{a^s}_{x}\ \underbrace{a^r}_{y}\ \underbrace{a^{n-s-r}b^{2n}}_{z}$$

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 contraction, we can say that the given language is not regular.

Example 3.6.4 SPPU - May 16, 3 Marks

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.

$$\begin{aligned} \omega &= 0^n 1^m 0^{m+n} \\ |\omega| &= 2(m+n) \geq n \end{aligned}$$

Let us write ω as xyz with

$|y| > 0$

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

$$\underbrace{0^s}_{x}\ \underbrace{0^r}_{y}\ \underbrace{0^{n-r-s}1^m0^{m+n}}_{z}$$

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 $r > 0$, $0^{n+r}1^m0^{m+n}$ is not of the form $0^i1^j0^{i+j}$

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

$$\omega = \underbrace{a^n b}_{\omega}\ \underbrace{ba^n}_{\omega^R} \longleftarrow \text{from } \omega\omega^R$$

$|\omega| = 2n + 2 \geq n$

Let us write w as xyz with

$|y| > 0$

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 $xy^i z$ 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 SPPU - Dec. 12, 6 Marks

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 a n states.

Step 2 : Let us choose a string

$$\omega = \underbrace{a^n b}_{\omega}\ \underbrace{a^n b}_{\omega} \longleftarrow \text{from } \omega\omega$$

$|\omega| = 2n + 2 \geq n$

Let us write w as xyz with

$|y| > 0$

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,

$$\omega = a^n ba^n b = \underbrace{a^s}_{x}\ \underbrace{a^r}_{y}\ \underbrace{a^{n-s-r}ba^n b}_{z}$$

Step 3 : Let us check whether xy^iz for i = 2 belongs to L.

$xy^2z = a^s a^{2r} a^{n-s-r} ba^n b$

$= a^{n+r} ba^n b$

Since $r > 0$, $a^{n+r}ba^nb$ 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 SPPU - May 12, Dec. 13, 8 Marks

Using pumping lemma for regular sets, prove that the language,

$L = \{a^n \mid n \text{ is a prime}\}$ 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

$|y| > 0$

and $|xy| \leq n$

We can assume that $y = a^m$ for $m > 0$.

Step 3 : Length of xy^iz can be written as given below:

$|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m$

as $|y| = |a^m| = m$

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 3.6.8 SPPU - Dec. 12, 6 Marks

Using pumping lemma for regular sets, prove that the language,

$L = \{a^{i^2} \mid i \geq 1\}$ 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

$$|y| > 0$$

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.

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

$$|xy^2z| = |xyz| + |y|$$

$= n^2 + (>0)$ as the length of $y > 0$ and length of xyz is n^2.

$$\therefore |xy^2z| > n^2 \quad \ldots(1)$$

again, $|xy^2z| = |xyz| + |y|$

Since the length of $|y| \leq n$ as $|xy| \leq n$ we can say that

$$|xy^2z| \leq n^2 + n$$

or, $|xy^2z| < n^2 + n + n + 1$ [n + 1 is added on the right of the inequality]

or, $|xy^2z| < (n+1)^2 \quad \ldots(2)$

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 3.6.9

Is $L = \{a^{2n} \mid n \geq 1\}$ 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. 3.6.9.

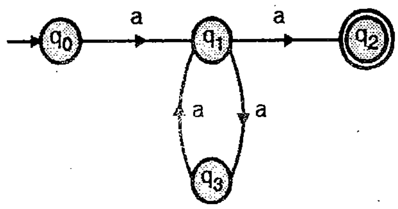


Fig. Ex. 3.6.9 : FA of example 3.6.9

Syllabus Topic : Closure Properties of Regular Language

3.7 Closure Properties of Regular Language SPPU - May 13

University Question

Q. State and explain in detail the closure properties of regular sets. **(May 2013, 8 Marks)**

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.

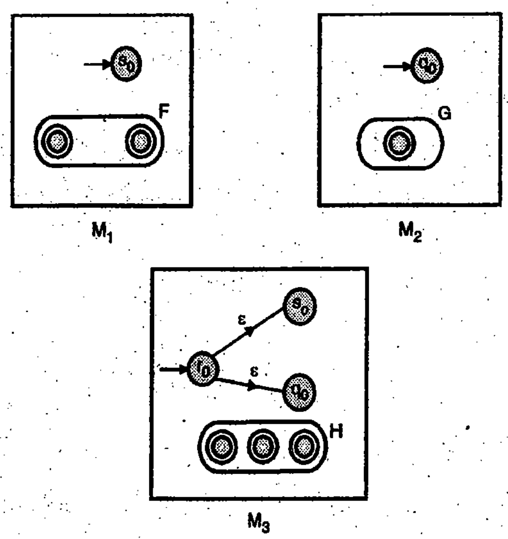


Fig. 3.7.1 : M_3 is constructed such the $L(M_3) = L(M_1) \cup (M_2)$

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.

$$R = S \cup Q \cup \{r_0\}$$

$$H = F \cup G$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(r_0, \varepsilon, s_0), (r_0, \varepsilon, q_0)\}$$

Machine M_3 can not 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.

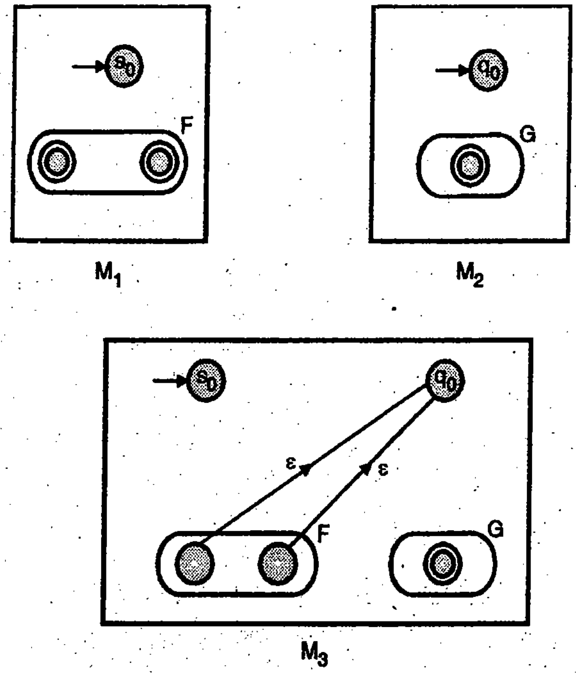


Fig. 3.7.2 : M_3 is constructed such that $L(M_3) = L(M_1) \cdot L(M_2)$

M_3 is constructed by adding ε-move from every final state of M_1 to start state of M_2.

Machine M_3 is given by :

$$M_3 = (R, \Sigma, \delta_3, s_0, G) \text{ where}$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{\varepsilon\text{-move from every final state of } M_1 \text{ to start state of } M_2\}$$

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.

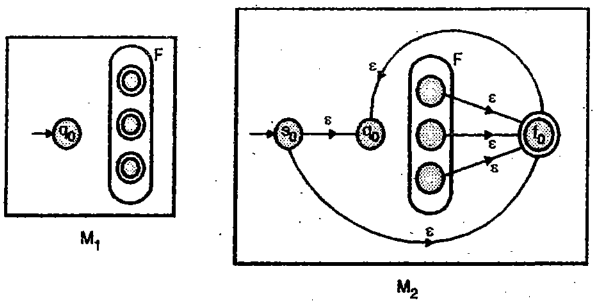


Fig. 3.7.3 : M_2 is constructed such that $L(M_2) = L(M_1)^*$

M_2 is constructed as given below :

(a) A new start state s_0 is added with an ε-move from s_0 to q_0.

(b) 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 $L(M_1)^*$.

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 $\overline{M}$ which accepts $L(\overline{M})$ where

$$\overline{L(M)} = L(\overline{M}) = \Sigma^* - L(M)$$

Given machine — Machine after complementation

If M is a deterministic finite automata then $\overline{M}$ can be constructed by interchanging final and non final states of M.

$$\therefore \quad \overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

3.7.5 Regular Language is Closed under Intersection

If L_1 and L_2 are two regular languages, then

$$L_1 \cap L_2 = ((L_1 \cap L_2)')' = (\bar{L}_1 \cup \bar{L}_2)'$$
$$= \Sigma^* - [(\Sigma^* - L_1) \cup (\Sigma^* - L_2)]$$

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 Section 2.3. 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,

if $L = \{aab, abb, aaa\}$, then

$L^R = \{baa, bba, aaa\}$

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 $L(M_1)^R$.

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 :

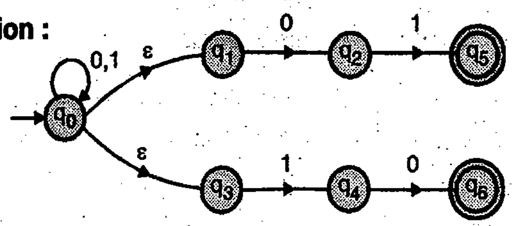


Fig. Ex. 3.7.1(a) : An NFA accepting strings ending in 01 or 10

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).

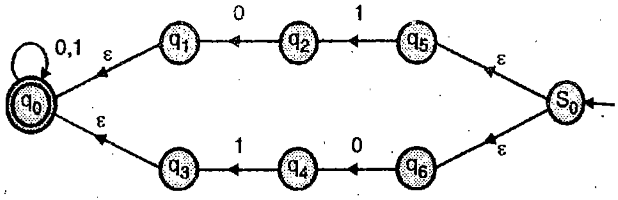


Fig. Ex. 3.7.1(b) : An NFA accepting L^R

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 as empty language then it will have no proper subset. Hence the above argument is not true for an empty language.

if $L = \phi$ then it has no proper subset

(2) If L is $\{\varepsilon\}$ then ϕ is its proper subset. A language $L = \phi$ is a regular language.

(3) For any other language, let us consider a word $\omega_1 \in L$ where ω_1 is given by $a_1 a_2 \ldots a_n$, 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 is not regular.

$L_b = \{\omega \mid \omega \in \{a, b\}^*\}$ L_b is regular.

$L_c = \{ab\}$, L_c 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_s is not regular but L_c 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 :

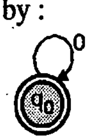


Fig. Ex. 3.7.4

L* can be considered to be regular if we can construct an FA for L*. FA for L* is given below :

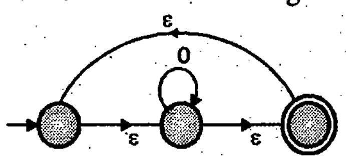


Fig. Ex. 3.7.4(a) : FA for L*

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.

$$L^+ = L L^* = (\varepsilon + L_1) L^*$$

where L_1 is a language obtained from L after deleting ε from it.

$$\therefore \quad L^+ = \varepsilon L^* + L_1 L^* = L^* + L_1 L^*$$

$$= L^*, \text{ as } L_1 L^* \subseteq L^*$$

Thus, L^+ will be equal to L^* if ε is a member of L.

Syllabus Topic : Decision Properties of Regular Language

3.8 Decision Properties of Regular Language

The finite automata can handle only simple decision problems :

1. Given a regular language L and a string w, does w belongs to L ?

   $x \in L$?

2. Given an FA M,

   L(M) $= \phi$?

3. Given an FA M,

   is L(M) finite ?

4. Given two FAs M_1 and M_2,

   is $L(M_1) = L(M_2)$?

5. Given two regular expressions r_1 and r_2, do they correspond to the same language ?
6. Given an FA M, is it a minimum state FA for the language L(M) ?

Syllabus Topic : Case Study-RE in Text Search and Replace

3.9 Application of R.E.

3.9.1 R.E. in Unix

SPPU - May 12

University Question

Q. Explain the use of regular expressions in UNIX with any one example. **(May 2012, 4 Marks)**

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

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.

Many editors, like vi in unix, has powerful search and replace capabilities. The syntax for replacing one string with another string in the current line is

: s/pattern/replace/

Here "pattern"represents the old string and "replace" represents the new string. The syntax for replacing every occurrence of a sting in the entire text is similar :

: %s/pattern/replace/

The only difference is the addition of a "%" in front of the "s".

3.9.2 Lexical Analysis

SPPU - May 13, Dec. 13, May 14

University Questions

Q. Explain the application of regular expressions in lexical analysis phase of compiler. **(May 2013, 6 Marks)**

Q. Explain the application of regular expression to lexical analysis phase of compiler with suitable example. **(Dec. 2013, 6 Marks)**

Q. Explain the application of R.E. : Finding pattern in text. **(May 2014, 2 Marks)**

Lexical analysis is an important phase of a compiler. The lexical analyser scans the source program and converts it into a steam 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.

❑❑❑

Unit III

CHAPTER 4

Context Free Grammars (CFG) and Languages

Syllabus

Introduction, Regular Grammar, **Context Free Grammar** - Definition, Derivation, Language of grammar, sentential form, parse tree, inference, derivation, parse trees, ambiguity in grammar and Language-ambiguous Grammar, **Simplification of CFG:** Eliminating unit productions, useless production, useless symbols, and Є-productions, **Normal Forms** - Chomsky normal form, Greibach normal form, Closure properties of CFL, Decision properties of CFL, Chomsky Hierarchy, **Application of CFG :** Parser, Markup languages, XML and Document Type Definitions. Case Study - CFG for Palindromes, Parenthesis Match.

Syllabus Topic : Introduction

4.1 An Example to Explain Grammar

- 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. $L1 = \{a^p \mid p \text{ is a prime}\}$ is not regular.
  2. $L2 = \{a^n b^n \mid n \geq 0\}$ is not regular.
  3. $L3 = \{a^{i^2} \mid i \geq 1\}$ is not regular.
  4. $L4 = \{ww \mid w \in \{a, b\}^*\}$ is not regular.
  5. $L5 = \{ww^R \mid w \in \{a, b\}^*\}$ is not regular.
- We have seen two ways for representing a language :
  1. Using a 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).

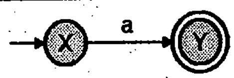
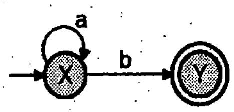
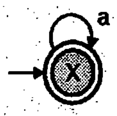
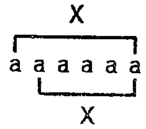
Sr. No.	DFA	Equivalent grammar
1.		$X \rightarrow a$
2.		$X \rightarrow aX$ $X \rightarrow b$
3.		$X \rightarrow aX$ $X \rightarrow \varepsilon$

Fig. 4.1.1 : A set of DFA represented using an equivalent set of productions

- 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. 4.1.2.



A string of 6 a's can be generated by X
A string of 5 a's can be generated by X.

Fig. 4.1.2 : Recursive definition of $X \rightarrow aX$

$\underbrace{\text{A string of 6 a's}}_{\text{Or}\ \ X} = \text{a followed by}\ \underbrace{\text{a string of 5 a's.}}_{aX}$

Every recursive definition has a base case or termination case.

Let us consider the DFA given below in Fig. 4.1.3.

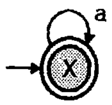


Fig. 4.1.3 : DFA taken as an example

The language of the DFA is given below

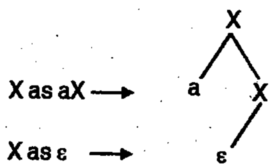
$$L = \{\varepsilon, a, aa, aaa, \ldots..\}$$

Now, let us consider the grammar given below and see if it can generate the language generated by the DFA of Fig. 4.1.3.

$$X \rightarrow aX \qquad \ldots(4.1.1)$$
$$X \rightarrow \varepsilon \qquad \ldots(4.1.2)$$

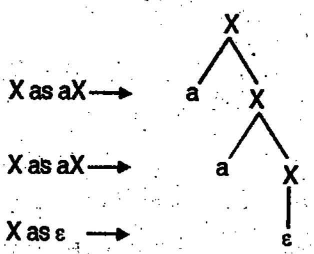
- The empty string $\in$ can be generated by the production $X \rightarrow \varepsilon$
- A string "a" can be generated as shown below.

First by writing X as aX and then by replacing X on the right hand side with an ε.



- A string "aa" can be generated as shown below.

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 $\{X \rightarrow aX, X \rightarrow \varepsilon\}$ is

$$L = \{\varepsilon, a, aa, \ldots\}$$

The production $X \rightarrow \varepsilon$, is for termination of recursion.

A non regular language can be expressed using a set of productions

Let us consider a language,

$$L = \{0^n1^n \mid n \geq 1\}$$

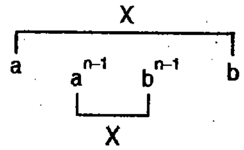
We know from pumping lemma that the above language is not regular. But this language can be represented using a set of productions. It will be worthwhile to understand the recursive nature of the above language.

1. A string $0, 1 \in L$

   [Base case or termination case]

2. a^nb^n can be written as $a(a^{n-1}b^{n-1})b$

   If a^nb^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^nb^n, after removing the first a and the last b, the string remains in the form $a^n b^n$.

Thus the language.

$L = \{a^nb^n \mid n \geq 1\}$ can be represented using

1. $X \rightarrow aXb \qquad \ldots(4.1.3)$
2. $X \rightarrow ab \qquad \ldots(4.1.4)$

The production $X \rightarrow aXb$ can recursively generate a string a^nb^n and the second production $X \rightarrow ab$ is used for termination of recursion.

Let us try to generate a string a^3b^3 using the two productions (4.1.3) and (4.1.4).

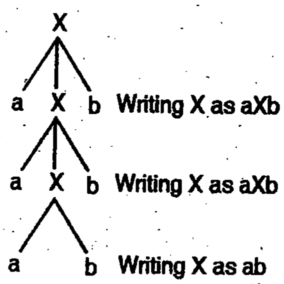


Fig. 4.1.4 : Generation of a^3b^3

Conclusion : Grammar can be used to represent both.

1. Regular languages
2. Non-regular languages.

Syllabus Topic : Context Free Grammar : Definition

4.2 Context Free Grammar

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.

4.2.1 Notations

1. Terminals are denoted by lower case letters a, b, c … or digits 0, 1, 2 …. etc.
2. Non-terminals (variables) are denoted by capital letters A, B, …, V, W, X …
3. A string of terminals or a word $w \in L$ is represented using u, v, w, x, y, z.
4. 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 :

1. Mohan eats.
2. Sohan plays.
3. 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

<sentence> $\rightarrow$ <noun> <verb>

Here, noun can be replaced with Mohan, Sohan or Ram and <verb> can be replaced with reads, plays or eats. We can write :

<noun> $\rightarrow$ Mohan

<noun> $\rightarrow$ Soham

<noun> $\rightarrow$ Ram

<verb> $\rightarrow$ eats

<verb> $\rightarrow$ plays

<verb> $\rightarrow$ reads

In the above example :

1. <sentence>, <noun> and <verb> are variables or non-terminals.
2. Mohan, Soham, Ram, eats, plays and reads are terminals.
3. The variable <sentence> is the start symbol as a sentence will be formed using the start symbol <sentence>.

Formally, the grammar can be written as :

G = (V, T, P, S), where

V = {<sentence>, <noun>, <verb>}

T = {Mohan, Soham, Ram, eats, plays, reads}

P = {<sentence> $\rightarrow$ <noun> <verb>,

<noun> $\rightarrow$ Mohan,

<noun> $\rightarrow$ Soham,

<noun> $\rightarrow$ Ram,

<verb> $\rightarrow$ eats,

<verb> $\rightarrow$ plays,

<verb> $\rightarrow$ reads

}

S = <sentence>

Several productions of <noun> and <verb> can be merged together and the set of productions can be re-written as :

P = {<sentence> $\rightarrow$ <noun> <verb>,

<noun> $\rightarrow$ Mohan | Soham | Ram,

<verb> $\rightarrow$ eats | plays | reads

}

Syllabus Topic : Language of Grammers

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

$$L(G) = \left\{ w \mid w \in T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$$

If the production rule is applied once, then we write $\alpha \underset{G}{\Rightarrow} \beta$. When it is applied a number of times then we write $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$

Derivations are represented either in the

1. Sentential form OR
2. Parse tree form.

Syllabus Topic : Sentential Form

4.2.2.1 Sentential Form

Let us consider a grammar given below :

$S \rightarrow A1B$ (Production 4-5)

$A \rightarrow 0A \mid \varepsilon$ (Production 4-6)

$B \rightarrow 0B \mid 1B \mid \varepsilon$ (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.

$S \rightarrow A1B$ [Starting production]

$\rightarrow 0A1B$ [Using the production $A \rightarrow 0A$]

$\rightarrow 00A1B$ [Using the production $A \rightarrow 0A$]

$\rightarrow 001B$ [Using the production $A \rightarrow \varepsilon$]

$\rightarrow 0010B$ [Using the production $B \rightarrow 0B$]

$\rightarrow 00101B$ [Using the production $B \rightarrow 1B$]

$\rightarrow 00101$ [Using the production $B \rightarrow \varepsilon$]

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.

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \mid \alpha \in (V \cup T)^*$$

- 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

$S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Give leftmost and rightmost derivation of the string 1001.

Solution :

(i) Leftmost derivation of 1001 (Leftmost variable is picked up for expansion.)

$S \rightarrow A1B$ [Derivation starts from the start symbol]

$\rightarrow 1B$ [Using the production $A \rightarrow \varepsilon$]

$\rightarrow 10B$ [Using the production $B \rightarrow 0B$]

$\rightarrow 100B$ [Using the production $B \rightarrow 0B$]

$\rightarrow 1001B$ [Using the production $B \rightarrow 1B$]

$\rightarrow 1001$ [Using the production $B \rightarrow \varepsilon$]

(ii) Rightmost derivation of 1001 (Rightmost variable is picked up for expansion)

$S \rightarrow A1B$ [Derivation starts from the start symbol]

$\rightarrow A10B$ [Using the production $B \rightarrow 0B$]

$\rightarrow A100B$ [Using the production $B \rightarrow 0B$]

$\rightarrow A1001B$ [Using the production $B \rightarrow 1B$]

$\rightarrow A1001$ [Using the production $B \rightarrow \varepsilon$]

$\rightarrow 1001$ [Using the production $A \rightarrow \varepsilon$]

Syllabus Topic : Parse Trees, Derivation

4.2.2.2 Parse Tree

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

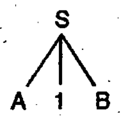
For the grammar given below

S → A1B A → 0A | ε B → 0B | 1B | ε

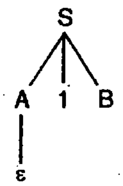
Give parse tree for leftmost and rightmost derivation of the string 1001.

Solution :

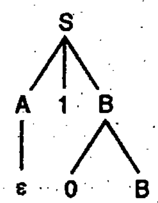
(i) Parse tree for leftmost derivation of 1001.



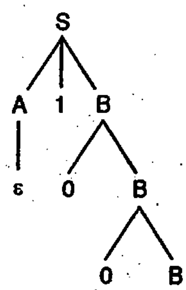
(a) S → A 1 B



(b) A → ε



(c) B → 0B



(d) B → 0B

Fig. Ex. 4.2.2

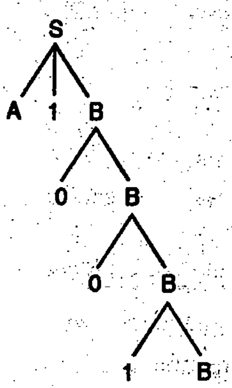


Fig. Ex. 4.2.2(e) : B → 1B

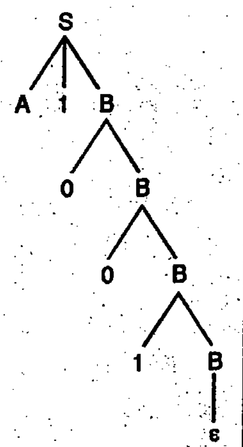


Fig. Ex. 4.2.2(f) : B → ε

Leftmost derivation of 1001 is shown by series of Figs. Ex. 4.2.2(a) to Ex. 4.2.2(f).

(ii) Parse tree for rightmost derivation of 1001.

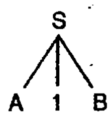


Fig. Ex. 4.2.2(g) : S → A 1 B

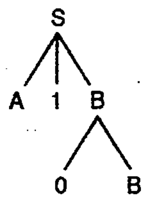


Fig. Ex. 4.2.2(h) : B → 0B

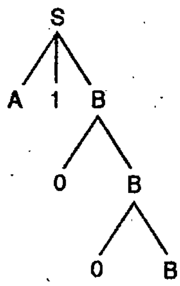


Fig. Ex. 4.2.2(i) : B → 0B

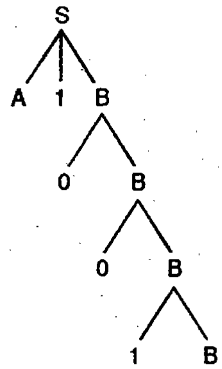


Fig. Ex. 4.2.2(j) : B → 1B

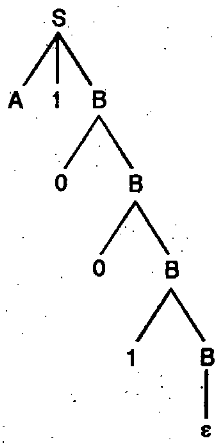


Fig. Ex. 4.2.2(k) : B → ε

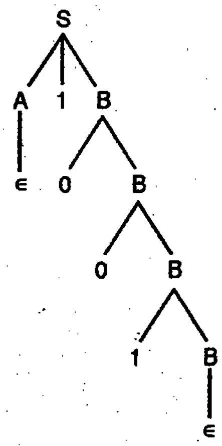


Fig. Ex. 4.2.2(*l*) : A → ε

Rightmost derivation of 1001 is shown by a series of Fig. Ex. 4.2.2(g) to Fig. Ex. 4.2.2 (*l*).

It may be noted that the final parse trees Fig. Ex. 4.2.2(f) and Fig. Ex. 4.2.2(*l*) are identical.

Example 4.2.3

For the grammar given below S → 0S1 | 01

Give derivation of 000111.

Solution :

1. Derivation in sentential form

S → 0S1 [Using production S → 0S1]

S → 00S11 [Using production S → 0S1]

S → 000111 [Using production S → 01]

2. Derivation using parse tree.

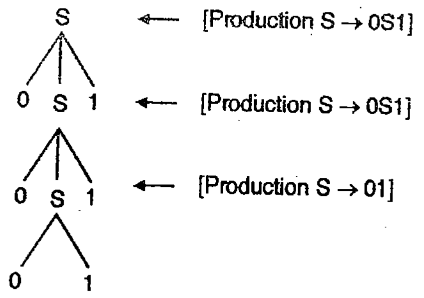


Fig. Ex. 4.2.3 : Derivation of 000111

Example 4.2.4

Find whether the string aabbb is in $L = L(G)$, where G is given by $S \rightarrow XY$, $X \rightarrow YY|a$, $Y \rightarrow XY \mid b$

Solution :

Parse tree for the string aabbb is given below.

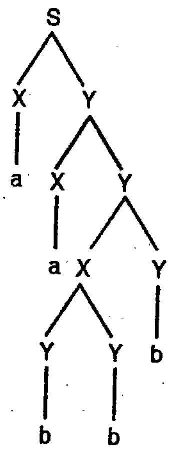


Fig. Ex. 4.2.4 : Derivation of aabbb

Hence, aabbb $\in$ L(G).

Example 4.2.5

For the grammar given below

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a \mid b$

Give derivation of $(a + b) * a + b$.

Solution :

For the grammar given in the example,

Set of variables V = {E, T, F}

Set of terminals Σ = {+, *, (,), 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

$E \rightarrow$	$E + T$	[Using production $E \rightarrow E + T$]
$\rightarrow$	$T + T$	[Using production $E \rightarrow T$]
$\rightarrow$	$T * F + T$	[Using production $T \rightarrow T * F$]
$\rightarrow$	$F * F + T$	[Using production $T \rightarrow F$]
$\rightarrow$	$(E) * F + T$	[Using production $F \rightarrow (E)$]
$\rightarrow$	$(E + T) * F + T$	[Using production $E \rightarrow E + T$]
$\rightarrow$	$(T + T) * F + T$	[Using production $E \rightarrow T$]
$\rightarrow$	$(F + F) * F + F$	[Using production $T \rightarrow F$]
$\rightarrow$	$(a + b) * a + b$	[Using production $F \rightarrow a \mid b$]

(ii) Derivation using parse tree :

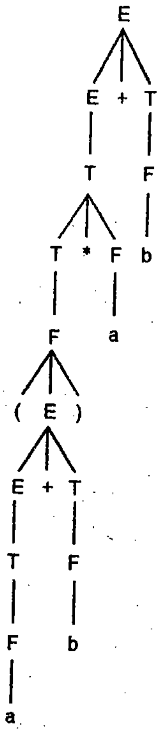


Fig. Ex. 4.2.5 : Parse tree for (a + b) * a + b

Syllabus Topic : Case Study - CFG for Palindromes

4.2.3 Writing Grammar for a Language

SPPU - May 12, May 13

University Questions

Q. Write a grammar G for generating the language $L = \{W \in \{a, b\}^* \mid w$ is an even length palindrome with $|w| > 0\}$. **(May 2012, 8 Marks)**

Q. Construct a grammar G to represent a language L which is a set of all palindromes over {a,b}. **(May 2013, 4 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 $L = \{\varepsilon, a, aa, \ldots\}$

The productions required to generate the above language are :

$$S \rightarrow aS$$
$$S \rightarrow \varepsilon$$

The production, $S \rightarrow \varepsilon$ 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 $S \rightarrow \varepsilon$]

Parse tree

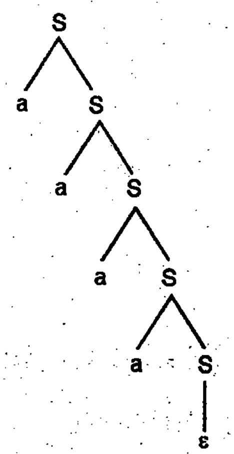


Fig. 4.2.1 : Parse tree for aaaa

(2) Language $L = \{a, aa, aaa, \ldots\}$

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

$S \rightarrow aS$ [Using production $S \rightarrow aS$]
$\rightarrow aaS$ [Using production $S \rightarrow aS$]
$\rightarrow aaaS$ [Using production $S \rightarrow aS$]
$\rightarrow aaaa$ [Using production $S \rightarrow aS$]

Parse tree

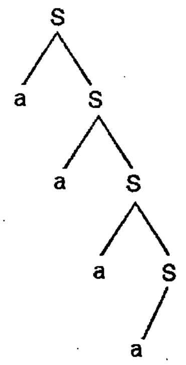


Fig. 4.2.2 : Parse tree for aaaa

(3) Language $L = \{b, ab, aab, aaab, \ldots\}$

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

$S \rightarrow aS$ [Using production $S \rightarrow aS$]
$\rightarrow aaS$ [Using production $S \rightarrow aS$]
$\rightarrow aaaS$ [Using production $S \rightarrow aS$]
$\rightarrow aaab$ [Using production $S \rightarrow b$]

Parse tree

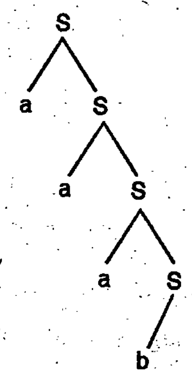


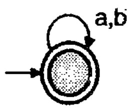
Fig. 4.2.3 : Parse tree for aaab

(4) Language $L = \{w \in \{a, b\}^*\}$

The above language can also be written as

$L = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$

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 :

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow \varepsilon$

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 $S \rightarrow \varepsilon$ is for termination of recursion. Since ε is also a member of L, $S \rightarrow \varepsilon$ is taken for termination of recursion.

Example : Generation of the string abba

Sentential form

S →	aS	[Using production S → aS]
→	abS	[Using production S → bS]
→	abbS	[Using production S → bS]
→	abbaS	[Using production S → aS]
→	abba	[Using production S → ε]

Parse tree

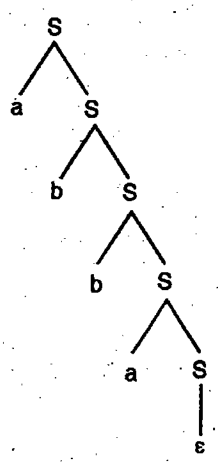


Fig. 4.2.4 : Parse tree for abba

(5) Language $L = \{\varepsilon, ab, aabb, \ldots, a^n b^n\}$

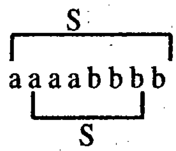
The above language can also be described as a language of words in which every word consists of any number of a's followed by equal number of b's.

The productions required to generate the above language are :

$S \rightarrow aSb$

$S \rightarrow \varepsilon$

The recursion involved in the above language is explained below.



Both the strings a^4b^4 and a^3b^3 are generated by S. a^4b^4 can be written as aa^3b^3b 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 $S \rightarrow \varepsilon$ is for termination of recursion. Since ε is a member of the given language, $S \rightarrow \varepsilon$ is taken for termination of recursion.

Example : Generation of the string aaabbb.

Sentential form

S →	aSb	[Using production S → aSb]
→	aaSbb	[Using production S → aSb]
→	aaaSbbb	[Using production S → aSb]
→	aaabbb	[Using production S → ε]

Parse tree

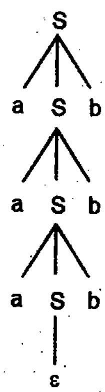


Fig. 4.2.5 : Parse tree for aaabbb or a^3b^3

(6) Language $L = \{ab, aabb, \ldots, a^n b^n\}$

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^nb^n. The production $S \rightarrow ab$ is for termination of recursion.

Example : Generation of the string aaabbb

Sentential form

$S \rightarrow$	aSb	[Using production $S \rightarrow aSb$]
$\rightarrow$	aaSbb	[Using production $S \rightarrow aSb$]
$\rightarrow$	aaabbb	[Using production $S \rightarrow ab$]

Parse tree

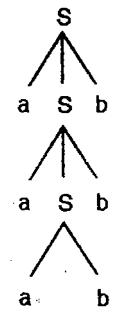
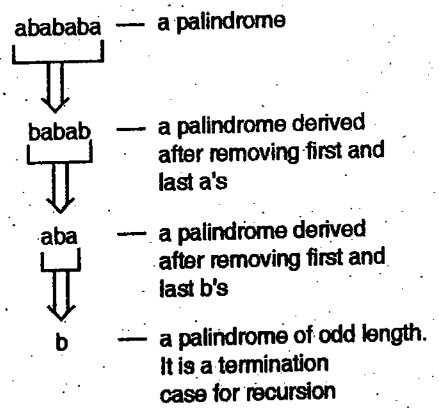


Fig. 4.2.6 : Parse tree for aaabbb or a^3b^3

(7) Language L = {w∈ {a, b}* | w is a palindrome of odd length}

- String 'a' is palindrome of odd length.
- String 'b' is a palindrome of odd length
- String abababa is palindrome. The palindrome nature of abababa can be understood as given below.



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 :

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a$

$S \rightarrow b$

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

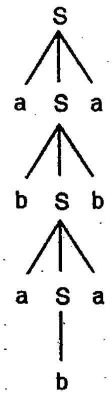


Fig. 4.2.7 : Parse tree for abababa

(8) Language L = {w∈ {a, b}* | w is an even length palindrome with | w | > 0}

SPPU - May 12

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

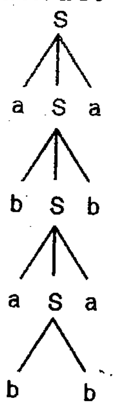


Fig. 4.2.8 : Derivation tree for ababbaba

(9) Language $L = \{w \in \{a, b\}^* \mid w$ is a palindrome of either even length or odd length with $|w| > 0\}$

SPPU - May 13

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 :

$S_1 \rightarrow aS \mid a$

Productions for L_2 are :

$S_2 \rightarrow bS \mid b$

Then the productions for $L = L_1 \cup L_2$ can be written as :

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow aS \mid a$

$S_2 \rightarrow bS \mid b$

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

$S \rightarrow S_1S_2$

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 :

$S_1 = aS \mid a$

Productions for L_2 are :

$S_2 \rightarrow bS \mid b$

Then the productions for $L = L_1 \cdot L_2$ can be written as :

$S \rightarrow S_1S_2$

$S_1 \rightarrow aS \mid a$

$S_2 \rightarrow bS \mid b$

Example 4.2.6

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.

$S_1 \rightarrow 0\,S_1 \mid 1\,S_1 \mid \varepsilon$

Using the product rule, we can write a set of productions for the language

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

$P = \{S \rightarrow 0S_101S_11$

$S_1 \rightarrow 0S_1 \mid 1S_1 \mid \varepsilon$

$\}$

Where, Set of variables $V = \{S, S_1\}$

Set of terminals $T = \{0, 1\}$

Start symbol $= S$

Example 4.2.7

Construct the context free grammar corresponding to the regular expression.

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

Solution : Grammar for (0 + 1) is given by

$S_1 \rightarrow 0 \mid 1$ [union rule]

Grammar for 1^* is given by

$S_2 \rightarrow 1S_2 \mid \varepsilon$

Grammar for $(1 + (01)^*)$ is given by

$S_3 \rightarrow 1 \mid S_4$ where S_4 stands for $(01)^*$

$S_4 \rightarrow 01S_4 \mid \varepsilon$

Using the concatenation rule, we can write a set of productions for

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

$P = \{$

$S \rightarrow S_1S_2S_3$

$S_1 \rightarrow 0 \mid 1$

$S_2 \rightarrow 1S_2 \mid \varepsilon$

$S_3 \rightarrow 1 \mid S_4$

$S_4 \rightarrow 01S_4 \mid \varepsilon$

$\}$

where, Set of variables V = $\{S_1, S_2, S_3, S_4\}$

Set of terminals T = $\{0, 1\}$

Start symbol = S

Example 4.2.8 SPPU - May 15, 4 Marks

Write a CFG that generates language L denoted by,

(a + b) *bbb(a + b) *

Solution :

The set of production for (a + b)* bbb (a + b)* is given by

$S \rightarrow A\ bbb\ A$

$A \rightarrow aA \mid bA \mid \in$

Set of variable V = {S, A}

Set of terminal T = {0, 1}

Start symbol = S

Example 4.2.9

Give the CFG for $L = \{a^ib^j, i \le j \le 2i, i \ge 1\}$

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 :

$P = \{$

$S \rightarrow aSb$ [Number of b's = Number of a's]

$S \rightarrow aSbb$ [Number of b's = 2 × Number of a's]

$S \rightarrow ab \mid abb$ [Termination cases]

$\}$

where, Set of variables V = {S}

Set of terminals T = {a, b}

Start symbol = S.

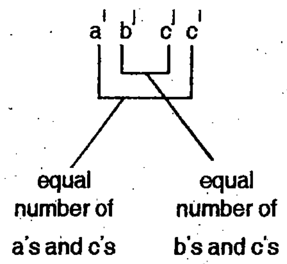
Example 4.2.10

Give CFG for

$L = \{a^ib^jc^q ; i + j = q; i, j \ge 1\}$

Solution : A string of the form $a^ib^jc^q$ with i + j = q can be written as

$a^ib^jc^{i+j}$



A string of the form a^ic^i 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^jc^j.

$X \rightarrow bXc$

$X \rightarrow bc$

Thus, the set of productions for the above language are given by :

$P = \{ \quad S \rightarrow aSc \mid X$

$X \rightarrow bXc \mid bc$

$\}$

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 aabbbccccc

Parse tree

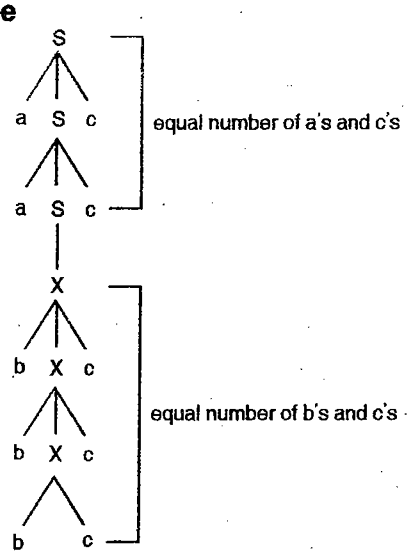


Fig. Ex. 4.2.10 : Parse tree for aabbbccccc

Example 4.2.11

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

$$\underbrace{a^i b^i}_{\text{a's and b's are equal (generated by X)}} \quad \underbrace{b^k c^k}_{\text{B's and c's are equal (generated by Y)}}$$

$S \rightarrow XY$

$X \rightarrow aXb \mid ab$

$Y \rightarrow bYc \mid bc$

(3) $L_3 = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

$S \rightarrow AB$ [i = j]

$S \rightarrow CD$ [j = k]

$A \rightarrow aAb \mid ab$ [equal number of a's and b's

$B \rightarrow CB \mid c$ followed by some c's]

$C \rightarrow aC \mid a$ [Some a's followed by equal

$D \rightarrow b\,DC \mid bc$ number of b's and c's]

Example 4.2.12 SPPU - May 14, 8 Marks

Give the context free grammar for the following languages.

(a) $(011 + 1)^* (01)^*$

(b) $0^i 1^{i+k} 0^k$ where i,k >= 0

Solution :

(a) CFG for $(011 + 1)^* (01)^*$

Let us assume that $(011 + 1)^*$ is generated by the variable X.

$\therefore$ $X \rightarrow 011X \mid 1X \mid \varepsilon$

Let us assume that $(01)^*$ is generated by the variable Y.

$\therefore$ $Y \rightarrow 01Y \mid \varepsilon$

Using the concatenation rule, the set of productions for $(011 + 1)^* (01)^*$ can be written as :

$$P = \{\ S \rightarrow XY$$
$$X \rightarrow 011X \mid 1X \mid \varepsilon$$
$$Y \rightarrow 01Y \mid \varepsilon$$
$$\}$$

Where, the set of variable V = {S,X,Y}

the set of terminals T = {0,1}

The start symbol = S

(b) CFG for $0^i 1^{i+k} 0^k \mid i,k >= 0$

The language L can be written as :

$$L = \{\underbrace{0^i 1^i}_{X}\ \underbrace{1^k 0^k}_{Y}\}$$

Let us assume that $0^i 1^i$ is generated by the variable X.

$\therefore$ $X = 0X1 \mid \varepsilon$

Let us assume that $1^k 0^k$ is generated by the variable Y.

$\therefore$ $Y = 1Y1 \mid \varepsilon$

The set of productions P is given by :

$P = \{\ S \rightarrow XY$

$X \rightarrow 0X1 \mid \varepsilon$

$Y \rightarrow 1Y0 | \varepsilon$

}

Where, The set of variable V $= \{S, X, Y\}$

The set of terminals T $= \{0, 1\}$

The start symbol $= S$

Example 4.2.13

Show that the language,

(1) $L_1 = \{a^i b^i c^j \mid i, j >= 1\}$ and

(2) $L_2 = \{a^i b^j c^j \mid i, j >= 1\}$ are context free language.

Solution :

These languages can be shown to CFL by writing the required productions in each case.

Productions for $L_1 = \{a^i b^i c^j \mid i, j >= 1\}$

$S \rightarrow XY$

$X \rightarrow aXb | ab$ [X generates $a^i b^i$ and

$Y \rightarrow cY | c$ Y generates c^j]

Productions for $L_2 = \{a^i b^j c^j \mid i, j >= 1\}$

$S \rightarrow XY$ [X generates a^i and

$X \rightarrow aX | a$ Y generates $b^j c^j$]

$Y \rightarrow bYc | bc$

Example 4.2.14 SPPU - May 15, 4 Marks

Give CFG for language $L = \{0^i 1^j 0^k \mid j > i + k\}$. Show the derivation of the string '0111100'

Solution : A string of the form $0^i 1^j 0^k$ with $j > i + k$ can be written as

$$0^i 1^{i+k+P} 0^k \text{ with } P > 0$$

$$= \underbrace{0^i 1^i}_{X} \underbrace{1^P}_{Y} \underbrace{1^k 0^k}_{Z}$$

A string $0^i 1^i 1^P 1^k 0^k$ 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 :

P = { $S \rightarrow XYZ$

$X \rightarrow 0X1 | \varepsilon$

$Y \rightarrow 1Y | 1$

$Z \rightarrow 1Z0 | \varepsilon$

}

where, Set of variables V $= \{X, Y, Z\}$

Set of terminals T $= \{0, 1\}$

Start symbol $= S$

Derivation of the string 0111100

S	$\rightarrow$ xyz	[using $s \rightarrow xyz$]
	$\rightarrow$ 0x1yz	[using $x \rightarrow 0x1$]
	$\rightarrow$ 0x1y1z0	[using $z \rightarrow 1z0$]
	$\rightarrow$ 0x1y11z00	[using $z \rightarrow 1z0$]
	$\rightarrow$ 01y1100	[using $x \rightarrow \epsilon, z \rightarrow \epsilon$]
	$\rightarrow$ 0111100	[using $y \rightarrow 1$]

Syllabus Topic : Case Study - CFG for Parenthesis Match

Example 4.2.15

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 : ((()))

Side ways : $\underbrace{(())}_{S}$ $\underbrace{(())}_{S}$

Thus, the set of productions for the above language are given by :

$P = \{S \rightarrow (S) | SS | \varepsilon\}$

where, Set of variables V $= \{S\}$

Set of terminals T $= \{(,)\}$

Start symbol $= S$

Example 4.2.16

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 :

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

Let us assume that 1^* is generated by the variable X.

$\therefore$ X $\rightarrow$ 1X | ε

Let us assume that $(0 + 1)^*$ is generated by the variable Y.

$\therefore$ Y $\rightarrow$ 0Y | 1Y | ε

Using the concatenation rule, the set of productions for $1^*01^*0(0 + 1)^*$ can be written as :

P = {
S $\rightarrow$ X0X0Y
X $\rightarrow$ 1X | ε
Y $\rightarrow$ 0Y | 1Y | ε
}

where, the set of variables V = {S, X, Y}
the set of terminals T = {0, 1}
the start symbol = S.

Example 4.2.17 SPPU - May 14, 4 Marks

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 :

$\underbrace{\text{String of length n from } \{0, 1\}^*}$ 1 $\underbrace{\text{String of length n from } \{0, 1\}^*}$

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 are given by :

P = {
S $\rightarrow$ 0S0 | 0S1 | 1S0 | 1S1 | 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 4.2.18

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 :

$\underbrace{\text{String of length n from } \{a, b, c, d\}^*}$ $\begin{bmatrix} aa \\ bb \\ cc \\ dd \end{bmatrix}$ $\underbrace{\text{String of length n from } \{a, b, c, d\}^*}$

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 | aSb |aSc | aSd |
bSa | bSb | bSc | bSd |
cSa | cSb | cSc | cSd |
dSa | dSb | dSc | dSd |
aa | bb | cc | dd
}

where, set of variables V = {S}
set of terminals T = {a, b, c, d}
start symbol = S

Example 4.2.19

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

$$a^q \underbrace{a^{m-q} b^n c^p}_{X} d^q$$

where $a^{m-q} b^n c^p$ is generated by the variable X (say).

The string X is of the form $a^i b^j c^{i+j} = a^i \underbrace{b^j c^j}_{Z} c^i$

Productions for X can be written as given below :

X $\rightarrow$ aXc | Z
Z $\rightarrow$ bZc | ε

Case II : $m < q$, $a^m b^n c^p d^q$ can be written as

$$a^m \underbrace{b^n c^p d^{q-m}}_{Y} d^m$$

where $b^n c^p d^{q-m}$ is generated by the variable Y (say).

The string Y is of the form $b^{i+j} c^i d^j = b^j \underbrace{b^i c^i}_{Z} d^j$

Productions for Y can be written as given below :

$Y \rightarrow bYd \mid Z$

$Z \rightarrow bZc \mid \varepsilon$

Case III : $m = q$, $a^m b^n c^p d^q$ can be written as

$$a^m \underbrace{b^n c^n}_{Z} d^m$$

Z(say)

where $b^n c^n$ is generated by Z (say)

$Z = bZc \mid \varepsilon$

Thus the set of productions for the above language are given by :

$$P = \{ \quad S \rightarrow aSd \mid X \mid Y$$
$$X \rightarrow aXc \mid Z$$
$$Y \rightarrow bYd \mid Z$$
$$Z \rightarrow bZc \mid \varepsilon$$
$$\}$$

where, Set of variables V = {S, X, Y, Z}

Set of terminals T = {a, b, c, d}

Start symbol = S.

Example 4.2.20

Give CFG for $L = \{a^n b^m c^n \mid m, n >= 1\}$

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 :

$$P = \{ \quad S \rightarrow aSc \mid X$$
$$X \rightarrow bX \mid b$$
$$\}$$

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 4.2.21

Give CFG for L = {x | x contains equal number of a's 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,

  $B \rightarrow bS$

and $B \rightarrow aBB$

Thus the set of productions for the above language are given by :

$$P = \{ \quad S \rightarrow aB \mid bA$$
$$A \rightarrow aS \mid bAA \mid a$$
$$B \rightarrow bS \mid aBB \mid b$$
$$\}$$

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

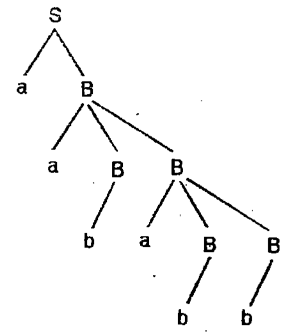


Fig. Ex. 4.2.21 : Parse tree for aababb

Example 4.2.22

Give CFG for strings in ab*

Solution : Let us say that b* is generated by variable B.

$\therefore$ B $\to$ bB | ε

The set of productions for the above language can be written using the rule of concatenations.

P = { S $\to$ aB
B $\to$ bB | ε
}

where, Set of variables V = {S, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.23

Give CFG for strings in a*b* .

Solution : Let us assume that a* and b* are generated by variable A and B respectively.

A $\to$ aA | ε
B $\to$ bB | ε

The set of productions for the above language can be written using the rule of concatenation.

P = { S $\to$ AB
A $\to$ aA | ε
B $\to$ bB | ε
}

where, Set of variables V = {S, A, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.24

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 $\to$ aSa | bSb | c

where, Set of variables V = {S}
Set of terminals T = {a, b, c}
Start symbol = S.

Example 4.2.25

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. $a^i a^j b^i$ where $j > 0$

or 2. $a^i b^j b^i$ where $j > 0$

Let us assume that a^j is generation by A and b^j is generated by B.

The set of productions for the above language is given by :

P = { S $\to$ aSb | A | B
A $\to$ aA | a
B $\to$ bB | b
}

where, Set of variables V = {S, A, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.26

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.

where, $L_1 = \{a^n b^m c^k \mid n = m\} = \{a^n b^n c^k\}$

$L_2 = \{a^n b^m c^k \mid m \leq k\} = \{a^n b^m c^{m+p} \mid p \geq 0\}$

$= \{a^n b^m c^m c^p \mid p \geq 0\}$

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 | ε

B = cB | ε

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 $b^m c^m$.

$$X \rightarrow aX \mid \varepsilon$$
$$Y \rightarrow bYc \mid \varepsilon$$

Now, we can write productions for the given language L using the rule of union.

$$L = L_1 \cup L_2$$

P = {
$$S \rightarrow S_1 \mid S_2$$
$$S_1 \rightarrow AB$$
$$A \rightarrow aAb \mid \varepsilon$$
$$B \rightarrow cB \mid \varepsilon$$
$$S_2 \rightarrow XYB$$
$$X \rightarrow aX \mid \varepsilon$$
$$Y \rightarrow bYc \mid \varepsilon$$
}

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 4.2.27

Give CFG for the language $L = \{0^m 1^n 0^{m+n} \mid m, n >= 0\}$

Solution : The language L can be written as :

$$L = \{0^m 1^n 0^{m+n}\} = \{\underbrace{0^m}_{X}\underbrace{1^n 0^n}_{Y}\underbrace{0^m}_{X}\}$$

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 = {
$$S \rightarrow XYX$$
$$X \rightarrow 0X \mid \varepsilon$$
$$Y \rightarrow 1Y0 \mid \varepsilon$$
}

where, The set of variables V = {S, X, Y}
The set of terminals T = {0, 1}
The start symbol = S.

Example 4.2.28

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 $0^{c+b} 1^b 2^c$

$$= \underbrace{0^c\ \underbrace{0^b\ 1^b}\ 2^c}$$

The set of productions for the given language is :

P = { $S \rightarrow 0S2 \mid X$ [X generates a string of the form $0^b 1^b$]
$$X \rightarrow 0X1 \mid \varepsilon$$
}

where, The set of variables V = {S, X}
The set of terminals T = {0, 1}
Start symbol = S

Example 4.2.29

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) $L = \{a^n b^{2n} \mid n > 1\}$
$S \rightarrow aSbb \mid aabbbb$

(b) $L = \{a^m b^n \mid n > m\}$
$S \rightarrow aSb \mid B$
$B \rightarrow bB \mid b$

Example 4.2.30

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 a string of length n over {a, b}* a a string of length n over {a, b}* a

Generated by X

Form 2 : b a string of length n over {a, b}* b a string of length n over {a, b}* b

Generated by Y

The set of productions for the given language is :

$$P = \{ S \rightarrow aXa \mid bYb$$
$$X \rightarrow aXa \mid aXb \mid bXa \mid bXb \mid a$$
$$Y \rightarrow aYa \mid aYb \mid bYa \mid bYb \mid b$$
$$\}$$

where, The set of variables $V = \{S, X, Y\}$

The set of terminals $T = \{a, b\}$

The start symbol = S.

Example 4.2.31

Describe the language generated by each of these grammars. Justify your answer with an example.

(1) $S \rightarrow a S a \mid b S b \mid \in$

(2) $S \rightarrow a S a \mid b S b \mid a \mid b$

(3) $S \rightarrow a S b \mid b S a \mid \in$

(4) $S \rightarrow SS \mid bS \mid a$

Solution :

(1) $L = \{\omega \in \{a, b\}^* \mid \omega$ is a palindrome of even length$\}$

Example : abba

(2) $L = \{\omega \in \{a, b\}^* \mid \omega$ is a palindrome of odd length$\}$

Example : abbba, ababa

(3) $L = \{\omega \in \{a, b\}^*$

ω can be written as $\omega_1 \omega_2$ where

1. ω_2 is reverse of ω_1 with 'a' changed to 'b' and 'b' changed to 'a'.
2. $|\omega| \geq 0$

(4) The production $S \rightarrow bS \mid a$ will generate a string of the form b*a.

The production $S \rightarrow SS$ is for sideway growth of the recursion.

The language generated by the grammar is (b*a) (b*a)*

Syllabus Topic : Ambiguity in Grammar and Ambiguous Grammar

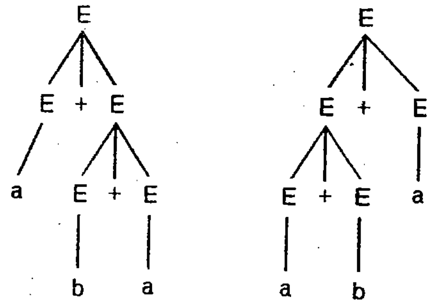
4.3 Ambiguous Grammar

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 :

$$E \rightarrow E + E \mid a \mid b \qquad \text{...(Grammar 4.1)}$$

A string a + b + a is generated by the given grammar.



(a) **(b)**

Fig. 4.3.1 : Parse trees considered for ambiguity

- The grammar 4.1, generates a + b + a in two different ways. The two derivations are shown in Fig. 4.3.1(a) and Fig. 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 gets a 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 (4.1) can be enforced by introducing one more variable T. Variable T can not be broken by + operator.
- An unambiguous grammar for the grammar of (4.1) is given in (4.2)

$$E \rightarrow E + T \mid T$$

$$T \rightarrow a \mid b \quad \text{...(Grammar 4.2)}$$

- 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 (4.2), for the string $a + b + a$.

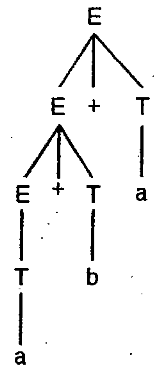


Fig. 4.3.2 : A parse tree for a + b + a using an unambiguous grammar

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.

An example of inherently ambiguous language is

$$L = L_1 \cup L_2$$

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

$E \rightarrow E + E \mid E * E \mid (E) \mid I$

$I \rightarrow a \mid b$

a) Show that the grammar is ambiguous.

b) 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

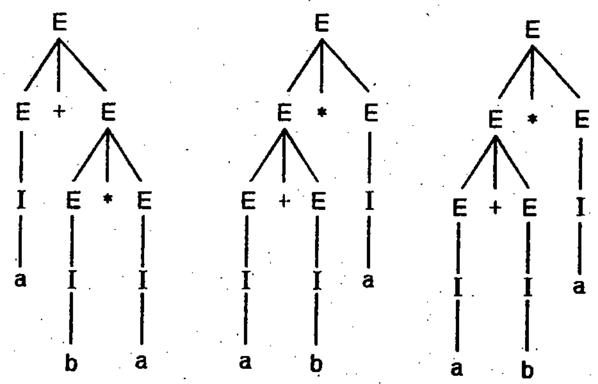
(a) Identifiers (b) Literals

(c) Parenthesized expressions

An unambiguous grammar is given below :

$$\left.\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid a \mid b \end{aligned}\right\} \text{(Grammar 4.3)}$$

A parse tree for $a + b * a$ using both ambiguous and unambiguous grammar is shown in Fig. Ex. 4.3.1.



(a) Using ambiguous grammar **(b) Using ambiguous grammar** **(c) Using unambiguous grammar**

Fig. Ex. 4.3.1 : Parse trees for a + b * a

Fig. Ex. 4.3.1(a) and Fig. 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 Fig. Ex. 4.3.1(b)).

Example 4.3.2 SPPU - May 14. 8 Marks

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.

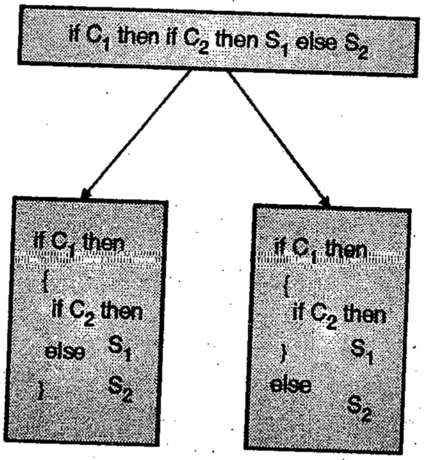


Fig. Ex. 4.3.2 : Two interpretations of a nested if-statement

– 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 :

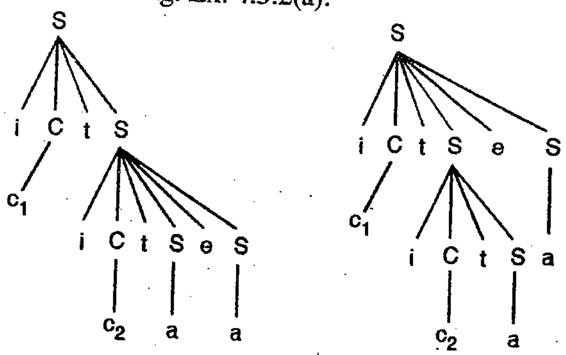
$S \rightarrow$ if C then S
 | if C then S else S
 | a [termination of a statement by 'a']
$C \rightarrow c_1 \mid c_2$

The above grammar can be written in short as :

$S \rightarrow iCtS \mid iCtSeS \mid a$
$C \rightarrow c_1 \mid c_2$ (Grammar 4.4)

Where, i stands for if,
t stands for then,
and e stands for else

Example : Grammar (4.4) can be shown to be ambiguous by drawing two different derivation trees for the string ic_1tic_2t aea. Two different derivations are shown in the Fig. Ex. 4.3.2(a).



(i) (ii)

Fig. Ex. 4.3.3(a) : Two derivations for ic_1tic_2taea

Removing ambiguity : The ambiguity from the grammar (4.4) can be removed by binding dangling else with inner if – statement. We will introduce two more variables.

U → For if-statement without else part.
M → For if-statement with else part.

An unambiguous grammar is given below :

$S \rightarrow U \mid M$
$M \rightarrow iCtMeM \mid a$
$U \rightarrow iCtS \mid iCtMeU$
$C \rightarrow c_1 \mid c_2$ (Grammar 4.5)

The working of grammar (4.5) can be understood with the help of derivation tree for ic_1tic_2 taea. It is shown in Fig. Ex.4.3.3(b).

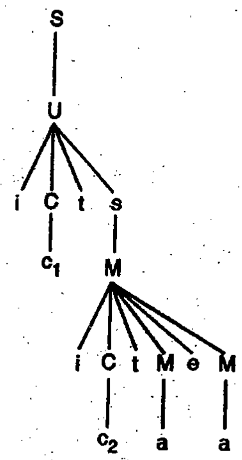


Fig. Ex. 4.3.3(b) : Derivation tree for ic_1tic_2taea

- Outer if-statement is without else part.
- Inner if-statement is with else part.

Example 4.3.4

Show that the CFG given below. Which generates all strings of balanced parentheses is ambiguous. Give an equivalent unambiguous grammar.

$S \rightarrow SS \mid (S) \mid \varepsilon$

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).

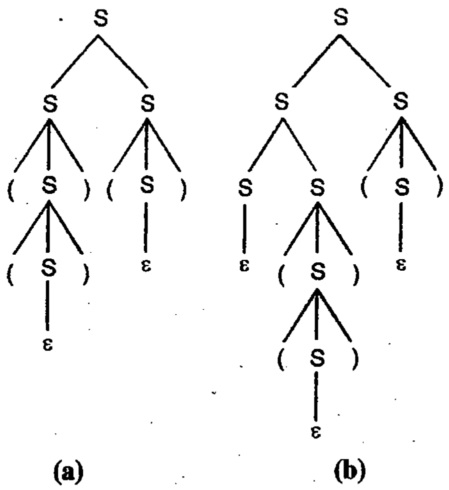


Fig. Ex. 4.3.4: Two different derivations for (( ))( ).

Since, there are two different derivation for the string (( ))( ) the grammar is ambiguous.

An unambiguous grammar is given by

$S \rightarrow ST \mid T$

$T \rightarrow (S) \mid ()$

- 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]

$F \rightarrow F \wedge G \mid G$ [^ 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)

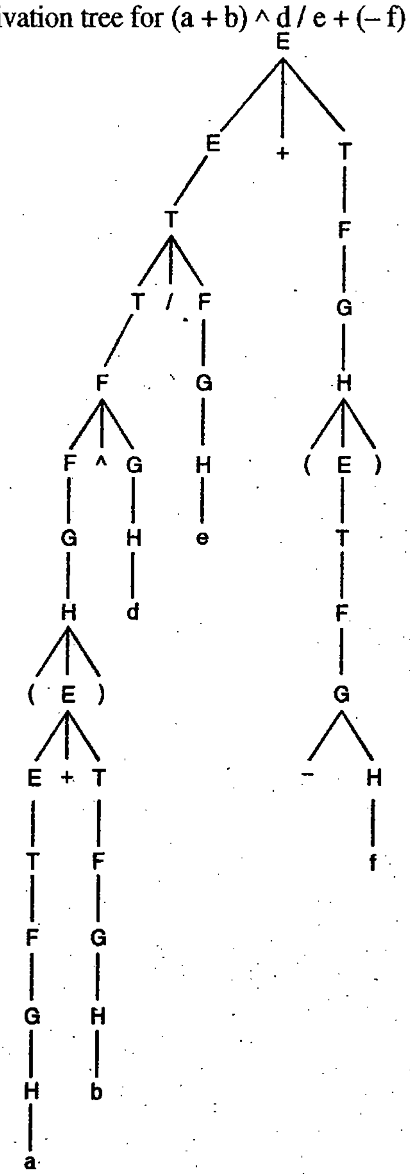


Fig. Ex. 4.3.5 : Derivation tree for (a + b) ^ d / e + (– f)

Example 4.3.6

Is the following CFG ambiguous ?

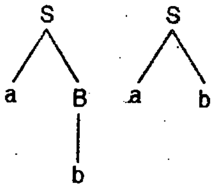
$S \rightarrow aB \mid ab$

$A \rightarrow aAB \mid a$

$B \rightarrow ABb \mid b$

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.



(a) (b)

Fig. Ex. 4.3.6 : Two derivations for ab

Example 4.3.7

Is the following CFG ambiguous ?

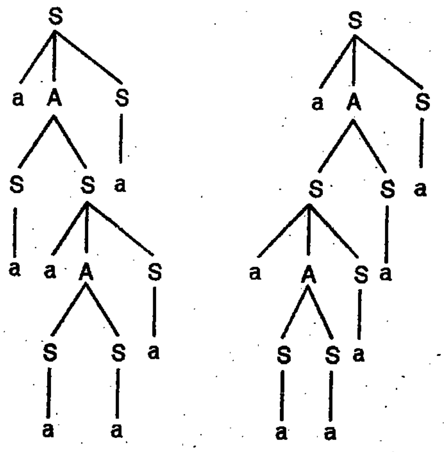
G = ({S, A}, {a, b}, P, S)

where P consists of

S → aAS | a

A → SbA | SS | 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 are shown in Fig. Ex. 4.3.7.



(a) (b)

Fig. Ex. 4.3.7 : Two derivations for a^7

Example 4.3.8

Consider the grammar having productions :

S → aS | ε

S → aSbS

The grammar is ambiguous.

(i) Show in particular that the string aab has two parse trees.

(ii) Find an unambiguous grammar for the same.

Solution :

(i) Two parse trees for aab are given in Fig. Ex. 4.3.8.

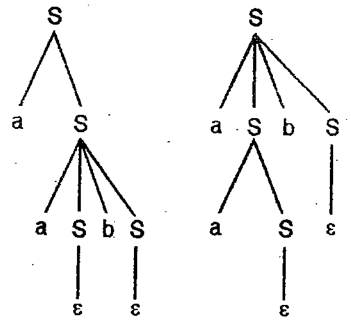


Fig. Ex. 4.3.8 : Two derivations for aab

(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 given below :

S → U | M

U → aS | aMbU

M → aMbM | ε

Example 4.3.9 SPPU - May 12, Dec. 14, 8 Marks

Let G be the grammar

S → aB | bA

A → a | aS | bAA

B → b | bS | aBB

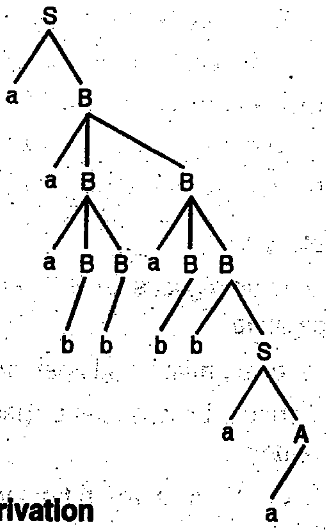
For string aaabbabbba find

(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



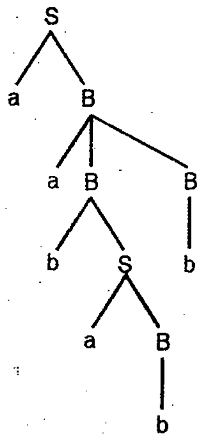
(i) Left most derivation

S → aB → aaBB → aaaBBB → aaabBB → aaabbB → aaabbaBB → aaabbabB → aaabbabbS → aaabbabbbA → aaabbabbba

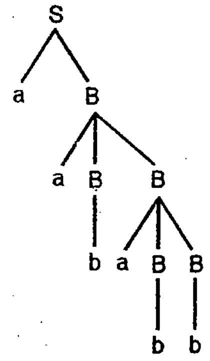
(ii) Right most derivation

S→aB→aaBB→aaBaBB→aaBaBbS→aaBaBbbA
→aaBaBbba→aaBabbba→aaaBBabbba→aaaBbabbba
→aaabbabbba

(iv) The grammar is ambiguous as we can draw two parse trees for aababb



Parse Tree 1



Parse Tree 2

Example 4.3.10

In each case, show that the grammar is ambiguous, and find the equivalent unambiguous grammar

(a) S→ SS |a|b

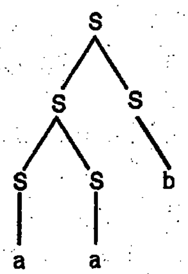
(b) S→ ABA, A → aA|ε, B→ bB|ε

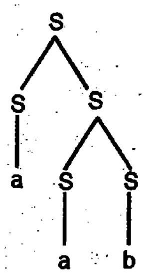
(c) S→ aSb|aaSb|ε

Solution :

(a) S→SS|a|b is ambiguous

Two parse trees can be generated for the string aab.



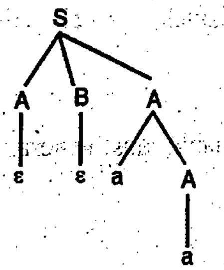


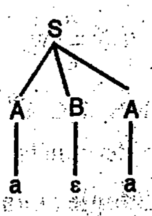
Unambiguous grammar :

S→ aS|bS|aa|ab|ba|bb

(b) S→ABA, A→aA |ε, B→bB|ε

Two parse trees can be generated for the string aa





Unambiguous grammar :

S →aS|bX|ε

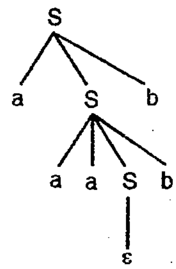
[String of only a's will be generated by S→aS|ε]

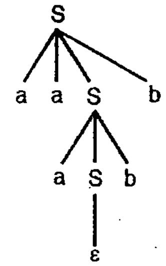
X →bX|y

Y →aY|ε

(c) S→aSb|aaSb|ε is ambiguous

We can draw two production trees for aaabb





Unambiguous grammar

Given language is of the form

L $= \{a^i b^j \mid i \geq j \text{ and } i, j \geq 0\}$

S →aSb|X

X →aX|ε

Example 4.3.11

Consider the grammar

G = {V = {E, F}, T = {a, b, –}, E, P}

where P consists of rules :

E → F – E, F → a, E → E – F, F → b, E → F

(a) Show that G is ambiguous

(b) Remove the ambiguity.

Solution : The production rules can be re-written as given below :

E →F – E | E – F | F

F →a | b

(a) The grammar can be shown to be an ambiguous grammar by drawing two different derivation trees for the string a – b.

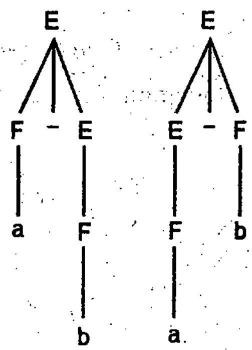


Fig. Ex. 4.3.11 : Two derivations for a – b

(b) The production E → F – E, makes the evaluation from right to left which is normally not allowed.

Thus the ambiguity can be removed by deleting the product the production E → F – E. The resulting an unambiguous grammar is :

E →E – F | F

F →a | b

Example 4.3.12

Consider the CFG with productions

S→ S + S | S – S | S ∗ S | S/S | (S) | a Is the grammar ambiguous ? If 'Yes' find the equivalent unambiguous grammar.

Solution : The given grammar is ambiguous. We can find two different derivations for a + a *a.

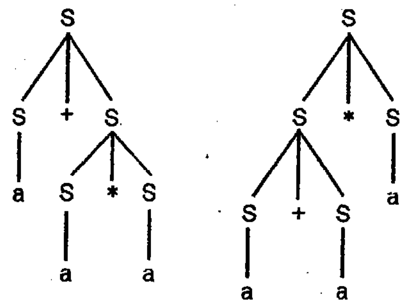


Fig. Ex. 4.3.12

We can write an equivalent ambiguous grammar by incorporating the concept of precedence and associatively. An unambiguous grammar is given below.

S → S + T | S – T

T → T ∗ F | T/F

F → (S) | a

Example 4.3.13 SPPU - Dec. 12, 4 Marks

Show that following grammar is ambiguous :

S → a | absb | aAb

A → bs | aAAb

Solution : There are two distinct derivations for the string abab. Hence, the grammar is ambiguous.

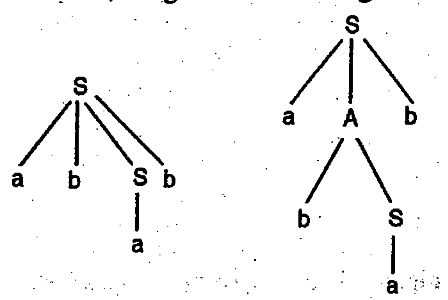


Fig. Ex. 4.3.13 : Two derivations for abab

Example 4.3.14 SPPU - May 13, 6 Marks

Consider the grammar G given by S→S+S|S*S|a|b. find the derivation tree for a*b+a*b. is the grammar ambiguous.

Solution : Derivation tree :

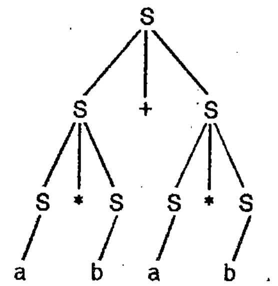


Fig. Ex. 4.3.14

Grammar is ambiguous, it does not handle the precedence of operators + and *

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.

Syllabus Topic : Elimination of Useless Symbols and Productions

4.4.1 Elimination of Useless Symbols

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 ∈ V (Set of variables) is a generating symbol if $X \overset{*}{\underset{G}{\Rightarrow}} w$,

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 (4.6)

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 (4.6) can be simplified by deleting every production containing the useless symbol B. A simplified grammar is given in (4.7).

$S \rightarrow Aa \mid a \mid b$
$A \rightarrow Aa \mid a$ ...Grammar (4.7)

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.

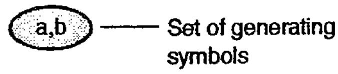
Example 4.4.1

Find non-generating symbols in the grammar given below

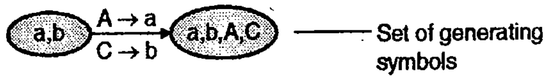
$S \rightarrow AB \mid CA$
$B \rightarrow BC \mid AB$
$A \rightarrow a$
$C \rightarrow aB \mid b.$

Solution :

Step 1 : Every terminal is generating

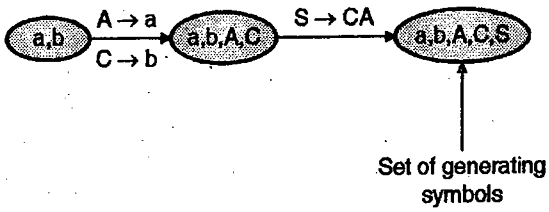


Step 2 : A and C are generating as $A \rightarrow a$ and $C \rightarrow b$.



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 40 can be simplified by deleting productions with the useless symbol B. Simplified grammar is given below :

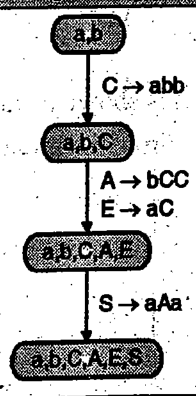
$S \rightarrow CA$
$A \rightarrow a$
$C \rightarrow b$

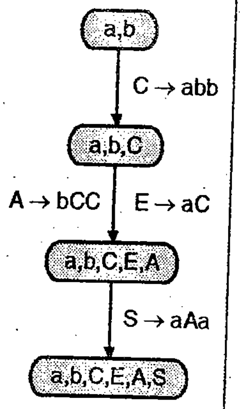
Example 4.4.2

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
2. $S \rightarrow aAa$ $A \rightarrow Sb \mid bCC \mid DaA$ $C \rightarrow abb \mid DD$ $E \rightarrow aC$ $D \rightarrow aDa$		Symbol D is non-generating	$S \rightarrow aAa$ $A \rightarrow Sb \mid bCC$ $C \rightarrow abb$ $E \rightarrow aC$

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 $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ and α contains a variable X then X is reachable.

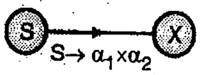
> A grammar containing a non-reachable symbol V_i should be simplified by deleting every production containing the non-reachable symbol V_i.

Finding non-reachable symbols

Non-reachable symbols can be located with the help of a dependently 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 given below



- 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.

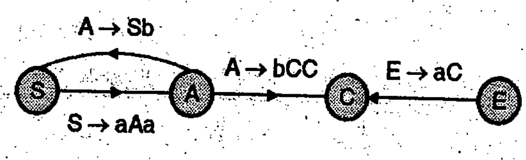
Example 4.4.3

Eliminate non-reachable symbols from the given grammar.

P = {S → aAa, A → Sb | bCC, C → abb, E → aC}

Solution :

Step 1 : Drawing of 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 :

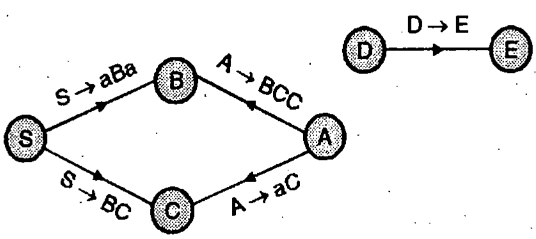
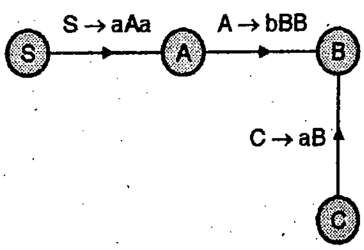
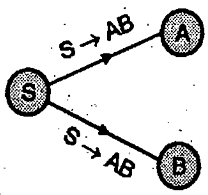
$P_1 \quad = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb\}$

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. $S \rightarrow aBa \mid BC$ $A \rightarrow aC \mid BCC$ $C \rightarrow a, B \rightarrow bCC$ $D \rightarrow E, E \rightarrow d$		A, D and E are non-reachable	$S \rightarrow aBa \mid BC$ $C \rightarrow a$ $B \rightarrow bCC$
2. $S \rightarrow aAa$ $A \rightarrow bBB$ $B \rightarrow ab$ $C \rightarrow aB$		Symbol C is non-reachable	$S \rightarrow aAa$ $A \rightarrow bBB$ $B \rightarrow ab$
3. $S \rightarrow aS \mid AB$ $A \rightarrow bA$; $B \rightarrow AA$		Every symbol is reachable	Same as the original grammar

Syllabus Topic : Elimination of ε-productions

4.4.2 Elimination of ε-productions

A productions of the form $A \rightarrow \varepsilon$, 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

$L(G_1) \; = L(G) - \{\varepsilon\}$

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 $X \rightarrow \varepsilon$.
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 $S \rightarrow ABA, A \rightarrow aA \mid \varepsilon, B \rightarrow bB \mid \varepsilon$

A is nullable as there is a production $A \rightarrow \varepsilon$

B is nullable as there is a production $B \to \varepsilon$

S is nullable as there is a production $S \to ABA$

With both A and B as nullable.

$\therefore$ The set of nullable symbols = {S, A, B}

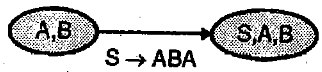
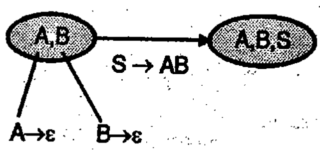
Step 2 : Addition of productions with nullable variables removed.

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 $A \to \alpha_1$, create all possible productions of the form $A \to \alpha_2$, 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.	$S \to aS$ $S \to \varepsilon$	{S}	$S \to aS$ $S \to a$ $S \to \varepsilon$ [$S \to aS$ is written as $S \to a$ and $S \to aS$. S is nullable and hence $S \to aS$ can also be written as $S \to a$]
2.	$S \to ABA$ $A \to \varepsilon$ $B \to \varepsilon$	 A and B are nullable as $A \to \varepsilon$ and $B \to \varepsilon$. S is null as every symbol on the right of $S \to ABA$ is nullable. $\therefore$ Nullable symbols = {A, B, S}	$S \to ABA$ $S \to AB$ $S \to AA$ $S \to BA$ $S \to A$ $S \to B$ $S \to \varepsilon$ $A \to \varepsilon$ $B \to \varepsilon$ Productions derived from $S \to ABA$ by making one or more symbols null.
3.	$S \to aS \mid AB$ $A \to \varepsilon$ $B \to \varepsilon$	 Nullable set = {A, B, S}	$S \to aS$ $\boxed{S \to a}$ Derived from $S \to As$ $S \to AB$ $S \to A$ $S \to B$ $S \to \varepsilon$ Derived from $S \to AB$ $A \to \varepsilon$ $B \to \varepsilon$

Step 3 : Remove ε-productions from the grammar obtained in step 2.

1. $\{S \to aS \mid a\}$
2. $\{S \to ABA \mid AB \mid AA \mid BA \mid A \mid B\}$
3. $\{S \to aS \mid a \mid AB \mid S \to A \mid S \to B\}$

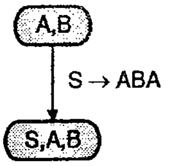
Example 4.4.5

Remove ε-productions from the given grammar.

$S \to ABA$, $A \to aA \mid \varepsilon$, $B \to bB \mid \varepsilon$

Solution :

Step 1 : Find nullable variables



A and B are nullable as $A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$. 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.

$$S \rightarrow ABA \text{ can be written as } \left\{\begin{array}{l} S \rightarrow ABA \\ S \rightarrow AB \\ S \rightarrow BA \\ S \rightarrow AA \\ S \rightarrow A \\ S \rightarrow B \\ S \rightarrow \varepsilon \end{array}\right\}$$

$$A \rightarrow aA \text{ can be written as } \left\{\begin{array}{l} A \rightarrow aA \\ A \rightarrow a \end{array}\right\}$$

$$B \rightarrow bB \text{ can be written as } \left\{\begin{array}{l} B \rightarrow bB \\ B \rightarrow b \end{array}\right\}$$

$\therefore$ Equivalent set of productions.

$$P_1 = \left\{\begin{array}{l} S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \mid \varepsilon \\ A \rightarrow aA \mid a \mid \varepsilon \\ B \rightarrow bB \mid b \mid \varepsilon \end{array}\right\}$$

Step 3 : Remove ε – productions

The final set of productions is given by

$$P_2 = \left\{\begin{array}{l} S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{array}\right\}$$

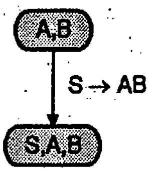
Example 4.4.6

Remove ε-productions from the given grammar.

$S \rightarrow AB \quad A \rightarrow aAA \mid \varepsilon \quad B \rightarrow bBB \mid \varepsilon$

Solution :

Step 1 : Find nullable variables



A and B are nullable as $A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$.

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.

$$S \rightarrow AB \text{ can be written as } \left\{\begin{array}{l} S \rightarrow AB \\ S \rightarrow A \\ S \rightarrow B \\ S \rightarrow \varepsilon \end{array}\right\} \text{ By making}$$

one or more nullable symbols as null.

$$A \rightarrow aAA \text{ can be written as } \left\{\begin{array}{l} A \rightarrow aAA \\ A \rightarrow aA \\ A \rightarrow a \end{array}\right\} \text{ By making}$$

one or more nullable symbols as null.

$$B \rightarrow bBB \text{ can be written as } \left\{\begin{array}{l} B \rightarrow bBB \\ B \rightarrow bB \\ B \rightarrow b \end{array}\right\} \text{ By making}$$

one or more nullable symbols as null.

Step 3 : Remove ε-productions

$\therefore$ Equivalent set of productions.

$$P_1 = \left\{\begin{array}{l} S \rightarrow AB \mid A \mid B \\ A \rightarrow aAA \mid aA \mid a \\ B \rightarrow bBB \mid bB \mid b \end{array}\right\}$$

Example 4.4.7 SPPU - Dec. 12, 6 Marks

Construct reduced grammar equivalent to the grammar

$S \rightarrow a\,A\,a$

$A \rightarrow sb \mid b\,cc \mid D\,a\,A$

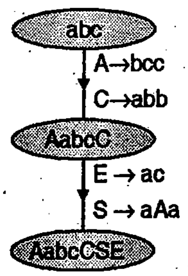
$C \rightarrow abb \mid D\,D$

$D \rightarrow a\,D\,A$

$E \rightarrow a\,C$

Solution :

Step 1 : Finding non-generating symbols



The symbol D is non-generating.

$\therefore$ The reduced grammar after deleting productions containing the symbol D is given by :

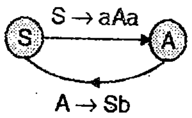
$S \rightarrow aAa$

$A \rightarrow Sb \mid bcc$

$C \rightarrow abb$

$E \rightarrow ac$

Step 2 : Finding non-reachable symbols.



The symbols C and E are non-reachable.

$\therefore$ The reduced grammar is

$S \rightarrow aAa$

$A \rightarrow Sb \mid bcc$

Syllabus Topic : Elimination of Unit Productions

4.4.3 Elimination of Unit Productions

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

  $A_1 \rightarrow A_2, A_2 \rightarrow A_3, A_3 \rightarrow A_4$ 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 $A_i \overset{*}{\underset{G}{\Rightarrow}} A_j$.

Step 3 : From pairs constructed in step 3, we can construct a chain like

$A_1 \rightarrow A_2 \ldots A_j \rightarrow \alpha$, where $A_j \rightarrow \alpha$ is a non unit production.

Each variable A_i to A_j will derive α.

Example 4.4.8

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.

$$P_1 = \begin{Bmatrix} S \rightarrow ABA \mid BA \mid AA \mid AB \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{Bmatrix}$$

Step 2 : Locate every pair of variables (A_i, A_j) such that $A_i \overset{*}{\underset{G}{\Rightarrow}} A_j$

1. (S, A) [Due to unit productions $S \rightarrow A$]
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) Pair (S, B)	$S \rightarrow aA \mid a$ $S \rightarrow bB \mid b$

$$\text{Thus, } P_1 = \begin{Bmatrix} S \rightarrow ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bB \mid b \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{Bmatrix}$$

Example 4.4.9

Eliminate unit productions from the grammar

P = { $E \rightarrow E+T \mid T, T \rightarrow T*F \mid F,$

$F \rightarrow (E) \mid I$

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

}

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

$$P_1 = \begin{Bmatrix} E \rightarrow E+T \\ T \rightarrow T*F \\ F \rightarrow (E) \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{Bmatrix}$$

Step 2 : Locate every pair of variables (A_i , A_j) such that $A_i \overset{*}{\underset{G}{\Rightarrow}} A_j$

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 ot (T, F) and (F, I)]

Step 3 : Unit productions are removed as shown below :

From step 1	$E \rightarrow E+T$ $T \rightarrow T*F$ $F \rightarrow (E)$ $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
Pair (E, T)	$E \rightarrow T*F$
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

$$P_1 = \left\{ \begin{array}{l} E \rightarrow E + T \mid T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ T \rightarrow T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F \rightarrow (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array} \right\}$$

Example 4.4.10

Simplify the following grammar

$S \rightarrow ASB \mid \varepsilon$

$A \rightarrow aAS \mid a$

$B \rightarrow SbS \mid A \mid bb$

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 $S \rightarrow \varepsilon$

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.
$S \rightarrow \varepsilon$ $A \rightarrow aAS$	$S \rightarrow \varepsilon$ $A \rightarrow aAS$ $A \rightarrow aA$] 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 :

$$G_1 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid Sb \mid bS \mid b \mid A \mid bb \end{array} \right\}$$

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 | aA | b

$\therefore$ $\boxed{B \rightarrow A}$ → can be written as → $\boxed{B \rightarrow aAS \mid aA \mid b}$

G_2 , after elimination of unit productions from G_1 is given by :

$$G_2 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid Sb \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \end{array} \right\}$$

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.

$$\therefore\ G_3 = G_2 = \left\{ \begin{array}{l} S \rightarrow ASB \mid AB \\ A \rightarrow aAS \mid aA \mid a \\ B \rightarrow SbS \mid Sb \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \mid a \end{array} \right\}$$

Example 4.4.11 SPPU - Dec. 16. 8 Marks

Simplify the following grammar :

$S \rightarrow 0A0 \mid 1B1 \mid BB$

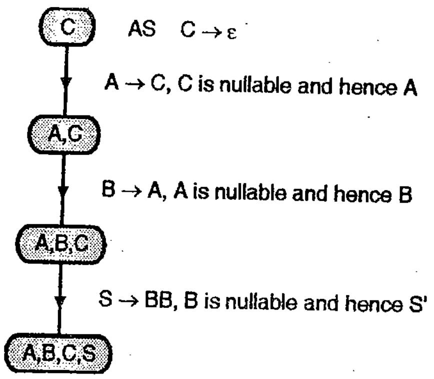
$A \rightarrow C$

$B \rightarrow S \mid A$

$C \rightarrow S \mid \varepsilon$

Solution :

Step 1 : Eliminate ε-production from G (given grammar) and obtain G_1. Set of nullable symbols are given below :



The grammar after removal of null production is given below :

$$\left.\begin{cases} S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid B \mid BB \\ A \rightarrow C,\ B \rightarrow S \mid A,\ C \rightarrow S \end{cases}\right\} \text{– grammar } G_1$$

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.

$B \rightarrow A \rightarrow C \rightarrow S \rightarrow B$

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 :

$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

$A \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

$B \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

Example 4.4.12 SPPU - Dec. 16. 8 Marks

Simplify the following grammar

$S \rightarrow Ab$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

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 $A_i \overset{*}{\underset{G_1}{\Rightarrow}} A_j$

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 $S \rightarrow Ab,\ A \rightarrow a,\ E \rightarrow a$

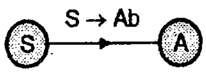
Pair (B, C)	$B \rightarrow a$	[there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$]
Pair (C, D)	$C \rightarrow a$	[there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$]
Pair (D, E)	$D \rightarrow a$	[there is a chain $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$]
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.



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.

$$P_3 = \begin{Bmatrix} S \to Ab \\ A \to a \end{Bmatrix}$$

Example 4.4.13

Find a reduced grammar equivalent to

$S \to aC \mid SB$

$A \to bSCa$

$B \to aSB \mid bBC$

$C \to aBC \mid ad$

Solution : Set of generating symbols can be found as given below :

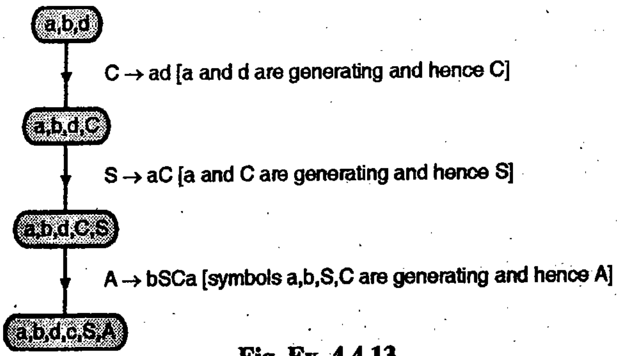


Fig. Ex. 4.4.13

∴ Non generating symbols = {B}

Thus the productions after elimination of non-generating symbols is :

$$P_1 = \begin{Bmatrix} S \to aC \\ A \to bSCa \\ C \to ad \end{Bmatrix}$$

The set of reachable symbols can be found by drawing a dependency graph.

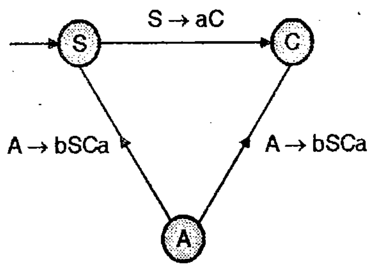


Fig. Ex. 4.4.13(a)

The symbol {A} is non-reachable.

Thus the final set of productions after elimination of non-reachable symbols is :

$$P_2 = \begin{Bmatrix} S \to aC \\ C \to ad \end{Bmatrix}$$

Example 4.4.14 SPPU - Dec. 14, 6 Marks

Simplify given grammar.

$S \to AB$

$A \to aAA \mid \epsilon$

$B \to bBB \mid \epsilon$

Solution : Nullable symbols = {S, A, B}

Equivalent set of productions after removal of ∈-productions.

$$= \begin{Bmatrix} S \to AB \mid A \mid B \\ A \to aAA \mid aA \mid a \\ B \to bBB \mid bB \mid b \end{Bmatrix}$$

Now, the unit productions $S \to A$ and $S \to B$ are removed. The equivalent set of productions becomes :

$$\begin{Bmatrix} S \to AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b \\ A \to aAA \mid aA \mid a \\ B \to bBB \mid bB \mid b \end{Bmatrix}$$

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)

Syllabus Topic : Chomsky Normal Form

4.5.1 Chomsky Normal Form (CNF)

A Context Free Grammar (CFG) without ε-production is said to be in CNF if every production is of the form :

1. $A \to BC$, where $A, B, C \in V$.
2. $A \to a$, 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 consists only of variables.

   i.e. every production of the form

   $A \to \alpha$ with $|\alpha| \geq 2$, α should consists only of variables.

   Example : Consider a production

   $A \to V_1V_2aV_3bV_4$.

   Terminal symbols a and b can be removed by rewriting the production

   $A \to V_1V_2aV_3bV_4$

   as $A \to V_1V_2C_aV_3C_bV_4$

   And adding two productions

   $C_a \to a$ and $C_b \to b$
3. Every production deriving 3 or more variables ($A \to \alpha$ 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 $A \to X_1X_2 \ldots X_n$ where $n \geq 3$ and as X_i's are variables.

   The production $A \to X_1X_2 \ldots X_n$ should be broken down as given below :

$$\begin{aligned} A &\to X_1C_1 \\ C_1 &\to X_2C_2 \\ C_2 &\to X_3C_3 \\ &\vdots \\ C_{n-2} &\to X_{n-1}X_n \end{aligned}$$

Each with two variables on the right.

Example 4.5.1

Find the CNF equivalent to

$S \to aAbB$, $A \to aA$, $B \to bB \mid b$.

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 $A \to \alpha$ where $|\alpha| \geq 2$ should be a variable :

This can be done by adding two productions.

$C_a \to a$ and $C_b \to b$

The set of productions after the above changes is :

$$P_2 = \begin{Bmatrix} S \to C_aAC_bB \\ A \to C_aA \\ B \to C_bB \mid b \\ C_a \to a \\ C_b \to b \end{Bmatrix}$$

Step 3 : Convert to CNF

$$\left.\begin{aligned} S &\to C_aC_1 \\ C_1 &\to AC_2 \\ C_2 &\to C_bB \end{aligned}\right| \text{ Corresponding to } S \to C_aAC_bB$$

$$\begin{aligned} A &\to C_aA \\ B &\to C_bB \mid b \\ C_a &\to a \\ C_b &\to b \end{aligned}$$

$$\therefore P_3 = \begin{Bmatrix} S \to C_aC_1,\ C_1 \to AC_2,\ C_2 \to C_bB \\ A \to C_aA,\ B \to C_bB \mid b,\ C_a \to a \\ C_b \to b \end{Bmatrix}$$

Example 4.5.2 SPPU - May 13, May 14, 6 Marks

Convert the grammar given below to its equivalent CNF :

$S \to PQP$

$P \to 0P \mid \varepsilon$

$Q \to 1Q \mid \varepsilon$

Solution :

Step 1 : Simplify the grammar

Step 1.1 : Eliminate ε productions Nullable symbols = {P, Q, S}

P and Q are nullable as $P \to \varepsilon$ and $Q \to \varepsilon$. S is also nullable as $S \to PQP$ with every symbol on the right as nullable.

Eliminating ε-productions, we get a set of productions P_1

$$P_1 = \begin{Bmatrix} S \to PQP \mid PQ \mid QP \mid P \mid Q \\ P \to 0P \mid 0 \\ Q \to 1Q \mid 1 \end{Bmatrix}$$

Step 1.2 : Eliminate unit productions :

There are two unit productions $S \to P$ and $S \to Q$.

Eliminating unit productions from P_1, we get a set of productions P_2

$$P_2 = \begin{Bmatrix} S \to PQP \mid PQ \mid QP \mid 0P \mid 0 \mid 1Q \mid 1 \mid PP \\ P \to 0P \mid 0 \\ Q \to 1Q \mid 1 \end{Bmatrix}$$

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

$C_0 \rightarrow 0 \qquad C_1 \rightarrow 1$

The set of productions after the changes is given by P_3.

$$P_3 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid PP \mid C_0P \mid 0 \mid C_1Q \mid 1 \\ P \rightarrow C_0P \mid 0 \\ Q \rightarrow C_1Q \mid 1 \\ C_0 \rightarrow 0 \\ C_1 \rightarrow 1 \end{array} \right\}$$

Step 3 : Convert to CNF :

Original production	Equivalent productions in CNF
$S \rightarrow PQP$	$S \rightarrow PC_2$ $C_2 \rightarrow QP$
$S \rightarrow PQ$	$S \rightarrow PQ$
$S \rightarrow QP$	$S \rightarrow QP$
$S \rightarrow PP$	$S \rightarrow PP$
$S \rightarrow C_0P$	$S \rightarrow C_0P$
$S \rightarrow 0$	$S \rightarrow 0$
$S \rightarrow C_1Q$	$S \rightarrow C_1Q$
$S \rightarrow 1$	$S \rightarrow 1$
$P \rightarrow C_0P \mid 0$	$P \rightarrow C_0P \mid 0$
$Q \rightarrow C_1Q \mid 1$	$Q \rightarrow C_1Q \mid 1$
$C_0 \rightarrow 0$	$C_0 \rightarrow 0$
$C_1 \rightarrow 1$	$C_1 \rightarrow 1$

$\therefore$ The set of final productions in CNF is :

$$P_3 = \left\{ \begin{array}{l} S \rightarrow PC_2 \mid PQ \mid QP \mid PP \mid C_0P \mid 0 \mid C_1Q \mid 1 \\ C_2 \rightarrow QP \\ P \rightarrow C_0P \mid 0 \\ Q \rightarrow C_1Q \mid 1 \\ C_0 \rightarrow 0 \\ C_1 \rightarrow 1 \end{array} \right\}$$

Example 4.5.3 SPPU - Dec. 12. Dec. 13. 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.

$C_a \rightarrow a$

and $C_b \rightarrow b$

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

$V = \{\{S\}, \{(,)\}, \{S \rightarrow (S), S \rightarrow SS, S \rightarrow \varepsilon\}, S\}$

where the set of productions

$$P = \left\{ \begin{array}{l} S \rightarrow (S) \\ S \rightarrow SS \\ S \rightarrow \varepsilon \end{array} \right\}$$

An equivalent set of productions without $\in$-productions can be written as

$$P_1 = \begin{Bmatrix} S \to (S) \mid (\ ) \\ S \to SS \end{Bmatrix}$$

Conversion of P_1 in CNF can be done in two steps.

Step 1 : Addition of two productions $C_1 \to ($ and $C_2 \to)$ will change the set of productions to :

$$P_2 = \begin{Bmatrix} S \to C_1SC_2 \mid C_1C_2 \\ S \to SS, C_1 \to (, C_2 \to) \end{Bmatrix}$$

Step 2 : Finding an equivalent CNF :

Production in CFG	Equivalent productions in CNF
$S \to C_1SC_2$	$S \to C_1C_3$ $C_3 \to SC_2$
$S \to C_1C_2$	$S \to C_1C_2$
$S \to SS$	$S \to SS$
$C_1 \to ($	$C_1 \to ($
$C_2 \to)$	$C_2 \to)$

$\therefore$ The set of final productions in CNF is

$$\begin{Bmatrix} S \to C_1C_3 \mid C_1C_2 \mid SS \\ S \to SC_2 \\ C_1 \to (\\ C_2 \to) \end{Bmatrix}$$

Example 4.5.5 SPPU - May 12. 4 Marks

Convert the following grammar to CNF.

S→Aba, S→aab, B→Ac

Solution :

Step 1 : Symbol B is non-reachable. The symbol can be deleted. The grammar after simplification is :

S→Aba|aab

Step 2 : Every symbol is α, in a production of the from A →α with |α| ≥ 2 should be a variable :

This can be done by adding two productions

$C_a \to a$

and $C_b \to b$

The set of productions after above changes is :

$S \to AC_bC_a \mid C_aC_aC_b$

Step 3 : Convert to CNF

$S \to AC_1$

$C_1 \to C_bC_a$

$S \to C_aC_2$

$C_2 \to C_aC_b$

Example 4.5.6

Convert the following grammar to CNF

S → AACD

A → aAb|ε

C → aC|a

A → aDa|bDb|ε

Solution : First of all, the grammar must be simplified.

Step 1 : Removing null productions.

Nullable set = {A}

Null productions are removed with the resulting set of production as :

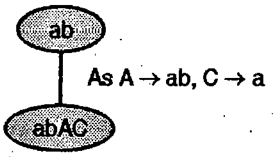
S → AACD|ACD|CD

A → aAb|ab

C → aC|a

A → aDa|bDb

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**.

Example 4.5.7

Simplify and convert the following CFG to Chomsky Normal Form.

S → AACD

A → aAb/∈

C → aC/a

D → aDa/bDb/∈

Solution :

Simplification of Grammar :

Step 1 : Removing null productions

Nullable set = {A, D}

Null productions are removed. The resulting set of productions is given below :

$S \rightarrow AACD \mid ACD \mid CD \mid AAC \mid AC \mid C$

$A \rightarrow aAb \mid ab$

$C \rightarrow aC \mid a$

$D \rightarrow aDa \mid bDb \mid aa \mid bb$

Step 2 : The unit production $S \rightarrow C$ is eliminated. The resulting set of production is gives below.

$S \rightarrow AACD \mid ACD \mid CD \mid AAC \mid AC \mid aC \mid a$

$A \rightarrow a\,Ab \mid ab$

$C \rightarrow aC \mid a$

$D \rightarrow aDa \mid bDb \mid aa \mid bb$

Finding an equivalent CNF

Step 1 : 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

$C_a \rightarrow a$

$C_b \rightarrow b$

The set of production 3 after the above changes is

$$\left\{\begin{array}{l} S \rightarrow AACD \mid ACD \mid CD \mid AAC \mid AC \mid C_aC \mid a \\ A \rightarrow C_a\,AC_b \mid C_a\,C_b \\ C \rightarrow C_a\,C \mid a \\ D \rightarrow C_a\,D\,C_a \mid C_b\,DC_b \mid C_a\,C_a \mid C_b\,C_b \\ C_b \rightarrow a \\ C_b \rightarrow b \end{array}\right\}$$

Step 2 : Grammar in CNF

	Original Production	Equivalent productions in CNF
1.	$S \rightarrow AACD$	$S \rightarrow A\,C_1$ $C_1 \rightarrow A\,C_2$ $C_2 \rightarrow CD$
2.	$S \rightarrow ACD$	$S \rightarrow A\,C_3$ $C_3 \rightarrow CD$
3.	$S \rightarrow CD$	$S \rightarrow CD$
4.	$S \rightarrow AAC$	$S \rightarrow A\,C_4$ $C_4 \rightarrow AC$
5.	$S \rightarrow AC \mid C_a\,C \mid a$	$S \rightarrow AC \mid C_a\,C \mid a$
6.	$A \rightarrow C_a\,AC_b$	$A \rightarrow C_aC_5$ $C_5 \rightarrow A\,C_b$
7.	$A \rightarrow C_aC_b$	$A \rightarrow C_aC_b$
8.	$C \rightarrow C_a\,C \mid a$	$C \rightarrow C_a\,C \mid a$
9.	$D \rightarrow C_a\,DC_a$	$D \rightarrow C_6\,C_b$ $C_6 \rightarrow D\,C_a$
10.	$D \rightarrow C_b\,DC_b$	$D \rightarrow C_b\,C_7$ $C_7 \rightarrow DC_b$
11.	$D \rightarrow C_a\,C_a \mid C_b\,C_b$	$D \rightarrow C_a\,C_a \mid C_b\,C_b$
12.	$C_a \rightarrow a$	$C_a \rightarrow a$
13	$C_b \rightarrow b$	$C_b \rightarrow b$

Example 4.5.8

Justify whether following grammars are in CNF or not.

1.	$S \rightarrow AS \mid a$
	$A \rightarrow SA \mid b$

2.	$S \rightarrow AS \mid AAS$
	$A \rightarrow SA \mid aa$

Solution :

1. In the grammar $S \rightarrow AS \mid a$, $A \rightarrow SA \mid b$ every production is in CNF.
2. In the grammar $S \rightarrow AS \mid AAS$, $A \rightarrow SA \mid aa$ following productions are not in CNF.

   $S \rightarrow AAS$

   $A \rightarrow aa$

Example 4.5.9 SPPU - Dec. 15, 6 Marks

Convert the following CFG into Chomsky Normal Form (CNF):

$S \rightarrow AB$

$A \rightarrow CA \mid \wedge$

$B \rightarrow DB \mid \wedge$

$C \rightarrow 011 \mid 1$

$D \rightarrow 01$

Solution :

Step 1 : Removing null productions

Nullable set $= \{S, A, B\}$

The set of production becomes :

$S \rightarrow AB \mid A \mid B$

$A \rightarrow CA \mid C$

$B \rightarrow DB \mid D$

$C \rightarrow 011 \mid 1$

$D \rightarrow 01$

Step 2 : Removing unit productions we get

$S \rightarrow AB \mid CA \mid 011 \mid 1 \mid DB \mid 01$

$A \rightarrow CA \mid 011 \mid 1$

$B \rightarrow DB \mid 01$

$C \rightarrow 0\,11 \mid 1$

$D \rightarrow 01$

Step 3 : Substituting x for 0 and y for 1

$$
\begin{aligned}
S &\rightarrow AB \mid CA \mid xyy \mid 1 \mid DB \mid xy \\
A &\rightarrow CA \mid xyy \mid 1 \\
B &\rightarrow DB \mid xy \\
C &\rightarrow xyy \mid 1 \\
D &\rightarrow xy \\
x &\rightarrow 0 \\
y &\rightarrow 1
\end{aligned}
$$

Step 4 : Re-writing production in CNF

Sr. No.	Production in step 3	Equivalent production in CNF
1.	$S \rightarrow AB \mid CA \mid 1 \mid DB \mid xy$	$S \rightarrow AB \mid CA \mid 1 \mid DB \mid xy$
2.	$S \rightarrow xyy$	$S \rightarrow XA_1,\ A_1 \rightarrow yy$
3.	$A \rightarrow CA \mid 1$	$A \rightarrow CA \mid 1$
4.	$A \rightarrow xyy$	$A \rightarrow xA1$
5.	$B \rightarrow DB \mid xy$	$B \rightarrow DB \mid xy$
6.	$C \rightarrow xyy \mid 1$	$C \rightarrow xA_1 \mid 1$
7.	$D \rightarrow xy$	$D \rightarrow xy$
8.	$x \rightarrow 0$	$x \rightarrow 0$
9.	$y \rightarrow 1$	$y \rightarrow 1$

Syllabus Topic : Greibach Normal Form

4.5.2 Greibach Normal Form (GNF)

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 $\underbrace{A \rightarrow A}\alpha$ 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

$$A \rightarrow A\alpha \quad \text{... [Left recursive]}$$

$$A \rightarrow \beta \quad \text{... [For termination of recursion]}$$

The language generated by above production is :

$$
\begin{aligned}
A &\rightarrow A\alpha && \text{[From production } A \rightarrow A\alpha] \\
&\rightarrow A\alpha\alpha && \text{[From production } A \rightarrow A\alpha] \\
&\rightarrow A\alpha\alpha\alpha && \text{[From production } A \rightarrow A\alpha] \\
&\quad\vdots \\
&\quad\vdots \\
&\rightarrow A\alpha^n && \text{[From production } A \rightarrow A\alpha] \\
&\rightarrow \beta\alpha^n && \text{[From production } A \rightarrow \beta]
\end{aligned}
$$

Right recursive grammar for $\beta\alpha^n$:

A right recursive grammar for $\beta\alpha^n$ can be written as :

$A \rightarrow \beta B \mid \beta$ [where B generates a string α^n, production $A \rightarrow \beta$ is for termination of recursion]

$B \rightarrow \alpha B \mid \alpha$

Thus a left recursive grammar

$$A \rightarrow A\alpha \mid \beta$$

can be written using a right recursive grammar as :

$$\boxed{\begin{aligned} A &\rightarrow \beta B \mid \beta \\ B &\rightarrow \alpha B \mid \alpha \end{aligned}}$$ right recursive grammar

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. $A \rightarrow Aa \mid b$	$\{b, ba, baa, \ldots ba^n\}$	$A \rightarrow b \mid bB$ $B \rightarrow aB \mid a$
2. $A \rightarrow Aa \mid b \mid c$	$\left\{\begin{matrix} b, ba, baa, \ldots ba^n, \\ c, ca, caa, \ldots ca^n \end{matrix}\right\}$	$A \rightarrow b \mid c \mid bB \mid cB$ $B \rightarrow aB \mid a$
3. $A_1 \rightarrow A_1A_2A_3 \mid A_2A_3$	$\left\{\begin{matrix} A_2A_3, A_2A_3A_2A_3, \ldots \\ \ldots, (A_2A_3)^n \end{matrix}\right\}$	$A_1 \rightarrow A_2A_3 \mid A_2A_3B_1$ $B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$

Grammar with left recursion	Language generated by grammar	Grammar without left recursion
4. $A_1 \rightarrow A_1A_2A_3 \mid A_4A_1 \mid A_5A_3$	$\begin{Bmatrix} A_4A_1, A_4A_1A_2A_3\ldots \\ A_4A_1(A_2A_3)^n, \\ A_5A_3, A_5A_3A_2A_3, \ldots \\ A_5A_3(A_2A_3)^n \end{Bmatrix}$	$A_1 \rightarrow A_4A_1 \mid A_5A_3 \mid A_4A_1B_1 \mid A_5A_3B_1$ $B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$
5. $S \rightarrow S10 \mid 0$	$\{0, 010, 01010, \ldots, 0(10)^n\}$	$S \rightarrow 0B \mid 0$ $B \rightarrow 10B \mid 10$

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 $A \rightarrow X_1X_2\ldots X_i\ldots X_n$, 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

$A \rightarrow V_1V_2aV_3bV_4$ as

$A \rightarrow V_1V_2C_aV_3C_bV_4$ 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 $A_1, A_2, A_3 \ldots A_n$ to create A-productions.

   Example : Consider a grammar given below

   $S \rightarrow aXSY \mid YSX \mid b$

   The variables S, X and Y can be renamed as A_1, A_2 and A_3 respectively. Then the productions become

   $A_1 \rightarrow aA_2A_1A_3 \mid A_3A_1A_2 \mid b$ [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 from $B_i \rightarrow a\alpha$, where a is a terminal and α is a string of non-terminals.

Example 4.5.10 SPPU - Dec. 12, 6 Marks

Construct a grammar in GNF which is equivalent to the grammar

$S \rightarrow AA \mid a$, $A \rightarrow SS \mid b$.

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 :

$$\left.\begin{array}{l} A_1 \rightarrow A_2A_2 \\ A_1 \rightarrow a \\ A_2 \rightarrow A_1A_1 \\ A_2 \rightarrow b \end{array}\right| \text{ Productions after renaming}$$

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 $A_2 \rightarrow A_1\ A_1$ should be modified.

$\Downarrow$

We must substitute $A_2A_2 \mid a$ for the first A_1. We should not touch the second A_1 of A_1A_1.

$$[A_2 \rightarrow A_1A_1] \Rightarrow \begin{bmatrix} A_2 \rightarrow A_2A_2A_1 \\ A_2 \rightarrow a\,A_1 \end{bmatrix}$$

The resulting set of productions is :

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$

Step 3 : Removing left recursion :

The A_2 – productions $A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$ contains left recursion. Left recursion from A_2–production can be removed through introduction of B_2-production.

$A_2 \rightarrow aA_1B_2 \mid bB_2$

$B_2 \rightarrow A_2A_1B_2 \mid A_2A_1$

The resulting set of productions is :

$A_1 \rightarrow A_2A_2 \mid a$

$A_2 \rightarrow aA_1B_2 \mid aB_2 \mid aA_1 \mid b$

$B_2 \rightarrow A_2A_1B_2 \mid A_2A_1$

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.

$A_2 \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b$ ... in GNF

$A_1 \rightarrow A_2 A_2$

$\Downarrow$ Substitute a $A_1B_2 \mid bB_2 \mid aA_1 \mid b$ for first A_2

$A_1 \rightarrow aA_1B_2A_2 \mid bB_2A_2 \mid aA_1A_2 \mid bA_2$

$A_1 \rightarrow a$ ... in GNF

Now, for B_2 – Production

$B_2 \rightarrow A_2A_1B_2$

$\Downarrow$ Substitute a $A_1B_2 \mid bB_2 \mid aA_1 \mid b$ for the first A_2

$B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2$

$B_2 \rightarrow A_2A_1$

$\Downarrow$ Substitute a A_1 B_2 $\mid$ b B_2 $\mid$ a A_1 $\mid$ b for the first A_2

$B_2 \rightarrow a\ A_1B_2A_1 \mid bB_2A_1 \mid aA_1A_1 \mid bA_1$

The final set of productions is :

$$\left.\begin{array}{l} A_2 \rightarrow aA_1B_2 \mid bB_2 \mid aA_1 \mid b \\ A_1 \rightarrow aA_1B_2A_2 \mid bB_2A_2 \mid aA_1A_2 \mid bA_2 \mid a \\ B_2 \rightarrow aA_1B_2A_1B_2 \mid bB_2A_1B_2 \mid aA_1A_1B_2 \mid bA_1B_2 \\ \quad a\ A_1B_2A_1 \mid bB_2A_1 \mid aA_1A_1 \mid bA_1 \end{array}\right\} \text{A set of productions P}$$

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 4.5.11

Find a grammar in GNF for the given CFG.

$E \rightarrow E + T \mid T \qquad T \rightarrow T * F \mid F \qquad F \rightarrow (E) \mid a$

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)

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.

- Left recursion from A_1-production can be removed through introduction of B_1 – production.

  $A_1 \rightarrow A_3A_4A_5B_1 \mid (A_1A_6B_1 \mid aB_1$

  $B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$

- 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 :

$A_1 \rightarrow A_3A_4A_5B_1 \mid (A_1A_6B_1 \mid aB_1 \mid A_3A_4A_5 \mid (A_1A_6 \mid a$

$B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$

$A_3 \rightarrow (A_1A_6B_3 \mid aB_3 \mid (A_1A_6 \mid a$

$B_3 \rightarrow A_4A_5B_3 \mid A_4A_5$

$A_5 \rightarrow (A_1A_6 \mid a$

$A_2 \rightarrow +$

$A_4 \rightarrow *$

$A_6 \rightarrow)$

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.

$A_2 \rightarrow +$

$A_3 \rightarrow (A_1A_6B_3 \mid aB_3 \mid (A_1A_6 \mid a$

$A_4 \rightarrow *$

$A_5 \rightarrow (A_1A_6 \mid a$

$A_6 \rightarrow)$

Already in GNF

$A_1 \rightarrow A_3A_4A_5B_1$

$\Downarrow$ Substituting $(A_1 A_6 B_3 \mid a B_3 \mid (A_1 A_6 \mid a$ for first A_3

$\mathbf{A_1 \rightarrow (A_1A_6B_3A_4A_5B_1 \mid aB_3A_4A_5B_1 \mid (A_1A_6A_4A_5B_1 \mid aA_4A_5B_1}$

$A_1 \rightarrow (A_1A_6B_1 \mid aB_1$ …in GNF

$A_1 \rightarrow A_3A_4A_5$

$\Downarrow$ Substituting $(A_1A_6B_3 \mid aB_3 \mid (A_1A_6 \mid a$ for the first A_3

$\mathbf{A_1 \rightarrow (A_1A_6B_3A_4A_5 \mid aB_3A_4A_5 \mid (A_1A_6A_4A_5 \mid aA_4A_5}$

$A_1 \rightarrow (A_1 A_6 \mid a$ …in GNF

$B_1 \rightarrow A_2A_3B_1 \mid A_2A_3$

$\Downarrow$ Substituting + for A_2

$\boxed{B_1 \rightarrow +A_3B_1 \mid +A_3}$

$B_3 \rightarrow A_4A_5B_3 \mid A_4A_5$

$\Downarrow$ Substituting * for A_4

$\boxed{B_3 \rightarrow *A_5B_3 \mid *A_5}$

The final set of productions is :

$A_2 \rightarrow +$

$A_3 \rightarrow (A_1A_6B_3 \mid aB_3 \mid (A_1A_6 \mid a$

$A_4 \rightarrow *$

$A_5 \rightarrow (A_1A_6 \mid a$

$A_6 \rightarrow)$

$A_1 \rightarrow (A_1A_6B_3A_4A_5B_1 \mid aB_3A_4A_5B_1 \mid (A_1A_6A_4A_5B_1 \mid aA_4A_5B_1 \mid (A_1A_6B_1 \mid aB_1 \mid (A_1A_6B_3A_4A_5 \mid aB_3A_4A_5 \mid (A_1A_6A_4A_5 \mid aA_4A_5 \mid (A_1A_6 \mid a$

$B_1 \rightarrow +A_3B_1 \mid +A_3$

$B_3 \rightarrow *A_5B_3 \mid *A_5$.

Example 4.5.12 SPPU - May 13, Dec. 14, 8 Marks

Give the GNF for following CFG

$S \rightarrow AB$

$A \rightarrow BS \mid b$

$B \rightarrow SA \mid a$

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 :

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow A_1A_2 \mid a$

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 $A_3 \rightarrow A_1A_2$ should be modified

$\Downarrow$ A_1 is changed as per $A_1 \rightarrow A_2A_3$

$A_2A_3A_2$

$\Downarrow$ A_2 is changed as per $A_2 \rightarrow A_3A_1 \mid b$

$A_3A_1A_3A_2 \mid bA_3A_2$

The resulting set of productions is :

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

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

$A_3 \rightarrow bA_3A_2B_3 \mid aB_3$

$B_3 \rightarrow A_1A_3A_2B_3 \mid A_1A_3A_2$

The resulting set of production is

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$

$B_3 \rightarrow A_1A_3A_2B_3 \mid A_1A_3A_2$

Step 4 : A_3 – productions are already in GNF

A_2 – production can be converted to GNF with the help of A_3 – productions.

$A_2 \rightarrow A_3A_1 \mid b$

$\Downarrow$ Substituting $bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$ for A_3.

$\mathbf{A_2 \rightarrow bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1 \mid b}$

A_1 – production can be converted to GNF with the help of A_2 – productions :

$A_1 \rightarrow A_2A_3$

$\Downarrow$ Substituting $bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$ for A_2

$\mathbf{A_1 \rightarrow bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3}$

B_3 – productions can be converted to GNF with the help of A_1 – productions.

$B_3 \rightarrow A_1A_3A_2B_3 \mid A_1A_3A_2$

$\Downarrow$ Substituting $bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3$ for A_1

$\mathbf{B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_3A_2B_3 \mid bA_3A_2A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3 \mid bA_3A_3A_2B_3}$

$\mathbf{B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2 \mid aB_3A_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_3A_2}$

The set of final productions in GNF is :

$A_1 \rightarrow bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$

$A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$

$B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3$

$\mid aB_3A_1A_3A_3A_2B_3$

$\mid bA_3A_2A_1A_3A_3A_2B_3$

$\mid aA_1A_3A_3A_2B_3$

$\mid bA_3A_3A_2B_3$

$\mid bA_3A_2B_3A_1A_3A_3A_2$

$\mid aB_3A_1A_3A_3A_2$

$\mid bA_3A_2A_1A_3A_3A_2$

$\mid aA_1A_3A_3A_2$

$\mid bA_3A_3A_2$

Example 4.5.13

Reduce the following grammar to GNF.

$S \rightarrow AB$, $A \rightarrow BSB \mid BB \mid b$

$B \rightarrow aAb \mid a$

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 :

$S \rightarrow AB$

$A \rightarrow BSB \mid BB \mid b$

$B \rightarrow aAC_b \mid a$, $C_b \rightarrow b$

Step2 : The variables S, A, B and C_b are renamed as A_1, A_2, A_3 and A_4 respectively. The resulting set of productions is to given below.

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1A_3 \mid A_3A_3 \mid b$

$A_3 \rightarrow aA_1A_4 \mid a$

$A_4 \rightarrow b$

Step 3 : Convert to CFG

Given production		Equivalent Production in GNF
$A_4 \rightarrow b$	$\rightarrow$	$A_4 \rightarrow b$
$A_3 \rightarrow aA_1A_4 \mid a$	$\rightarrow$	$A_3 \rightarrow aA_1A_4 \mid a$
$A_2 \rightarrow A_3A_1A_3$ $\Downarrow$	Substituting A_3	$A_2 \rightarrow aA_1A_4A_1A_3 \mid aA_1A_3$
$A_2 \rightarrow A_3A_3$ $\Downarrow$	Substituting A_3	$A_2 \rightarrow aA_1A_4A_3 \mid aA_3$
$A_2 \rightarrow b$	$\rightarrow$	$A_2 \rightarrow b$
$A_1 \rightarrow A_2A_3$ $\Downarrow$	Substituting A_2	$A_1 \rightarrow aA_1A_4A_1A_3A_3 \mid aA_1A_3A_3$ $\mid aA_1A_4A_1A_3A_3 \mid aA_3A_3 \mid bA_3$

$\therefore$ The final set of productions is :

$A_1 \rightarrow aA_1A_4A_1A_3A_3 \mid aA_1A_3A_3 \mid aA_1A_4A_3A_3 \mid aA_3A_3 \mid bA_3$

$A_2 \rightarrow aA_1A_4A_1A_3 \mid aA_1A_3 \mid aA_1A_4A_3 \mid aA_3 \mid b$

$A_3 \rightarrow aA_1A_4 \mid a$

$A_4 \rightarrow b$

Example 4.5.14

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.

$X \rightarrow a$

$Y \rightarrow c$

$S \rightarrow BS \mid AX$

$A \rightarrow bY$

$B \rightarrow AY$

Step 2 : Remaining of variables.

The variables S, A, B, X, Y are renamed as A_1, A_2, A_3 A_4, A_5

$A_4 \rightarrow a$

$A_5 \rightarrow c$

$A_1 \rightarrow A_3 A_1 \mid A_2 A_4$

$A_2 \rightarrow b A_5$

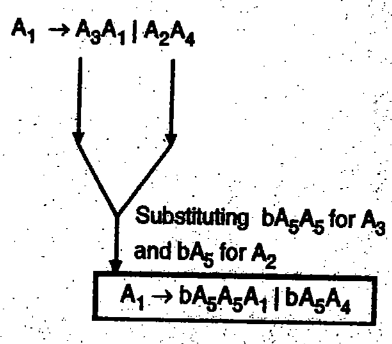
$A_3 \rightarrow A_2 A_5$

Step 3 : Productions for A_4, A_5, A_2 are in the required form. Productions for A_1 and A_3 can be converted to the required form.

$A_3 \rightarrow A_2 A_5$

$\Downarrow$ Substituting bA_5 for A_2

$\mathbf{A_3 \rightarrow bA_5 A_5}$



$\therefore$ **The final set of productions is given by**

$\mathbf{A_1 \rightarrow bA_5A_5A_1 \mid bA_5A_4}$

$\mathbf{A_2 \rightarrow bA_5}$

$\mathbf{A_3 \rightarrow bA_5A_5}$

$\mathbf{A_4 \rightarrow a}$

$\mathbf{A_5 \rightarrow c}$

Example 4.5.15

Find GNF of the grammar given below

$S \rightarrow ABAb \mid ab$

$B \rightarrow ABA \mid a$

$A \rightarrow a \mid b$

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 :

$S \rightarrow AB\,AC_b \mid aC_b$

$B \rightarrow ABA \mid a$

$A \rightarrow a \mid b$

$C_b \rightarrow b$

Step 2 : Substituting $A \rightarrow a \mid b$, in $S \rightarrow ABAC_b$ and $B \rightarrow ABA$, we can convert productions in GNF. The final set of productions is given by :

$S \rightarrow aBAC_b \mid bBAC_b \mid aC_b$

$B \rightarrow aBA \mid bBA \mid a$

$A \rightarrow a \mid b$

$C_b \rightarrow b$

Example 4.5.16

Convert the following grammar to GNF.

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

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Solution :

Step 1 : The two unit production

$S \rightarrow A$

and $S \rightarrow B$

are removed. The resultant set of productions is given by :

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

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

Step 2 : A - productions and B-Productions are already in GNF. The S-productions can be brought to GNF with the help of A and B-productions.

Given production	Productions in GNF
$A \rightarrow aA \mid a$	$A \rightarrow aA \mid a$
$B \rightarrow bB \mid b$	$B \rightarrow bB \mid b$
$S \rightarrow ABA$	$S \rightarrow aABA \mid aBA$
$S \rightarrow AB$	$A \rightarrow aAB \mid aB$
$S \rightarrow BA$	$S \rightarrow bBA \mid bA$
$S \rightarrow AA$	$S \rightarrow aAA \mid aA$
$S \rightarrow aA \mid a \mid bB \mid b$	$S \rightarrow aA \mid a \mid bB \mid b$

Example 4.5.17

Reduce the following grammar to Greibach Normal form.

$S \rightarrow SS$, $S \rightarrow 0S1|01$

Solution :

Step 1 : Substituting X for 1, we get

$$S \rightarrow SS$$
$$S \rightarrow 0SX$$
$$S \rightarrow 0X$$
$$X \rightarrow 1$$

Step 2 : Removing left recursion from $S \rightarrow SS$, we get.

$$S \rightarrow 0SXB \mid 0XB \mid 0SX \mid 0X$$
$$B \rightarrow SB \mid S$$
$$X \rightarrow 1$$

Step 3 : Productions can be brought GNF. Thus the required grammar is :

$$S \rightarrow 0SXB \mid 0XB \mid 0SX \mid 0X$$
$$B \rightarrow 0SXBB \mid 0XBB \mid 0SXB \mid 0XB \mid 0SXB \mid 0XB \mid 0SX \mid 0X$$
$$X \rightarrow 1$$

Syllabus Topic : Chomsky Hierarchy

4.6 Chomsky Classification for Grammar

SPPU - May 13

University Question

Q. Describe in detail Chomsky Hierarchy and context-sensitive languages. **(May 2013, 8 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 :

$$A \rightarrow \varepsilon$$
$$A \rightarrow a$$
$$A \rightarrow aB$$
$$A \rightarrow Ba$$

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 the 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.

$$S \stackrel{*}{\Rightarrow} a^{n-1} S (BC)^{n-1}$$

[by applying $S \rightarrow aSBC$, $(n-1)$ times]

$$\Rightarrow a^{n-1} aBC(BC)^{n-1} \quad \text{[by applying } S \rightarrow aBC]$$

$\Rightarrow$ $a^{n-1}abC(BC)^{n-1}$ [by applying $aB \to ab$]

$\Rightarrow$ $a^{n-1}abB^{n-1}C^n$

[by applying $CB \to BC$, $n-1$ times]

$\Rightarrow$ $a^nb^{n-1}bC^n$

[by applying $bB \to bb$ several times]

$\Rightarrow$ $a^nb^{n-1}bcC^{n-1}$ [by applying $bC \to bc$]

$\Rightarrow$ $a^nb^nc^n$

[by applying $cC \to cc$ several times]

4.6.4 Type 0 or Unrestricted Grammar

SPPU - May 15

University Question

Q. Define unrestricted grammar and give appropriate example. **(May 2015, 2 Marks)**

Productions can be written without any restriction in a unrestricted grammar. If there is production of the $\alpha \to \beta$, 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.

4.6.5 Derivation Graph

Derivation tree or graph are used to represent derivation of a string from CFG. The root of the tree is marked with the start symbol. Every internal node has a label which is a variable. Every leaf node has a label which is a terminal symbol. The yield of a derivation is the concatenation of the labels of the leaves in the left-to-right ordering.

Syllabus Topic : Regular Grammar

4.7 Regular Grammar

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 :

$A \to a$

$A \to aB$

$A \to Ba$

$A \to \varepsilon$

where $a \in T$ and $A, B \in V$.

A language generated by a regular to 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 :

$A \to a$ $A \to aB$ $A \to \varepsilon$	Variable B in $A \to aB$ is the second symbol on the right.

Left-linear form : A left linear regular grammar will have productions of the following form :

$A \to a$ $A \to Ba$ $A \to \varepsilon$	Variable B in $A \to Ba$ is the first symbol on the right.

4.7.1 DFA to Right Linear Regular Grammar

Every DFA can be described using a set of production using the following steps :

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

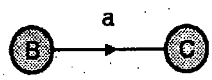
Let the corresponding right linear grammar

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

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,...... where A, B, C, D,....$\in$ V
3. Creating a set of productions P.

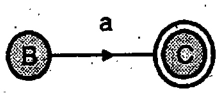
3.1 If $q_0 \in F$ then add a production $S \to \varepsilon$ to P.

3.2 For every transition of the form,



add a production $B \to aC$, where C is a non-accepting state.

3.3 For every transition of the form



add two productions $B \to aC$, $B \to a$, where C is an accepting state.

Example 4.7.1

Give a right linear grammar for the following DFA.

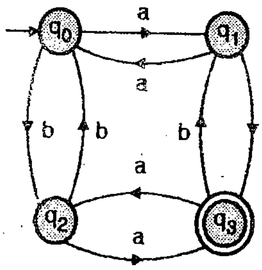
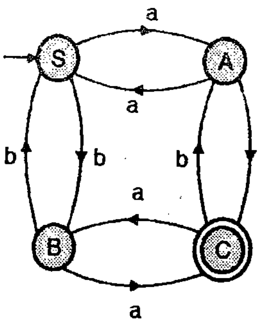


Fig. Ex. 4.7.1

Solution :

Step 1 : Renaming of states, we get the following DFA.



Step 2 : Set of productions are given by :

$$P = \begin{Bmatrix} S \rightarrow aA \mid bB \\ A \rightarrow aS \mid bC \mid b \\ B \rightarrow bs \mid aC \mid a \\ C \rightarrow aB \mid bA \end{Bmatrix}$$

Where, The set of variables $V = \{S, A, B, C\}$

The set of terminals $T = \{a, b\}$

The start symbol $= S$

Example 4.7.2

Give a right linear grammar for the following DFA

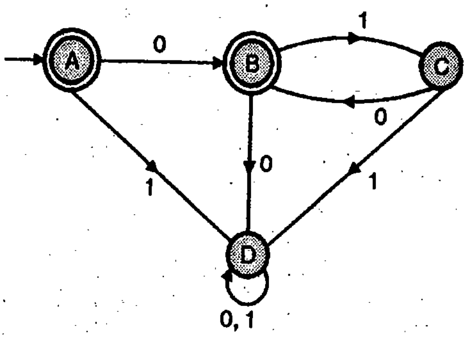


Fig. Ex. 4.7.2

Solution :

Step 1 : State D is a dead state and it can be removed. The FA after deletion of dead state D is given below.

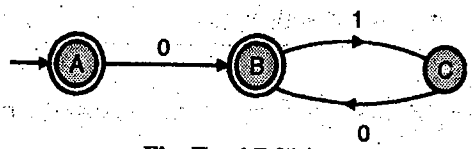


Fig. Ex. 4.7.2(a)

Step 2 : The set of productions are given below.

$$P = \begin{cases} A \rightarrow \varepsilon, & \text{- Start state is a final state} \\ A \rightarrow 0B \mid 0 & \text{- Transition from A to B on 0} \\ B \rightarrow 1C & \text{- Transition from B to C on 1} \\ C \rightarrow 0B \mid 0 & \text{- Transition from C to B on 0} \end{cases}$$

where,

Set of variables $V = \{A, B, C\}$

Set of terminals $T = \{0, 1\}$

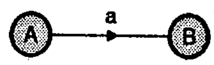
Start symbol $= A$.

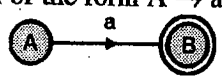
4.7.2 Right Linear Grammar to DFA

Every right linear grammar can be represented using a DFA

- A production of the form

  $A \rightarrow aB$

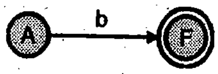
  Will generate a transition , for the DFA

- 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 $A \rightarrow \varepsilon$ 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 4.7.3

Convert the following right-linear grammar to an equivalent DFA.

S → bB

B → bC

B → aB

C → a

B → b

Solution : Re-writing the production we get

$S \rightarrow bB$

$B \rightarrow bC \mid b$

$B \rightarrow aB$

$C \rightarrow a$

Step 1 : Adding transitions corresponding to every production, we get

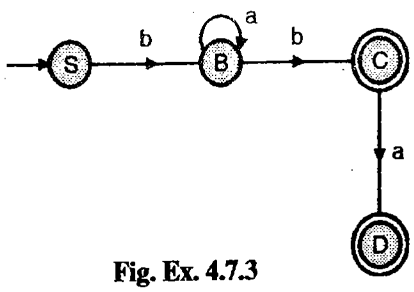


Fig. Ex. 4.7.3

Step 2 : Adding a state E to handle ϕ-transitions, we get the final DFA

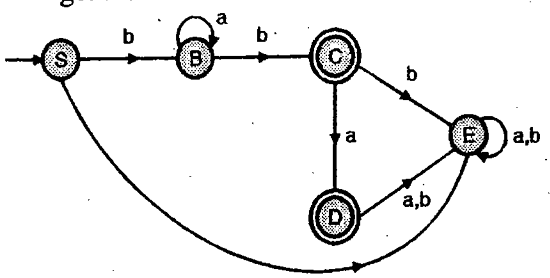


Fig. Ex. 4.7.3(a)

Example 4.7.4 SPPU - May 13, 8 Marks

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 given FA.

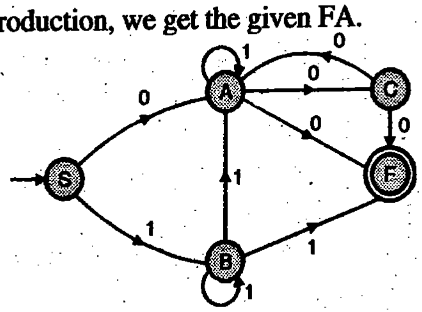


Fig. Ex. 4.7.4

Step 2 : Drawing an equivalent DFA, we get

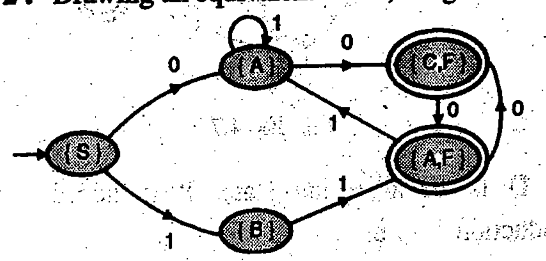


Fig. Ex. 4.7.4(a)

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 given :

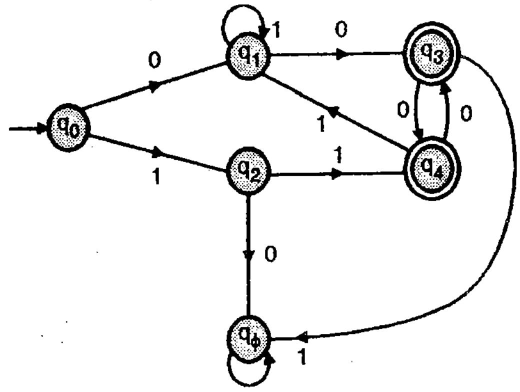


Fig. Ex. 4.7.4(b)

4.7.3 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 4.7.5

Give a left linear grammar for the following DFA

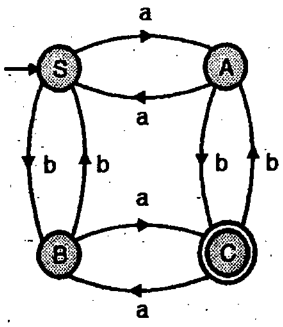


Fig. Ex. 4.7.5

Solution :

Step 1 : Interchanging the starting state and the final state and reversing the direction of transitions, we get a transition graph as given.

Step 2 : Writing an equivalent left linear grammar, we get :

$S \rightarrow Ba \mid Ab, A \rightarrow Sb \mid Ca \mid a$

$B \rightarrow Sa \mid Cb \mid b, C \rightarrow Bb \mid Aa$

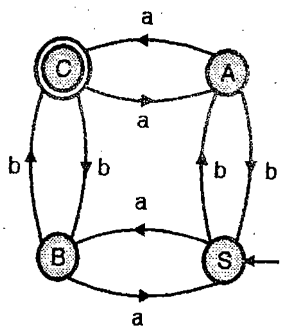


Fig. Ex. 4.7.5(a)

Example 4.7.6

Give a left linear grammar for the following DFA

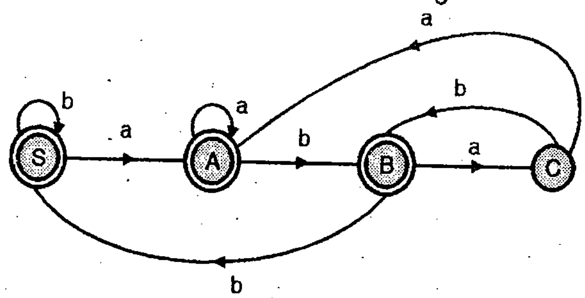


Fig. Ex. 4.7.6

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.

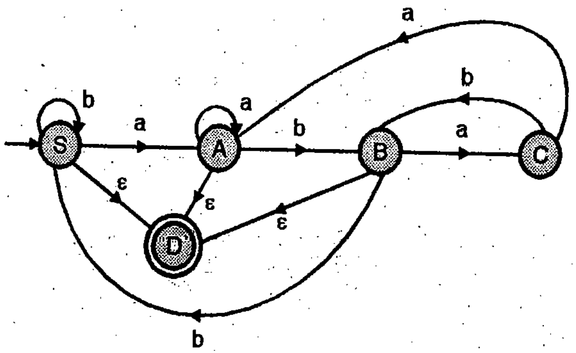


Fig. Ex. 4.7.6(a)

Step 2 : Interchanging the starting state and the final state and reversing direction of transitions, we get.

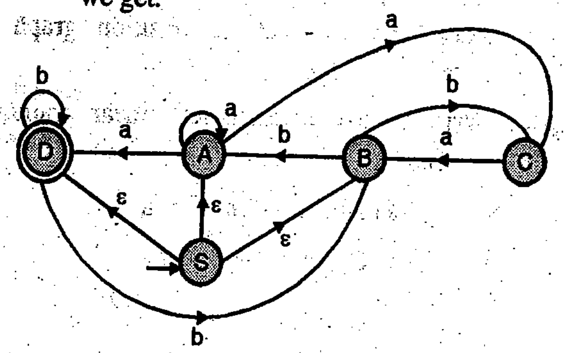


Fig. Ex. 4.7.6(b)

Step 3 : Writing an equivalent left linear grammar we get :

$S \rightarrow D$

$S \rightarrow A$

$S \rightarrow B$

$C \rightarrow Ba$

$B \rightarrow Cb \mid Ab$

$A \rightarrow Aa \mid Da \mid Ca \mid a$

$D \rightarrow Bb \mid Db \mid b$

Step 4 : Removing unit productions, the resulting productions are :

$S \rightarrow Cb \mid Ab \mid Aa \mid Da \mid Ca \mid a \mid Bb \mid Db \mid b$

$C \rightarrow Ba$

$B \rightarrow Cb \mid Ab$

$A \rightarrow Aa \mid Da \mid Ca \mid a$

$D \rightarrow Ba \mid Db \mid b$

4.7.4 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 4.7.7

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.

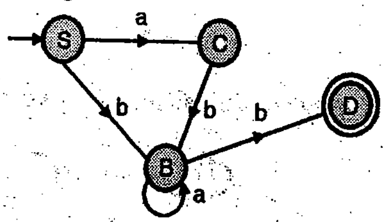


Fig. Ex. 4.7.7

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.

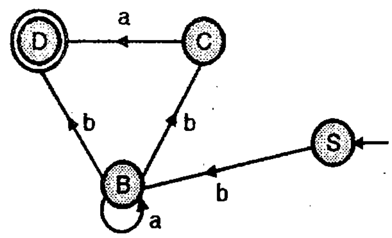


Fig. Ex. 4.7.7(a)

Step 3 : Conversion from FA to DFA

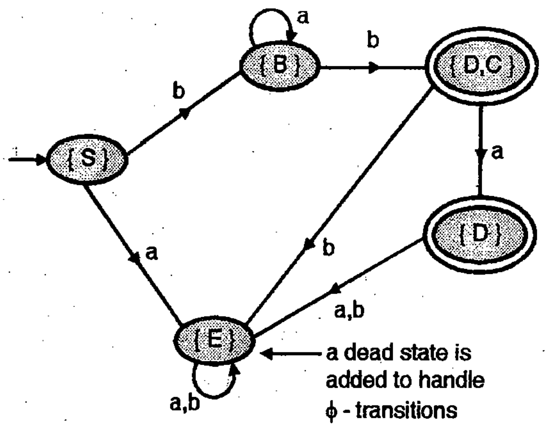


Fig. Ex. 4.7.7(b)

Example 4.7.8 SPPU - May 16, 8 Marks

Construct DFA accepting the language generated by the left linear grammar given below.

S → B1 | A0 | C0

B → B1 | 1

A → A1 | B1 | C0 | 0

C → A0

Solution :

Step 1 : Draw a transition graph from the given left linear grammar State D is a final state.
It is added to handle the following transitions:

$B \rightarrow 1, A \rightarrow 0$

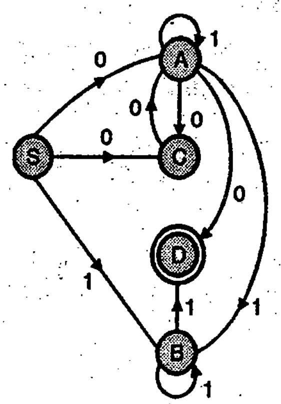


Fig. Ex. 4.7.8

Step 2 : Reverse the direction of transitions and interchange starting state and the final state.

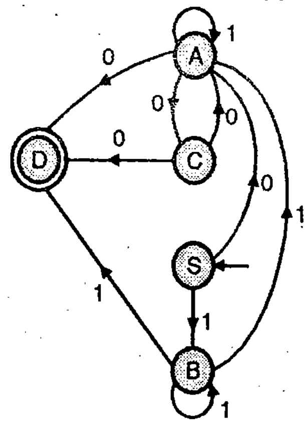


Fig. Ex. 4.7.8(a)

Step 3 : Conversion from FA to DFA.

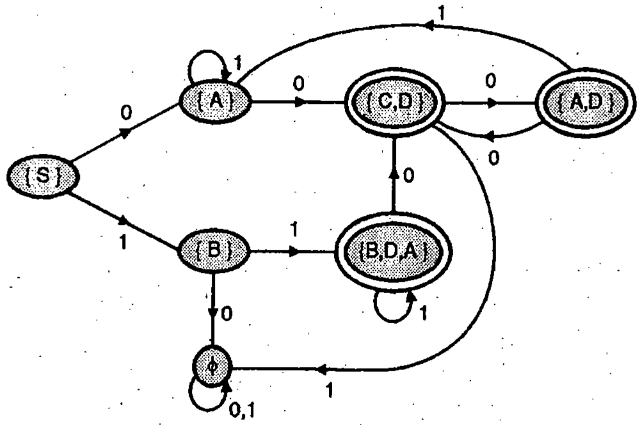


Fig. Ex. 4.7.8(b)

4.7.5 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

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.

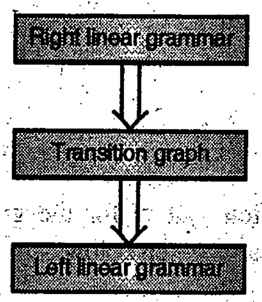


Fig. 4.7.1 : From right linear to left linear grammar

Example 4.7.9

Convert the following right linear grammar to an equivalent left-linear grammar.

S → bB | b

B → bC

B → aB

C → a

B → b

Solution :

Step 1 : Conversion of right linear grammar to transition system.

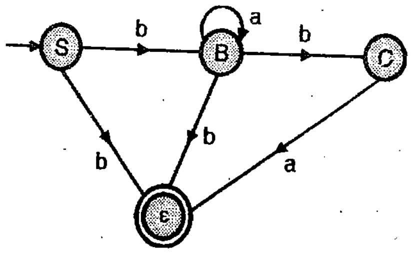


Fig. Ex. 4.7.9

Step 2 : Interchanging the start state with the final state and receiving direction of transitions, we get :

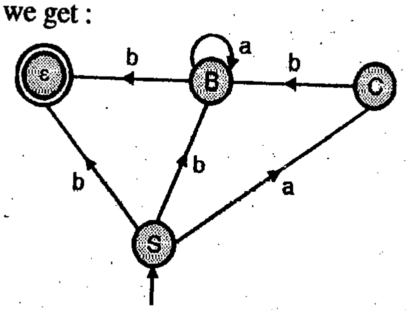


Fig. Ex. 4.7.9(a)

Step 3 : Writing of left linear grammar from the transition system, we get :

S → b | Bb | Ca

B → Ba | b

C → Bb

Example 4.7.10

Write an equivalent left linear grammar from the given right linear grammar.

S → 0A | 1B

A → 0C | 1A | 0

B → 1B | 1A | 1

C → 0 | 0A

Solution :

Step 1 : Transition system for the given right linear grammar is given below :

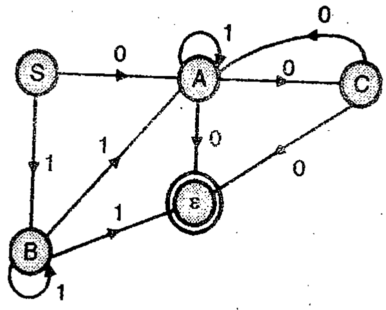


Fig. Ex. 4.7.10

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get

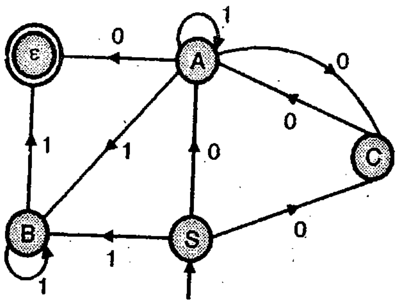


Fig. Ex. 4.7.10(a)

Step 3 : Writing of left linear grammar from the transition system, we get :

S → C0 | A0 | B1

A → A1 | C0 | B1 | 0

B → B1 | 1

C → A0

Example 4.7.11 SPPU - May 12. 6 Marks

For right linear grammar given below, obtain an equivalent left linear grammar.

S → 10A | 01

A → 00A | 1

Solution :

Step 1 : Transition system from the given right linear grammar is given below :

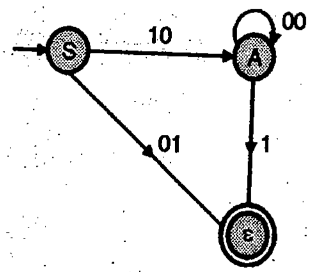


Fig. Ex. 4.7.11

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get :

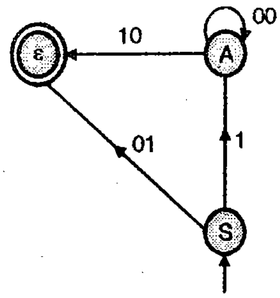


Fig. Ex. 4.7.11(a)

Step 3 : Writing of left linear grammar from the transition system, we get

S → A1 | 01

A → A00 | 10

4.7.6 Left Linear Grammar to Right Linear Grammar

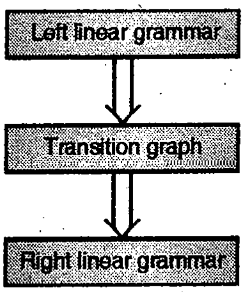


Fig. 4.7.2 : From left linear grammar to right linear grammar

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.

Example 4.7.12

Write an equivalent right linear grammar from the given left-linear grammar.

S → C0 | A0 | B1

A → A1 | C0 | B1 | 0

B → B1 | 1

C → A0

Solution :

Step 1 : Transition system for the left - linear grammar is given below :

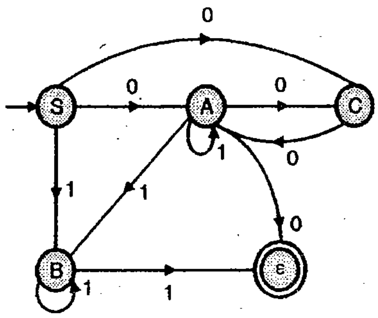


Fig. Ex. 4.7.12

Step 2 : Interchanging the start state and the final state and changing direction of all transitions, we get :

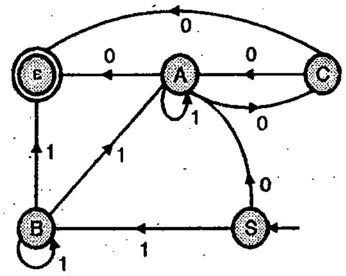


Fig. Ex. 4.7.12(a)

Step 3 : A right linear grammar can be written from the transition system. A set of productions is given below :

S → 1B | 0A

A → 0C | 1A | 0

B → 1B | 1A | 1

C → 0C | 0

Example 4.7.13 SPPU - May 14, 6 Marks

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.

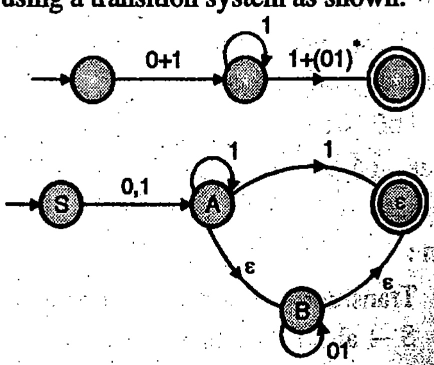


Fig. Ex. 4.7.13

Step 2 : Writing of right linear grammar from the transition system, we get :

$$S \rightarrow 0A \mid 1A,\ A \rightarrow 1A \mid 1 \mid B,\ B \rightarrow 01B \mid \varepsilon$$

The ε - transition. $B \rightarrow \varepsilon$, makes both A and B nullable. The ε–transitions, $B \rightarrow \varepsilon$ is removed and the resulting productions are given below.

$$S \rightarrow 0A \mid 1A \mid 0 \mid 1$$
$$A \rightarrow 1A \mid 1$$
$$B \rightarrow 01B \mid 01$$

Example 4.7.14

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.

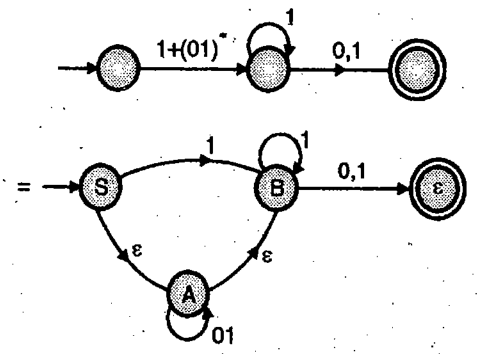


Fig. Ex. 4.7.14

Writing of right linear grammar from the transition system, we get :

$$S \rightarrow 1B \mid A$$
$$A \rightarrow 01A \mid B$$
$$B \rightarrow 1B \mid 0C \mid 1C \mid 0 \mid 1$$

Removing unit productions $S \rightarrow A$ and $A \rightarrow B$:

$$S \rightarrow 1B \mid 01A \mid 1B \mid 0C \mid 1C \mid 0 \mid 1$$
$$A \rightarrow 01A \mid 1B \mid 0C \mid 1C \mid 0 \mid 1$$
$$B \rightarrow 1B \mid 0C \mid 1C \mid 0 \mid 1$$

Example 4.7.15

Draw NFA accepting the language generated by grammar with productions :

$S \rightarrow abA \mid bB \mid aba$

$A \rightarrow b \mid aB \mid bA$

$B \rightarrow aB \mid aA$

Solution :

Step 1 : Transitions system corresponding to $S \rightarrow abA \mid bB \mid aba$ is given by

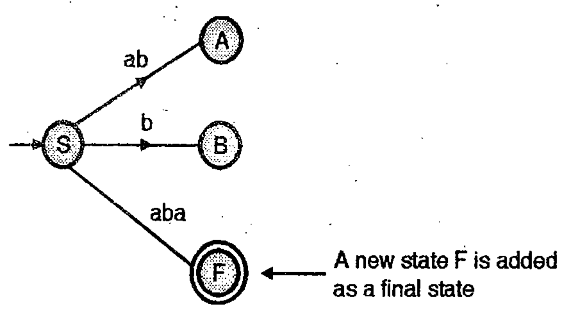


Fig. Ex. 4.7.15

Step 2 : Transitions for $A \rightarrow b \mid aB \mid bA$ and $B \rightarrow aB \mid aA$ are added to the transition graph

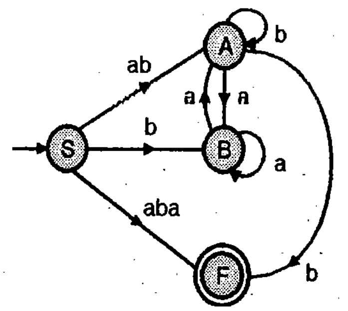


Fig. Ex. 4.7.15(a)

Step 3 : Transitions on ab and aba are expanded with resulting NFA as shown below.

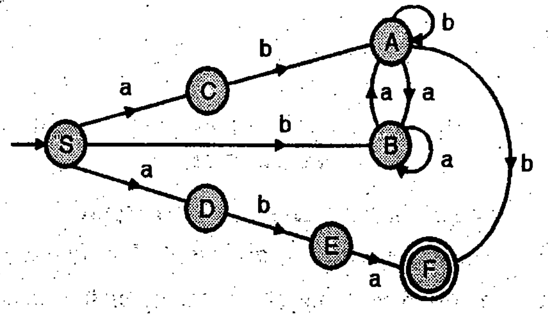


Fig. Ex. 4.7.15(b)

Example 4.7.16

Find the CFG to generate the language defined by the following regular expression.

(i) ab* (ii) a*b*

Solution :

(i) CFG for ab* is given by

$$P = \begin{Bmatrix} S \rightarrow aX \\ X \rightarrow bX \mid \varepsilon \end{Bmatrix}$$

where $G = \{\{S, X\}, \{a, b\}, P, S\}$

(ii) CFG for a* b* is given by

$$P = \begin{Bmatrix} S \rightarrow XY \\ X \rightarrow aX \mid \varepsilon \\ Y \rightarrow bY \mid \varepsilon \end{Bmatrix}$$

where $G = \{\{S, X, Y\}, \{a, b\}, P, S\}$

Example 4.7.17

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.

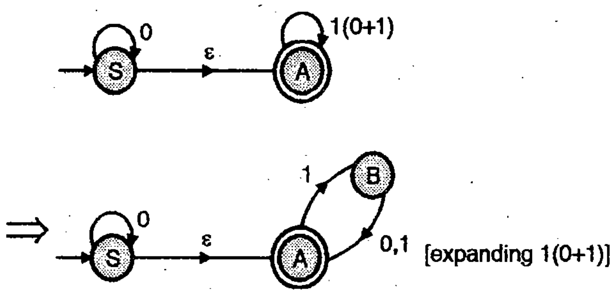


Fig. Ex. 4.7.17

Step 2 : A right linear grammar can be written from the transition graph.

$S \rightarrow 0S \mid A \mid \varepsilon$

$A \rightarrow 1B$

$B \rightarrow 0A \mid 1A \mid 0 \mid 1$

Removing the unit production, $S \rightarrow A$, the resulting grammar is given below :

$S \rightarrow 0S \mid 1B \mid \varepsilon$

$A \rightarrow 1B$

$B \rightarrow 0A \mid 1A \mid 0 \mid 1$

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 given below.

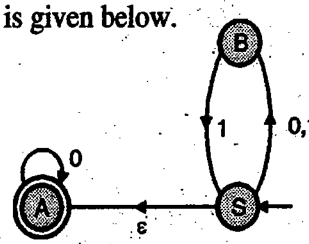


Fig. Ex. 4.7.17(a)

Step 4 : A left linear grammar can be written from the transition diagram

$S \rightarrow B0 \mid B1 \mid A \mid \varepsilon$

$B \rightarrow S1$

$A \rightarrow A0 \mid 0$

Removing the unit production, $S \rightarrow A$, the resulting grammar is given below.

$S \rightarrow B0 \mid B1 \mid A0 \mid 0 \mid \varepsilon$

$B \rightarrow S1$

$A \rightarrow A0 \mid 0$

Example 4.7.18

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.

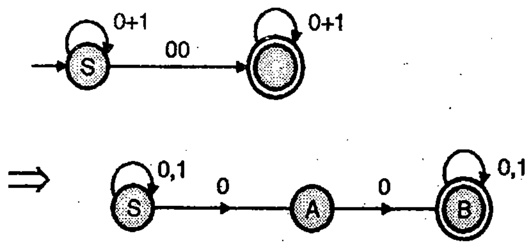


Fig. Ex. 4.7.18

Step 2 : Right linear grammar, from the transition graph can be written as given below :

$S \rightarrow 0S \mid 1S \mid 0A$

$A \rightarrow 0B \mid 0$ — Right linear grammar

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

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.

Resulting transition diagram with above modifications is given below :

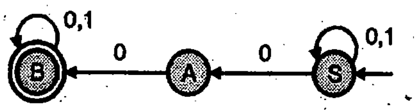


Fig. Ex. 4.7.18(a)

Step 4 : A left linear grammar can be written from the transition graph. It is given below :

$S \rightarrow S0 \mid S1 \mid A0$

$A \rightarrow B0 \mid 0$

$B \rightarrow B0 \mid B1 \mid 1 \mid 0$

Example 4.7.19 SPPU - May 12, 6 Marks

Describe the language generated by the following grammar.

$S \rightarrow bS \mid aA \mid \varepsilon$

$A \rightarrow aA \mid bB \mid b$

$B \rightarrow bS$

Solution : A language can be understood easily from a regular expression. Thus the above problem should be solved as given below :

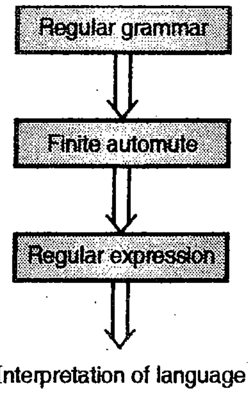


Fig. Ex. 4.7.19

Step 1 : We will first draw a transition diagram from the given regular grammar.

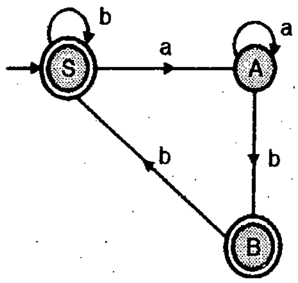


Fig. Ex. 4.7.19(a)

S is a final state, as S → ε

B is a final state, as A → bB | b

Step 2 : A regular expression can be written easily by moving the effect of loop S → A → B → S on S. The resulting transition diagram is given below.

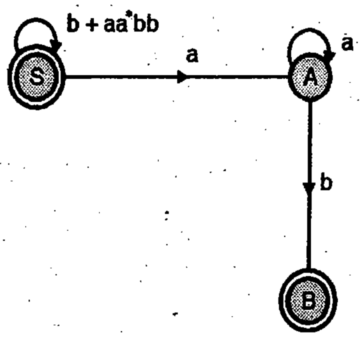


Fig. Ex. 4.7.19(b)

∴ R.E. $= (a + aa^*bb)^*[\varepsilon + aa^*b]$

Language : A string of length 0 or more ending in either ab or b's but never containing aba as a substring.

Example 4.7.20

Describe the language generated by the following grammar.

S → aA | bC | b

A → aS | bB

B → aC | bA | a

C → aB | bS

Solution : A language can be understood easily from a regular expression or a finite automata.

Step 1 : We will first draw a transition diagram from the given regular grammar.

Language generated : From the transition diagram, it clear that any path from S to C will involve :

1. Even number of a's
2. Odd number of b's

Thus the language $L = \{\omega \in \{a, b\}^* \mid \omega\}$ contains even number of a's and odd number of b's.

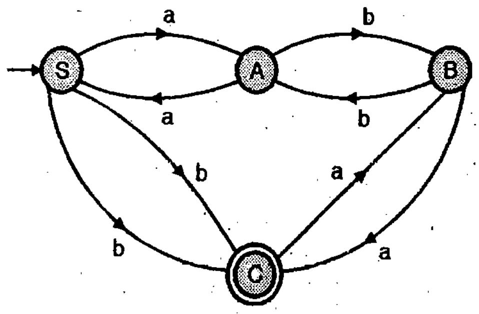


Fig. Ex. 4.7.20

Example 4.7.21

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 → aSa|bSb|aAb|bAa
 A → aAa|bAb|a|b|ε

(2) S → bT|aT|ε
 T → aS|bS

Solution :

(1) It generates a string in which there is an 'a' and if it rewritten as 'b' then the string will become a palindrome.

Example :

The string abaabbabba is in L(G). It the third character 'a' is node b then the modified string abbabbabba is a palindrome.

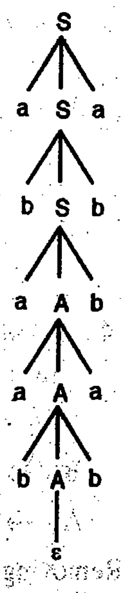


Fig. Ex. 4.7.21

(2) $S \rightarrow bT|aT|\varepsilon$

$A \rightarrow aS|bS$

The FA representing the grammar is given below.

Thus the grammar generates the following language :

$L(G) = \{\omega \in \{a,b\}^* \mid |\omega| \text{ is even}\}$

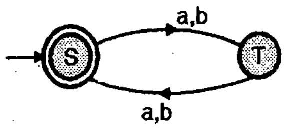


Fig. Ex. 4.7.21(a)

Example 4.7.22 SPPU - Dec. 14, 6 Marks

Prove that language $L = \{a^i b^j \mid i \neq j\}$ is not regular.

Solution :

A regular language is closed under complementation. If the language $L = a^i b^j \mid i = j$ is regular then $L^C = a^i b^j \mid i \neq j$ will also be regular. It can be proved that the language $a^i b^j \mid i = j$ is not regular. The same has been done in the Example 3.6.1.

Hence $L = a^i b^j \mid i \neq j$ is not regular.

Example 4.7.23 SPPU - Dec. 15, 8 Marks

Give the Right and Left linear grammar for the following DFA shown in Fig. Ex. 4.7.23.

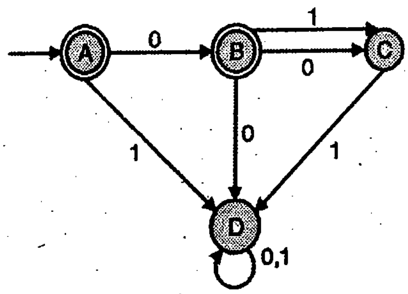


Fig. Ex. 4.7.23

Solution :

Right linear grammar

Set of productions :

$$P_3 = \begin{cases} A \rightarrow 0B \mid 1D \mid \varepsilon \mid 0 \\ B \rightarrow 1C \mid 0C \mid 0D \mid \varepsilon \\ Q \rightarrow 1D \\ D \rightarrow 0D \mid 1D \end{cases}$$

Here, p is the start state.

Left linear grammar

1. Interchange starting and final states.
2. Reverse the arrow direction.
3. Write grammar in left linear form.

Given DFA contains two final states, we can reduce it to one by adding extra state ab follow.

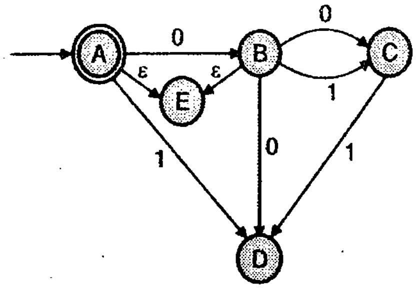


Fig. Ex. 4.7.23(a)

By interchanging start and final state and reversing the arrow directions.

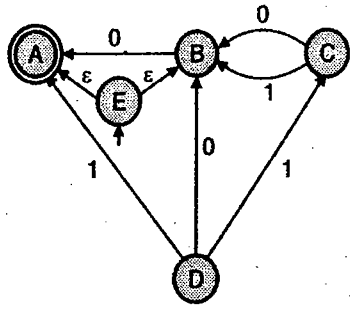


Fig. Ex. 4.7.23(b)

Equivalent left linear grammar.

$F \rightarrow A$

$E \rightarrow B$

$B \rightarrow A0 \mid 0$

$C \rightarrow B0 \mid B1$

$D \rightarrow A1 \mid C1 \mid B0 \mid 1$

4.8 Pumping Lemma for CFG

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. $vy \neq \varepsilon$, strings v and y will be pumped.
3. For all $i \geq 0$, uv^ixy^iz 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).

ϕ (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 :

$$n \geq \phi(G)^{|V-T|}$$

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. 4.8.1 shows paths in detail.

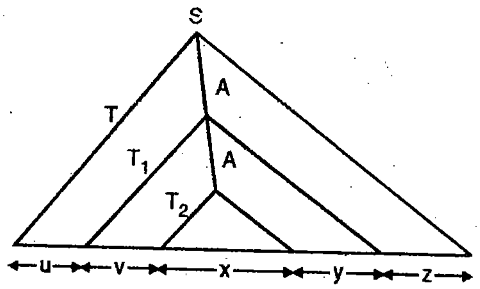


Fig. 4.8.1 : Paths in the parse tree

- 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^ixy^iz where $i \geq 0$.

Example 4.8.1 SPPU - May 13, 6 Marks

Prove that the language

$L = \{a^p \mid p$ 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 $uv^0xy^0z = uxz \in L$. Therefore,

   $|uxz|$ is a prime number.

   Let us assume that $|uxz| = q$.

   Now, let us consider a string uv^qxy^qz,

   The length of uv^qxy^qz is given by :

   $|uv^q xy^q z| = q + q(|v| + |y|)$, which is not a prime with q is a factor.

   Thus $uv^qxy^qz \notin L$. This is a contradiction.

   Therefore, L is not a context free language.

Example 4.8.2

Prove that $L = \{a^ib^ic^i \mid i \geq 1\}$ is not a CFL.

Solution :

1. Let us assume that L is CFL.
2. Let us pick up a word $w = a^nb^nc^n$ where the constant n is given as per the pumping lemma.
3. w is rewritten as uvxyz.

   Where $|vxy| \leq n$ and $v \cdot y \neq \varepsilon$ i.e., both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then uv^ixy^iz is in L (G) for each i = 0, 1, 2, ...

   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^2xy^2z and hence $uv^2xy^2z \notin L(G)$

Case II : If vy does not contain three symbols a, b and c then uv^2xy^2z will have unequal number of a's, b's and c's and hence $uv^2xy^2z \notin L(G)$.

Hence, proved by contradiction.

Example 4.8.3

Prove that $L = \{a^ib^ic^j \mid j \geq i\}$ is not a CFL.

Solution :

1. Let us assume that L is CFL.
2. Let us pick up a word $\omega = a^nb^nc^n$, where n is a constant given as per the pumping lemma.
3. ω is rewritten as uvxyz where, $|vxy| \leq n$ and $v.y \neq \varepsilon$, both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then uv^ixy^iz is in L(G) for each i = 0,1,2....

   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^2xy^2z and hence $uv^2xy^2z \notin L(G)$.

Case II : If vy does not contain three symbols a,b and c then uv^2xy^2z will have either :

(a) Unequal number of a and b

(b) Count of either a or b can be increased from the count of c.

Hence, proved by contradiction.

Example 4.8.4

Prove that the following language is not a context free language.

$L = \{a^n b^m c^p \mid 0 \le n<m<p\}$

Solution :

1. Let us assume that $L = \{a^n b^m c^p \mid 0 \le n<m<p\}$ 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 $|vxy| \le n$ and $v.y \ne \epsilon$, both v and y are not null.
4. From pumping lemma, if $uvxyz \in L$ then uv^ixy^iz is in L(G) for each i = 0,1,2...

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^2xy^2z and hence $uv^2xy^2z \notin L(G)$.

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^2xy^2z. Similarly, the count of the missing symbols/symbol will be more than or equal to the other symbol/symbols in uv^0xy^0z.

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^ixy^iz.

Hence, proved by contradiction.

Example 4.8.5

Prove that the language

$L = \{\omega\omega \mid \omega$ is in $(0 + 1)^*\}$ is not a CFL.

Solution :

1. Let us assume that $L = \{\omega\omega \mid \omega$ 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 $|vxy| \le n$ and $v.y \ne \epsilon$.
4. From pumping lemma, if $uvxyz \in L$, then uv^ixy^iz is in L for each i =0,1,2...
5. vxy must be in one of the following forms : $0^j, 1^j, 0^j1^k, 1^j0^k$

Case I : vxy is of the form 0^j - Two sets of 0's will be unequal in uv^0xy^0z.

Case II : vxy is of the form 1^j - two sets of 1's will be unequal in uv^0xy^0z.

Case III : vxy is of the form 0^j1^k or 1^j0^k-Two set of 0's or two sets of 1's will be unequal in uv^0xy^0z.

Thus the string uv^0xy^0z does not belong to L. Hence, proved by contradiction.

4.9 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 (Decision properties)

Syllabus Topic : Closure Properties of CFL

4.9.1 Closure Properties

SPPU - Dec. 14

University Question

Q. State and explain closure properties of CFLs (Context Free Languages). (Dec. 2014, 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.

4.9.1.1 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.

$\therefore \; L(G) = L_1 \cup L_2$

4.9.1.2 CFL is Closed under Concatenation

Theorem

If L_1 and L_2 are context-free languages, then L_1L_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 G1 and G2 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.

$$S \rightarrow S_1S_2$$

The start symbol S will generate a string w of the form :

$$w = w_1w_2, \text{ where } w_1 \in L_1 \text{ and } w_2 \in L_2.$$

The grammar G can be written as :

$$G = (V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup \{ S \rightarrow S_1S_2 \}, S)$$

4.9.1.3 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^*.

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.

$$S \rightarrow SS_1$$
$$S \rightarrow \varepsilon$$

The production $S \rightarrow SS_1 \mid \varepsilon$ will generate a string w^* where $w \in L_1$.

The grammar G can be written as :

$$G = (V_1, T_1, P_1 \cup \{ S \rightarrow SS_1 \mid \varepsilon \}, S)$$

4.9.1.4 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, $L_1 = \{ a^nb^nc^m \mid n, m \geq 0 \}$

$L_2 = \{ a^mb^nc^n \mid n, m \geq 0 \}$

The language L_1 is a CFL with set of productions given below :

$$S \rightarrow AB$$
$$A \rightarrow aAb \mid \varepsilon$$
$$B \rightarrow cB \mid \varepsilon$$

The language L_2 is a CFL with set of productions given below :

$$S \rightarrow AB$$
$$A \rightarrow aA \mid \varepsilon$$
$$B \rightarrow bBc \mid \varepsilon$$

- 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.

4.9.1.5 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.

4.9.1.6 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 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 :

$\delta((q_1, q_2), u, \beta) = ((p_1, p_2), y)$ [transition for M]

if and only if

$\delta_1(q_1, u, \beta) = (p_1, y)$ [transition for M_1]

and $(q_2, u) \underset{M_2}{\overset{*}{\Rightarrow}} (p_2, \varepsilon)$

- 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 times, it requires $\mid u \mid$ steps to reach the state p_2 from q_2.

Thus M is a PDA for intersection of $L(M_1)$ and $L(M_2)$.

$$L(M) = L(M_1) \cap L(M_2)$$

4.9.1.7 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

$A \rightarrow \alpha^R$ is a production in P^R.

Syllabus Topic : Decision Properties of CFL

4.9.2 Algorithmic Properties (Decision Properties)

There are algorithms for testing :

1. Whether the language generated by a CFG is empty i.e. Is L (G) = ϕ ?
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 : Applications of CFG

4.10 Applications of CFG

SPPU - Dec. 12, Dec. 15

University Questions

Q. Write short note : Application of CFG for parsing. **(Dec. 2012, 6 Marks)**

Q. Explain, with suitable examples, any two applications of context free grammars. **(Dec. 2015, 4 Marks)**

A statement has to follow the syntax of the language. CFG can be used to describe a statement of a programming language or a mark-up language like HTML. A statement like assignment statement,

x = y + z;

may include :

1. Identifiers
2. Operators
3. Constants

An identifier can be described using a regular expression and it can be processed by a DFA. Identifiers, operators, constants and reserve word etc. are also known as tokens. A compiler during compilation scans the source program recognizes coherent group of symbols as a token. The stream of tokens is passed to parser to establish that each statement is syntactically correct.

Syllabus Topic : Parser

4.10.1 Parsers

SPPU - Dec. 13

University Question

Q. Explain the application of context free grammar to syntax analysis phase of compiler with suitable example. **(Dec. 2013, 6 Marks)**

To parse a string x$\in$L(G), one has to draw a parse tree using the productions of G. Derivation of parse tree is similar to checking whether x$\in$L(G). A parser is based on CFG. A partial CFG for a subset of C statements is given below :

<statement> → <Assignment statement>
| <if-statement>
| <while loop>
| <for loop>
| <do while loop>
| <switch statement>

<while-loop> → while (<condition>)
<block of statements>

<block of statements> →{<list of statements>}
|<statement>;

<list of statements> → <statement>; <list of statements>
|<statement>;

<Assignment statement> →<identifier> = <expression>

<expression> →<expression> + <Term>
|<expression> – <Term> |<Term>

<Term> → <Term>* <factor> |<Term>/<factor>
|<factor>

<factor> →<identifier>
|<constant>
|(<expression>)

Syllabus Topic : Markup Languages

4.10.2 Markup Languages

Markup languages like HTML (Hyper Text Markup Language) can be used for describing format of a document. The "strings" in these languages are documents with certain marks (called tags) in them.

- Matching tags <EM> and </EM> indicate that the text between them should be emphasized.

  <EM> Love </EM>

  The word "Love" will be displayed emphasized.
- Matching tags <OL> and </OL> indicate and order list of items.
- Matching tags <p> and </p> indicates a paragraph.
- <LI> indicates a list item.

The things I love : 1. Going to a hotel. 2. People, who are always happy	The text as viewed
<P> The things I <EM> love </EM> <OL> <LI> Going to a hotel. <LI> people, who are always happy </OL>	The text as described in HTML

Syllabus Topic : XML and Document Type Definitions

4.10.3 XML and Document-Type Definitions

XML stands for Extensible Mark-up language. An XML document is made up of structured tags to form a hierarchy of elements. The purpose of XML is not to describe the formatting of documents. Formatting of documents can be handled by HTML. XML can describe the semantics of the text.

The XML grammar derives XML documents that consist of an optional document type definition (DTD) and the document proper, called the document instance. The **XML grammar** describes the syntax of DTDs and instances in general terms. The DTD is specific to an application domain and not only defines the vocabulary of elements, attributes and references in the document but also specifies how these constructs may be combined.

A DTD is based on context free grammar. It has its own notations for describing variables and productions. The form of a DTD is :

```
<!DOCTYPE name_of_DTD[
    list of element definitions
]>
```

An element is defined as :

<!ELEMEMT element_name (description of the element)>

- The special term \#PCDATA stands for any text that does not involve XML tags.
- Following operators are supported :

  1 – stands for union

  , – stands for concatenation

  + – Stands for one or more occurrences.

  * – Stands for zero or more occurrences.

  ? – Stands for zero or one occurrence.
- A DTD for parts can be used to publish items on the web. A sample DTD is given below :

```
<!DOCTYPE Partspees[
        <!ELEMENT PARTS (TITLE? PART*)>
        <!ELEMENT TITLE (\#PCDATA)>
        <!ELEMENT
PART(ITEM,MANUFACTURER,MODEL,COST)+>
        <!ATTLIST PART
            type (computer \auto\airplane)>
        <!ELEMENT ITEM (\#PCDATA)>
        <!ELEMENT MANUFACTURER(\#PCDATA)>
        <!ELEMENT (\#PCDATA)>
        <!ELEMENT  COST (\#PCDATA)>
        ]>
```

- An example of XML document is shown below :

```
<PARTS>
        <TITLE> computer parts </TITLE>
        <PART>
            <ITEM>Motherboard</ITEM>
            <MANUFACTIRER> ASUS
            </MANUFACTURER>
            <MODEL> P4</MODEL>
            <COST>5000</COST>
        </PART>
        <PART>
            <ITEM> sound card </ITEM>
            <MANUFACTURER> creative Labs
            </MANUFACTURER>
            <MODEL> Sound Blaster Live
            </MODEL>
            <COST>3000</COST>
        </PART>
        </PARTS>
```

□□□

CHAPTER 5 | Unit V

Pushdown Automata (PDA)

Syllabus

Basic Definitions, Equivalence of Acceptance by Finite State & Empty stack, PDA & Context Free Language, Equivalence of PDA and CFG, Parsing & PDA: Top-Down Parsing, Top-down Parsing Using Deterministic PDA, Bottom-up Parsing, Deterministic PDA.

5.1 Introduction to Pushdown Automata (PDA)

SPPU - May 12

University Question

Q. Differentiate between-Finite Automata and PDA. **(May 2012, 4 Marks)**

Informally, a pushdown automata can be viewed as a 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.
- 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. 5.1.1, after reading first 5 a's, the stack contains 5 a's.

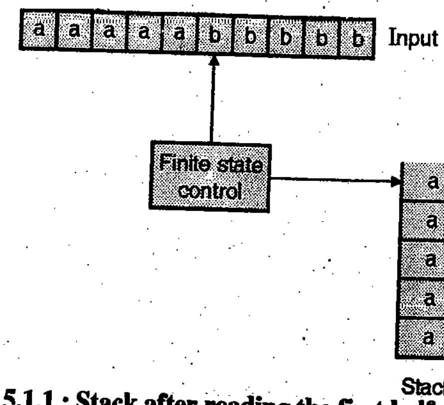


Fig. 5.1.1 : Stack after reading the first half of a^5b^5

- 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. 5.1.2.

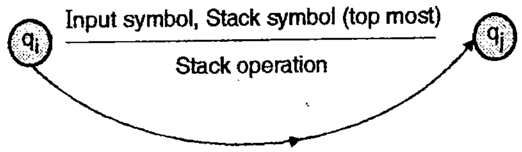


Fig. 5.1.2 : A transition from q_i to q_j

- 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. 5.1.3.

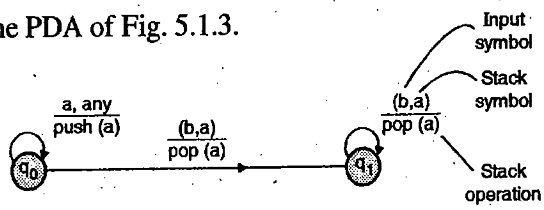


Fig. 5.1.3 : PDA for $a^n b^n$

- 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 : Basic Definitions

5.2 The Formal Definition of PDA

SPPU - Dec. 13. May 15. May 16. Dec. 16

University Questions

Q. Give formal definition of Pushdown automata (PDA). **(Dec. 2013, 2 Marks)**

Q. Define push down automata (PDA). **(May 2015, May 2016, 2 Marks)**

Q. What is PDA ? **(Dec. 2016, 2 Marks)**

Q. Construct a PDA that accept $L = \{a^n b^n \mid n \geq 1\}$ through Empty stack. **(May 2016, Dec. 2016, 7 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 $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ 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

$Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to $Q \times \Gamma^*$

Where

$Q \times (\Sigma \cup \varepsilon) \times \Gamma$ 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.

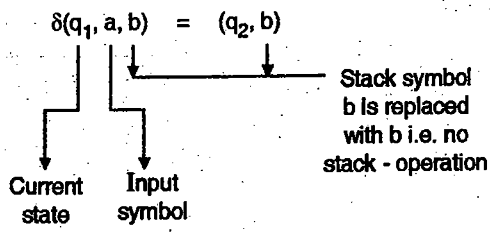
1. Read input with no-operation on stack

- o 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).



- o Right hand side of the transition, (q_2, b) implies that

$$\left[\begin{array}{l}\text{The machine enters a new state } q_2.\\ \text{The top symbol of the stack,}\\ \text{which is b is replaced with b.}\end{array}\right]$$

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.

$\delta(q_1, a, b) = (q_2, \varepsilon)$

Stack symbol (top most) is replaced with ε

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.

$\delta(q_1, a, b) = (q_2, ab)$

Replacing the stack symbol (top most) b with ab will push a symbol a onto the stack

The three operations for stack are shown in Table 5.2.1.

Table 5.2.1 : Transitions rules for stack operations

Stack before transition (assumed)	Transition rule	Stack after transition
1. b ← top, b, a, b, z_0	$\delta(q_1, a, b) = (q_2, b)$	b ← top, b, a, b, z_0 No - operation
2. b ← top, b, a, b, z_0	$\delta(q_1, a, b) = (q_2, \varepsilon)$	b ← top, a, b, z_0 Pop - operation
3. b ← top, b, a, b, z_0	$\delta(q_1, a, b) = (q_2, ab)$	a ← top, b, b, a, b, z_0 Push - operation

Example : Transition function for a PDA accepting a language $L = \{\omega \in \{a, b\}^* \mid \omega$ is of the $a^n b^n$ with $n \geq 1\}$, can be given in various forms :

1. Using a set of rules.
2. 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.]

$\delta(q_0, b, a) = (q_1, \varepsilon)$ [On Seeing the first b, the machine will make a move to q_1 with a POP]

$\delta(q_1, b, a) = (q_1, \varepsilon)$ [For each subsequent b, a POP operation is performed]

$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$ [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		Form 2
$\delta(q_0, a, z_0) = (q_0, a z_0)$	$\Rightarrow$	$((q_0, az_0), (q_0, az_0))$
$\delta(q_0, a, a) = (q_0, aa)$	$\Rightarrow$	$((q_0, a, a), (q_0, aa))$
$\delta(q_0, b, a) = (q_1, \varepsilon)$	$\Rightarrow$	$((q_0, b, a), (q_1, \varepsilon))$
$\delta(q_1, b, a) = (q_1, \varepsilon)$	$\Rightarrow$	$((q_1, b, a), (q_1, \varepsilon))$
$\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$	$\Rightarrow$	$((q_1, \varepsilon, z_0), (q_2, z_0))$

Using Graphical representation (transition diagram)

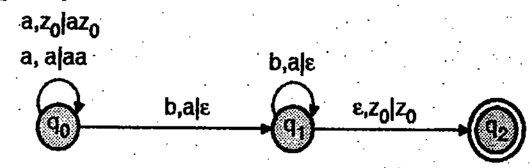


Fig. 5.2.1 : Transition diagram

5.3 Instantaneous Description of a PDA

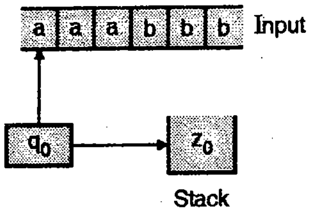
SPPU - Dec. 13

> University Question
>
> **Q.** Give formal definition of instantaneous description of PDA. **(Dec. 2013, 2 Marks)**

Instantaneous behaviour is useful in showing the processing of a string by PDA. Processing of the input string a^3b^3 by the PDA of Fig. 5.2.1 is shown in Fig. 5.3.1.

Graphical representation (instantaneous behaviour)	Tuple format (instantaneous behaviour)

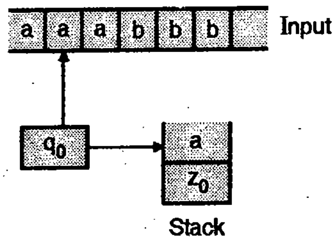
1.



(q_0, aaabbb, z_0)

Current state | Input string | Stack contents

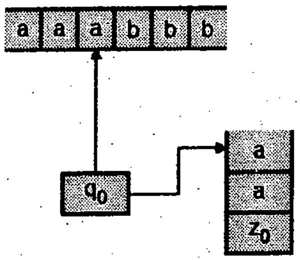
2.



(q_0, aabbb, az_0)

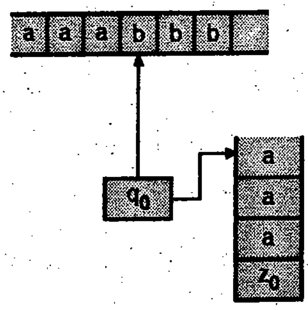
Remaining input | Stack contents

3.



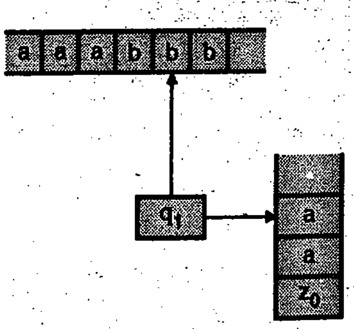
(q_0, abbb, aaz_0)

4.



(q_0, bbb, $aaaz_0$)

5.



(q_1, bb, aaz_0)

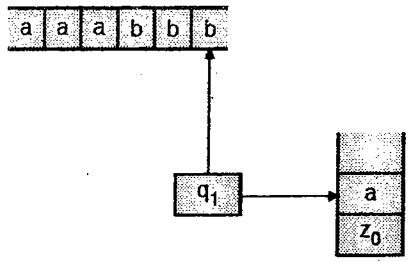
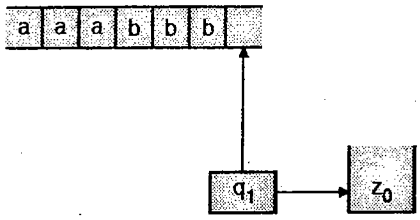
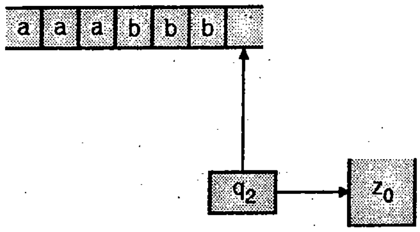
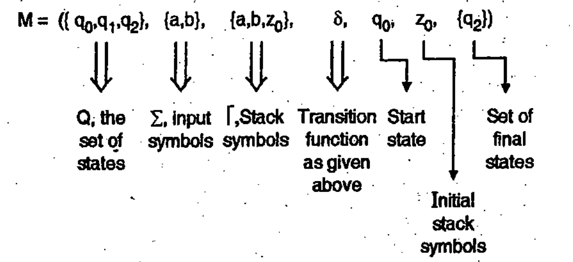
Graphical representation (instantaneous behaviour)	Tuple format (instantaneous behaviour)
6. 	(q_1, b, aZ_0)
7. 	(q_1, ε, z_0)
8. 	(q_2, ε, z_0)

Fig. 5.3.1 : Processing of a^3b^3 by the PDA of Fig. 5.2.1

The PDA for recognizing a string of the form a^nb^n can be described as given below :



Example 5.3.1

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. $\delta(q_0, a, \varepsilon) = (q_0, a)$
2. $\delta(q_0, b, \varepsilon) = (q_0, b)$
3. $\delta(q_0, c, \varepsilon) = (q_1, \varepsilon)$
4. $\delta(q_1, a, a) = (q_1, \varepsilon)$
5. $\delta(q_1, b, b) = (q_1, \varepsilon)$

Show the acceptance of abbcbba by the above PDA through an instantaneous description.

Solution :

$(q_0, abbcbba, z_0) \xrightarrow{\text{(Rule 1)}} (q_0, bbcbba, az_0)$

$\xrightarrow{\text{(Rule 2)}} (q_0, bcbba, baz_0)$

$\xrightarrow{\text{(Rule 2)}} (q_0, cbba, bb\,a\,z_0)$

$\xrightarrow{\text{(Rule 3)}} (q_1, bba, bbaz_0)$

$\xrightarrow{\text{(Rule 5)}} (q_1, ba, baz_0)$

$\xrightarrow{\text{(Rule 5)}} (q_1, a, az_0)$

$\xrightarrow{\text{(Rule 4)}} (q_1, \varepsilon, z_0)$

Since, the machine is in a final state q_1, the string abbcbba is accepted.

5.4 The Language of a PDA

SPPU - Dec. 15

University Question

Q. Explain the equivalence of PDA with acceptance by final state and empty stack. **(Dec. 2015, 6 Marks)**

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.

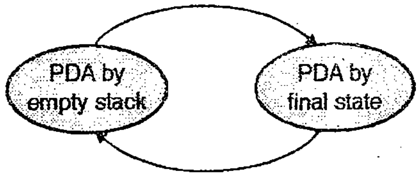


Fig. 5.4.1 : Equivalence of two PDAs

5.4.1 Acceptance by Final State

SPPU - Dec. 13

> **University Question**
> **Q.** Give formal definition of acceptance by PDA in terms of final state. **(Dec. 2013, 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 :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \alpha) \right\}$$

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.

5.4.2 Acceptance by Empty Stack

SPPU - Dec. 13

> **University Question**
> **Q.** Give formal definition of acceptance by PDA in terms of null store. **(Dec. 2013, 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 :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \varepsilon) \right\}$$

Where q_1 is any state belonging to Q and the stack becomes empty on application of input string w.

Syllabus Topic
Equivalence of Acceptance by Finite State and Empty Stack

Theorem 5.4.1

If a language is accepted by a PDA by empty stack then it is also accepted by a PDA by final state.

Proof : The basic idea of proof comes from the Fig. 5.4.2.

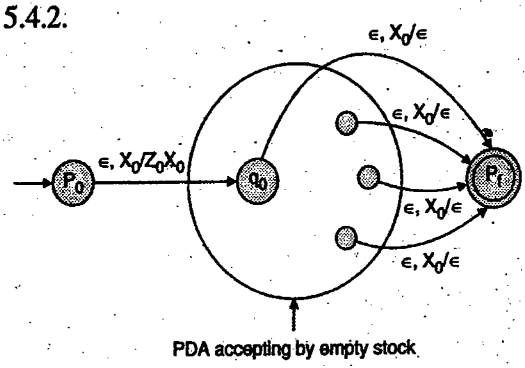


Fig. 5.4.2 : From empty stack to final state

Let P_E represents the PDA accepting L by empty stack. P_F is the equivalent PDA accepting L by final state.

- Construction of P_F from P_E is shown in the Fig. 5.4.2.
- A new symbol X_0 is added to P_F as the start symbol. It lets us know that P_E has reached an empty stack.
- For every state in P_E, a transition is added which takes the PDA from the empty stack state to final state of P_F.
- Thus for the language L of P_E, we can construct a PDA accepting L by final state.

Theorem 5.4.2

For every PDA accepting by final state, there exists a PDA accepting by empty stack.

Proof : The basic idea of proof comes from the Fig. 5.4.3.

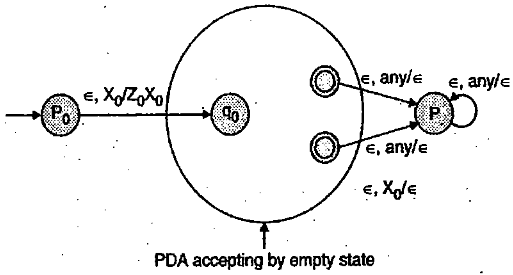


Fig. 5.4.3 : From final state to empty stack

- Let P_F represents the PDA accepting L by final state. P_E is the equivalent PDA accepting L by empty stack.
- Once the P_F enters its final state then the stack is emptied by P_E as shown in Fig. 5.4.3.

Example 5.4.1

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.

$\delta(q_0, 0, z_0) = (q_0, 0z_0)$ [Push the first 0]

$\delta(q_0, 0, 0) = (q_0, 00)$ [Push subsequent 0's]

2. A '0' should be popped for every 1 as input till the stack becomes empty.

$\delta(q_0, 1, 0) = (q_1, \varepsilon)$ [Pop on first 1 and change the state to q_1]

$\delta(q_1, 1, 0) = (q_1, \varepsilon)$ [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.

$$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$$

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. 5.4.1.

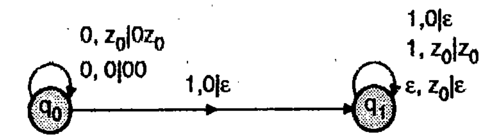


Fig. Ex. 5.4.1(a) : Transition diagram for acceptance through an empty stack

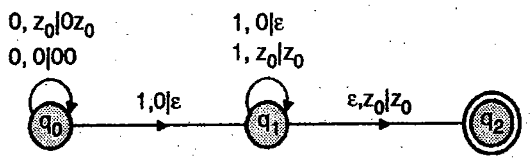


Fig. Ex. 5.4.1(b) : Transition diagram for acceptance through a final state

1. $\delta(q_0, 0, z_0) = (q_0, 0z_0)$
2. $\delta(q_0, 0, 0) = (q_0, 00)$
3. $\delta(q_0, 1, 0) = (q_1, \varepsilon)$
4. $\delta(q_1, 1, 0) = (q_1, \varepsilon)$
5. $\delta(q_1, 1, z_0) = (q_1, z_0)$
6. $\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$

Fig. Ex. 5.4.1(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. $\delta(q_0, 1, 0) = (q_1, \varepsilon)$
4. $\delta(q_1, 1, 0) = (q_1, \varepsilon)$
5. $\delta(q_1, 1, z_0) = (q_1, z_0)$
6. $\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$

Fig. Ex. 5.4.1(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. 5.4.1(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. 5.4.1(d).

Example : Processing of string 00111 by the PDA

Case I : Acceptance through empty stack.

$$(q_0, 00111, z_0) \xrightarrow{\text{(Rule 1)}} (q_0, 0111, 0z_0)$$
$$\xrightarrow{\text{(Rule 2)}} (q_0, 111, 00z_0)$$
$$\xrightarrow{\text{(Rule 3)}} (q_1, 11, 0z_0)$$
$$\xrightarrow{\text{(Rule 4)}} (q_1, 1, z_0)$$
$$\xrightarrow{\text{(Rule 5)}} (q_1, \varepsilon, z_0)$$
$$\xrightarrow{\text{(Rule 6)}} (q_1, \varepsilon, \varepsilon)$$

Case II : Acceptance through final state :

$$(q_0, 00111, z_0) \xrightarrow{\text{(Rule 1)}} (q_0, 0111, 0z_0)$$
$$\xrightarrow{\text{(Rule 2)}} (q_0, 111, 00z_0)$$
$$\xrightarrow{\text{(Rule 3)}} (q_1, 11, 0z_0)$$
$$\xrightarrow{\text{(Rule 4)}} (q_1, 1, z_0)$$
$$\xrightarrow{\text{(Rule 5)}} (q_1, \varepsilon, z_0)$$
$$\xrightarrow{\text{(Rule 6)}} (q_2, \varepsilon, z_0)$$

Example 5.4.2

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 abaabbbbaa is shown in the Fig. Ex. 5.4.2(A).

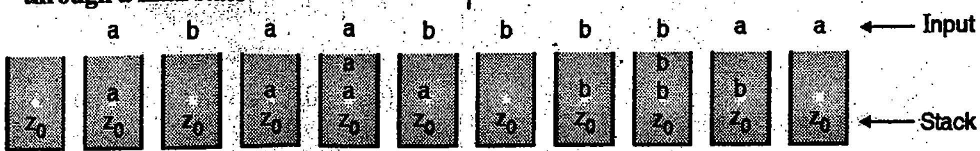


Fig. Ex. 5.4.2(A) : Stack preserving excess of a's over b's or b's over a's

Algorithm

At any point of time, there could be one the six situations :

1. Stack contains z_0(topmost) and the input is a

$\delta(q_0, a, z_0) = (q_0, az_0)$ [Extra a is pushed]

2. Stack contain z_0(topmost) and the input is b

$\delta(q_0, b, z_0) = (q_0, bz_0)$ [Extra b is pushed]

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

$\delta(q_0, b, a) = (q_0, \varepsilon)$ [Excess a's will decrease by 1]

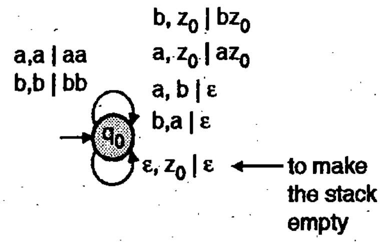
5. Stack contains b(topmost) and the input is a.

$\delta(q_0, a, b) = (q_0, \varepsilon)$ [Excess b's will decrease by 1]

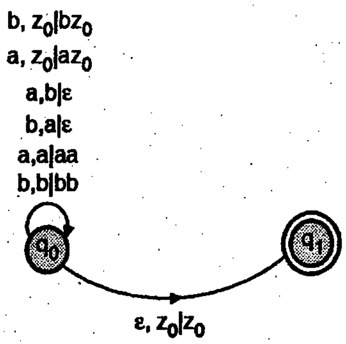
6. Stack contains b(topmost) and the input is b.

$\delta(q_0, b, b) = (q_0, bb)$ [Excess b's will increase by 1]

The PDA is shown in Fig. Ex. 5.4.2(B).



(i) Transition diagram for the PDA accepting through an empty stack



(ii) Transition diagram for the PDA accepting through a final state

$$\begin{aligned}
\delta(q_0, a, z_0) &= (q_0, az_0)\\
\delta(q_0, b, z_0) &= (q_0, bz_0)\\
\delta(q_0, a, b) &= (q_0, \varepsilon)\\
\delta(q_0, b, a) &= (q_0, \varepsilon)\\
\delta(q_0, a, a) &= (q_0, aa)\\
\delta(q_0, b, b) &= (q_0, bb)\\
\delta(q_0, \varepsilon, z_0) &= (q_0, \varepsilon)
\end{aligned}$$

(iii) Transition rules for the PDA accepting through empty stack

$$\begin{aligned}
\delta(q_0, a, z_0) &= (q_0, az_0)\\
\delta(q_0, b, z_0) &= (q_0, bz_0)\\
\delta(q_0, a, b) &= (q_0, \varepsilon)\\
\delta(q_0, b, a) &= (q_0, \varepsilon)\\
\delta(q_0, a, a) &= (q_0, aa)\\
\delta(q_0, b, b) &= (q_0, bb)\\
\delta(q_0, \varepsilon, z_0) &= (q_1, z_0)
\end{aligned}$$

(iv) Transition rules for the PDA accepting through final state

Fig. Ex. 5.4.2(B)

Fig. Ex. 5.4.2(B) ((i) to (iv)) shows state transition behaviour of PDA of example 5.4.2.

The PDA accepting through empty stack is given by :

$$M = (\{q_0\} \{a, b\} \{a, b, z_0\}, \delta, q_0, z_0, \phi)$$

where δ is given in Fig. Ex. 5.4.2(B) (iii).

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. 5.4.2(B) (iv).

Example 5.4.3

Construct pushdown automata for the following language : L = { the set of strings over alphabet {a, b} with exactly twice as many a's 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. 5.4.3. Input applied to the machine is abaabbaaaaabaab.

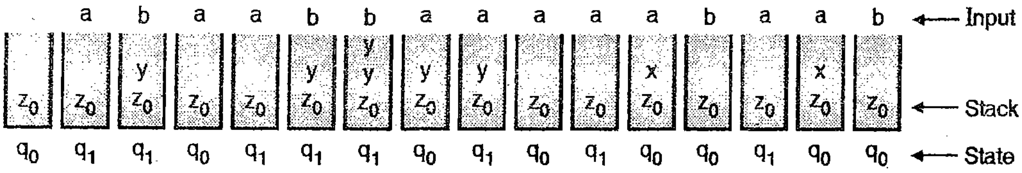


Fig. Ex. 5.4.3 : Status of stack for input abaabbaaaaabaab

Implementation 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. $\delta(q_0, b, x) = (q_0, \varepsilon)$ [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 a x for 2 a's]
8. $\delta(q_1, a, x) = (q_0, xx)$ [Push a x for 2 a's]
9. $\delta(q_1, a, y) = (q_0, \varepsilon)$

   [Two a's will remove a 'y']
10. $\delta(q_1, b, z_0) = (q_1, yz_0)$ [Excess y is saved]
11. $\delta(q_1, b, x) = (q_1, \varepsilon)$ [Excess x is removed]
12. $\delta(q_1, b, y) = (q_1, yy)$

    [Excess y increases by 1]
13. $\delta(q_0, \varepsilon, z_0) = (q_2, z_0)$

    [goes to final state to accept the string]

Example : Processing of string abaabbaaaaabaab by the PDA

$(q_0, \text{abaabbaaaaabaab}, z_0) \xrightarrow{\text{(Rule 1)}} (q_1, \text{baabbaaaaabaab}, z_0)$

$\xrightarrow{\text{(Rule 10)}} (q_1, \text{aabbaaaaabaab}, yz_0)$

$\xrightarrow{\text{(Rule 9)}} (q_0, \text{abbaaaaabaab}, z_0)$

$\xrightarrow{\text{(Rule 1)}} (q_1, \text{bbaaaaabaab}, z_0)$

$\xrightarrow{\text{(Rule 10)}} (q_1, \text{baaaaabaab}, yz_0)$

$\xrightarrow{\text{(Rule 12)}} (q_1, \text{aaaaabaab}, yyz_0)$

$\xrightarrow{\text{(Rule 9)}} (q_0, \text{aaaabaab}, yz_0)$

$\xrightarrow{\text{(Rule 4)}} (q_1, \text{aaabaab}, yz_0)$

$\xrightarrow{\text{(Rule 9)}} (q_0, \text{aabaab}, z_0)$

$\xrightarrow{\text{(Rule 1)}} (q_1, \text{abaab}, z_0)$

$\xrightarrow{\text{(Rule 7)}} (q_0, \text{baab}, xz_0)$

$\xrightarrow{\text{(Rule 5)}} (q_0, \text{aab}, z_0)$

$\xrightarrow{\text{(Rule 1)}} (q_1, \text{ab}, z_0)$

$\xrightarrow{\text{(Rule 7)}} (q_0, \text{b}, xz_0)$

$\xrightarrow{\text{(Rule 5)}} (q_0, \varepsilon, z_0)$

$\xrightarrow{\text{(Rule 13)}} (q_2, \varepsilon, z_0)$

Example 5.4.4

Give the transition table for PDA recognizing the following language

$L = \{ a^n x \mid n \geq 0 \text{ and } x \in \{a, b\}^* \text{ and } |x| \leq n \}$

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 $\delta(q_0, b, a) = (q_1, \varepsilon)$.
4. On subsequent a's or b's in x, an 'a' should be erased from the stack.

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

   $\delta(q_1, b, a) = (q_1, \varepsilon)$

5. After the end of input string, contents of stack should be erased one by one.

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

$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$

The PDA is given by :

$M = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Where, δ is given below :

$\delta(q_0, a, z_0) = (q_0, az_0)$

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

$\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$ [To accept a null string]

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

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

$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, a) = (q_1, \varepsilon)$ erase, everything from the stack

$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$

Example 5.4.5 SPPU - May 12, 8 Marks

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 :

$M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c, d\}, \{a, c, z_0\}, \delta, q_0, z_0, \phi)$

Where the transition function δ is given below :

$\delta(q_0, a, z_0) = (q_0, az_0)$ [Push the first a]

$\delta(q_0, a, a) = (q_0, aa)$ [Push remaining a's]

$\delta(q_0, b, a) = (q_1, \varepsilon)$ [Erase an 'a' on first b]

$\delta(q_1, b, a) = (q_1, \varepsilon)$

[Erase remaining a's on subsequent b's]

$\delta(q_1, c, z_0) = (q_2, cz_0)$ [First c is pushed]

$\delta(q_2, c, c) = (q_2, cc)$ [Subsequent c's are pushed]

$\delta(q_2, d, c) = (q_3, \varepsilon)$

[On first d, machine transits to q_3 with a pop]

$\delta(q_3, d, c) = (q_3, \varepsilon)$ [For every d, a 'c' is erased]

$\delta(q_3, \varepsilon, z_0) = (q_3, \varepsilon)$

[String is accepted through empty stack]

Example 5.4.6 SPPU - Dec. 15, 10 Marks

Let $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i + j = k\}$

(i) Find a PDA accepting through final state.

(ii) Find a PDA accepting through empty stack.

Solution : Algorithm

1. For every input symbol 'a', a symbol x is pushed onto the stack.
2. For every input symbol 'b', a symbol x is pushed onto the stack.
3. 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 :

1.	$\delta(q_0, a, z_0) = (q_0, xz_0)$	[x is pushed for the first a]
2.	$\delta(q_0, a, x) = (q_0, xx)$	[x is pushed for every subsequent a]
3.	$\delta(q_0, b, z_0) = (q_1, xz_0)$	[First b without a's]
4.	$\delta(q_0, b, x) = (q_1, xx)$	[First b after a's]
5.	$\delta(q_1, b, x) = (q_1, xx)$	[Subsequent b's]
6.	$\delta(q_1, c, x) = (q_2, \varepsilon)$	[x is erased for the first c]
7.	$\delta(q_2, c, x) = (q_2, \varepsilon)$	[x is erased for subsequent c's]
8.	$\delta(q_2, \varepsilon, z_0) = (q_3, z_0)$	[Accept through q_3]
9.	$\delta(q_0, \varepsilon, z_0) = (q_3, z_0)$	[Accept a null string]

(ii) PDA accepting through empty stack is given by

$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{x, z_0\}, q_0, z_0, \delta, \phi)$,

where the transition function δ is :

1. $\delta(q_0, a, z_0) = (q_0, xz_0)$
2. $\delta(q_0, a, x) = (q_0, xx)$
3. $\delta(q_0, b, z_0) = (q_1, xz_0)$
4. $\delta(q_0, b, x) = (q_1, xx)$
5. $\delta(q_1, b, x) = (q_1, xx)$
6. $\delta(q_1, c, x) = (q_2, \varepsilon)$
7. $\delta(q_2, c, x) = (q_2, \varepsilon)$
8. $\delta(q_2, \varepsilon, z_0) = (q_2, \varepsilon)$ [Stack is made empty]
9. $\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$ [Accept a null string]

Example 5.4.7 SPPU - May 12, Dec. 13, 8 Marks

(a) Construct a PDA accepting $\{a^n b^m a^n \mid m, n \geq 1\}$ by null store.

(b) Construct a PDA accepting $\{a^n \cdot b^{2n} \mid n >= 1\}$

Solution :

(a) Algorithm

1. The sequence of a's should be pushed onto the stack in state q_0

   $\delta(q_0, a, z_0) = (q_0, az_0)$

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

2. On first b, the machine moves to q_1 and remains there for b's. b's will have no effect on the stack.

3. For every 'a', an 'a' is erased from the stack.

The PDA accepting through empty stack is given by

$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

Where the transition function δ is :

1. $\delta(q_0, a, z_0) = (q_0, az_0)$ [First 'a' is pushed]
2. $\delta(q_0, a, a) = (q_0, aa)$ [Subsequent a's are pushed]
3. $\delta(q_0, b, a) = (q_1, a)$ [Input symbols b's are skipped]
4. $\delta(q_1, b, a) = (q_1, a)$
5. $\delta(q_1, a, a) = (q_2, \varepsilon)$ [An a is erased on first a of last a's]
6. $\delta(q_2, a, a) = (q_2, \varepsilon)$ [An a is erased on subsequent a's of last a's]
7. $\delta(q_2, \varepsilon, z_0) = (q_2, \varepsilon)$ [Accepting through empty stack]

(b) Algorithm

1. For every 'a' in $a^n b^{2n}$, xx are pushed on the stack.
2. For every 'b' in $a^n b^{2n}$, one x is popped from the stack.

The required PDA is given below.

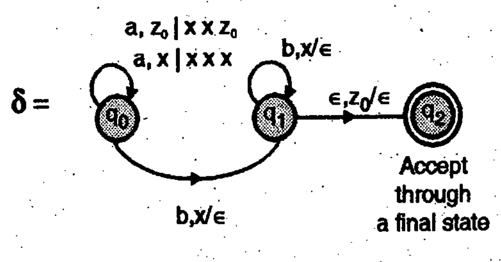


Fig. Ex. 5.4.7

Thus, the PDA

$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{x, z_0\}, \delta, q_0, z_0, \{q_2\})$

Example 5.4.8

SPPU - May 14, Dec. 14, May 15, 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. 5.4.8. Input applied is abbcbba.

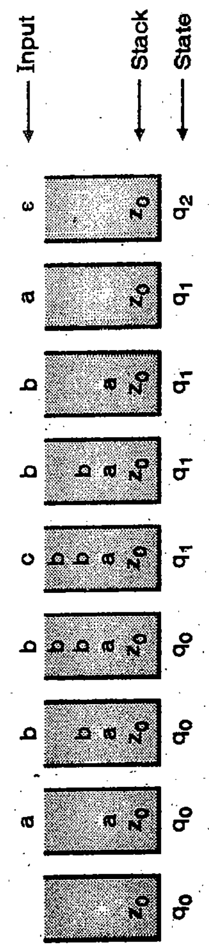


Fig. Ex. 5.4.8 : A PDA on input abbcbba

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. $\delta(q_0, a, \varepsilon) = (q_0, a)$ First n symbols are pushed onto the stack
2. $\delta(q_0, b, \varepsilon) = (q_0, b)$
3. $\delta(q_0, c, \varepsilon) = (q_1, \varepsilon)$ [State changes on c]
4. $\delta(q_1, a, a) = (q_1, \varepsilon)$ Last n symbols are matched with first n symbols in reverse order
5. $\delta(q_1, b, b) = (q_1, \varepsilon)$
6. $\delta(q_1, \varepsilon, z_0) = (q_2, z_0)$ [Accepted through final state]

A transition of the form $\delta(q_0, a, \varepsilon) = (q_0, a)$ implies that always push a, irrespective of stack symbol.

Example 5.4.9

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 :

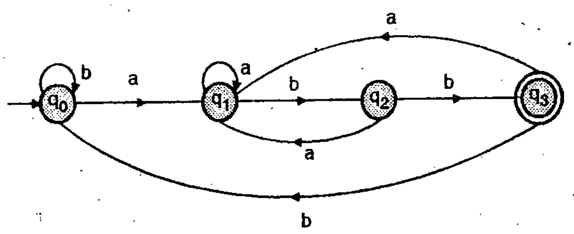


Fig. Ex. 5.4.9(a) : A DFA for string ending in abb

Step 2 : From DFA to PDA

– 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 :

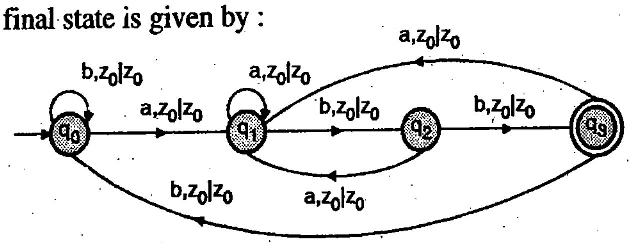


Fig. Ex. 5.4.9(b) : A PDA constructed from DFA

$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{z_0\}, \delta, q_0, z_0, \{q_3\})$

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 5.4.10 SPPU - Dec. 14. 8 Marks

Construct PDA for the given grammar containing

(i)	$S \to BD$	(ii)	$D \to SC$
(iii)	$C \to AA$	(iv)	$S \to BC$
(v)	$B \to 0$	(vi)	$A \to 1$

Solution : The equivalent PDA, M is given by :

$M = \{\{q\}, \{0, 1\}, \{0, 1, S, A, B, D, C\}, \delta, q, S, \phi\}$

Where δ is given by :

$\delta(q, \in, S) = \{(q, BD), (q, BC)\}$	$\delta(q, \in, B) = (q, 0)$
$\delta(q, \in, D) = \{q, SC\}$	$\delta(q, \in, A) = (q, 1)$
$\delta(q, \in, C) = \{(q, AA)$	$\delta(q, 0, 0) = (q, \in)$
	$\delta(q, 1, 1) = (q, \in)$

Example 5.4.11 SPPU - May 15. 10 Marks

Construct transition table for PDA that accepts the language $L = \{a^{2n} b^n \mid n >= 1\}$. Trace your PDA for the input with n = 3.

Solution :

1. For every pair of as z, one x is pushed onto the stack.
2. For every b, one x is popped out from the stack.
3. Final the stack should contain the initial stack symbol z_0.

Transition table

1. $\delta(q_0, a, z_0) = (q_1, z_0)$
2. $\delta(q_1, a, z_0) = (q_0, x z_0)$
3. $\delta(q_0, a, x) = (q_1, x)$
4. $\delta(q_1, a, x) = (q_0, x x)$
5. $\delta(q_0, b, x) = (q_2, \in)$
6. $\delta(q_2, b, x) = (q_2, \in)$
7. $\delta(q_2, \in, z_0) = (q_2, \in)$

accepting through empty stack.

Tracing PDA for a^6b^3

$\delta(q_0, aaaaaabbb, z_0) \overset{\text{Rule 1}}{\Rightarrow} (q_1, aaaaabbb, z_0)$

$\overset{\text{Rule 2}}{\Rightarrow} (q_0, aaaabbb, xz_0)$

$\overset{\text{Rule 3}}{\Rightarrow} (q_1, aaabbb, xz_0)$

$\overset{\text{Rule 4}}{\Rightarrow} (q_0, aabbb, xxz_0)$

$\overset{\text{Rule 3}}{\Rightarrow} (q_1, abbb, xxz_0)$

$\overset{\text{Rule 4}}{\Rightarrow} (q_0, bbb, xxxz_0)$

$\overset{\text{Rule 5}}{\Rightarrow} (q_2, bb, xxz_0)$

$\overset{\text{Rule 6}}{\Rightarrow} (q_2, b, xz_0)$

Rule 6 $\Rightarrow$ $(q_2, \in, z_0)$

Rule 7 $\Rightarrow$ $(q_2, \in, \in)$

Example 5.4.12 SPPU - Dec. 12, 6 Marks

Construct a PDA for following CFG.

$S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS$

Solution : The equivalent PDA, M is given by :

$$M = (\{q\}, \{a, b\}, \{a, b, s\}, \delta, q, s, \phi)$$

Where δ is given by :

$$\delta(q, \in, S) \Rightarrow \{(q, a), (q, aS), (q, bSS), (q, SSb), (q, SbS)\}$$

$$\delta(q, a, a) = \{(q, \in)\}$$

$$\delta(q, b, b) = \{(q, \in)\}$$

Example 5.4.13 SPPU - Dec. 12, 8 Marks

Design a PDA accepting balanced strings of brackets involving two types of brackets : { } and [ ]

Solution : δ is given by :

$\delta(q_0, \{, z_0) = (q_0, \{z_0)$

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

$\delta(q_0, \{, [) = (q_0, \{[)$

$\delta(q_0, [, z_0) = (q_0, [z_0)$

$\delta(q_0, [, \{) = (q_0, [\{)$

$\delta(q_0, [, [) = (q_0, [[)$

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

$\delta(q_0,], [) = (q_0, \in)$

$\delta(q_0, \in, z_0) = (q_0, \in)$

(Accepting through the final state)

z_0 is the initial stack symbol.

Example 5.4.14 SPPU - May 13, May 15, 6 Marks

Design a PDA to check whether the given expression is a valid Postfix expression.

Solution : Algorithm

1. For every operand, X is pushed into the stack.
2. For every operator, one X is popped out from the stack.

   Finally, the stack should contain Z_0X.

Assumption

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

$\delta(q_0,a,Z_0) = (q_0,XZ_0)$

$\delta(q_0,b,Z_0) = (q_0,XZ_0)$

$\delta(q_0,a,X) = (q_0,XX)$

$\delta(q_0,b,X) = (q_0,XX)$

$\delta(q_0,+,X) = (q_0,\in)$

$\delta(q_0,*,X) = (q_0,\in)$

$\delta(q_0,\in,X) = (q_1,\in)$

$\delta(q_1,\in,Z_0) = (q_2,\in)$

$\Rightarrow$ Accept through the empty stack

Where Z_0 = Initial stack symbol

Example 5.4.15 SPPU - May 15, 8 Marks

Construct PDA for the following regular grammar : $S \rightarrow 0A \mid 1B \mid 0 \quad A \rightarrow A0 \mid B \quad B \rightarrow c \mid d$

Solution :

Step 1 : Removing unit production $A \rightarrow B$, we get

$S \rightarrow 0A \mid 1B \mid 0$

$A \rightarrow A0 \mid c \mid d$

$B \rightarrow c \mid d$

Removing left recursion from A-productions, we get

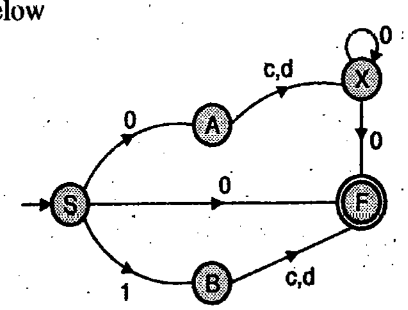
$S \rightarrow 0A \mid 1B \mid 0$

$A \rightarrow CX \mid dx$

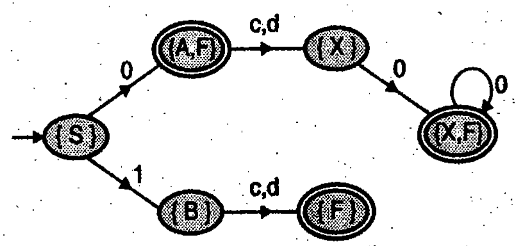
$X \rightarrow 0X \mid 0$

$B \rightarrow c \mid d$

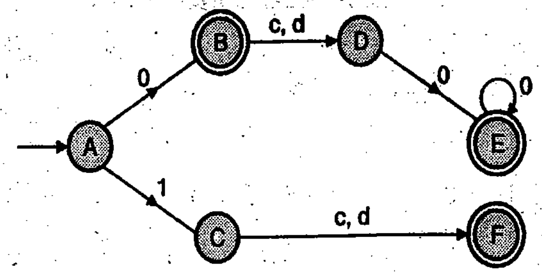
Transition diagram for the above grammar is given below



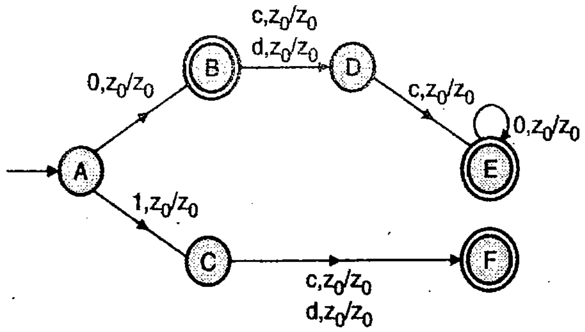
Drawing all equivalent DFA, we get



Renaming the variables, we get



We can construct a PDA from the above DFA.



The PDA, M $= (\{A, B, C, D, E, F\}, \{0, 1, c, d\}, \{z_0\}, \delta, A, z_0, \{B, E, F\})$

Where the transition function δ is given below

1. $\delta\,(A, 0, z_0) = (B, z_0)$
2. $\delta\,(A, 1, z_0) = (C, z_0)$
3. $\delta\,(B, C, z_0) = (D, z_0)$
4. $\delta\,(B, d, z_0) = (D, z_0)$
5. $\delta\,(C, C, z_0) = (F, z_0)$
6. $\delta\,(C, d, z_0) = (F, z_0)$
7. $\delta\,(D, 0, z_0) = (E, z_0)$
8. $\delta\,(E, 0, z_0) = (E, z_0)$

5.5 Non-deterministic PDA (NPDA)

SPPU - May 13, May 15, May 16, Dec. 16

University Questions

Q. Compare deterministic PDA with non-deterministic PDA. **(May 2013, 4 Marks)**

Q. What are the different types of PDA ? **(May 2015, May 2016, Dec. 2016, 2 Marks)**

Q. What is NPDA ? **(May 2016, Dec. 2016, 2 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. Whereas, 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 5.5.1

Design a PDA for detection of odd palindrome over {a, b}.

Solution : An odd palindrome will be of the form

1. waw^R $\quad \underbrace{ab}_{w}\,a\,\underbrace{ba}_{w^R},\ \underbrace{aba}_{w}\,a\,\underbrace{aba}_{w^R},\ \underbrace{aa}_{w}\,a\,\underbrace{aa}_{w^R}$
2. wbw^R $\quad \underbrace{ab}_{w}\,b\,\underbrace{ba}_{w^R},\ \underbrace{aba}_{w}\,b\,\underbrace{aba}_{w^R},\ \underbrace{aa}_{w}\,b\,\underbrace{aa}_{w^R}$

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 :
   $\delta(q_0, a, \varepsilon) \ \Rightarrow \ (q_0, a)$
   $\delta(q_0, b, \varepsilon) \ \Rightarrow \ (q_0, b)$
   (b) It is a middle character – go for matching of second half with the first half.
   $\delta(q_0, a, \varepsilon) \ \Rightarrow \ (q_1, \varepsilon)$
   $\delta(q_0, b, \varepsilon) \ \Rightarrow \ (q_1, \varepsilon)$

The status of the stack and the state of the machine is shown in the Fig. Ex. 5.5.1. Input applied is ababa.

- Left child $\rightarrow$ current input is taken as the middle character
- Right child $\rightarrow$ current input is not a middle character.

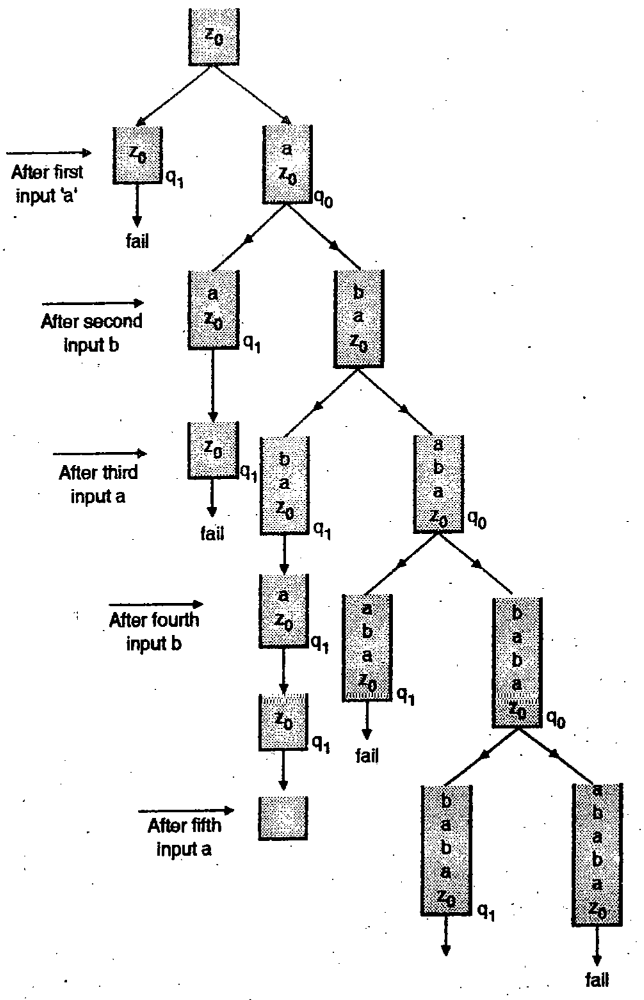


Fig. Ex. 5.5.1 : Processing of string by the PDA. String is taken as "ababa"

The transition table for the PDA is given below

$\delta(q_0, a, \varepsilon) \Rightarrow \{(q_1, \varepsilon), (q_0, a)\}$

ε indicates that irrespective of the current stack symbol, perform the transition.

$\delta(q_0, b, \varepsilon) \Rightarrow \{(q_1, \varepsilon), (q_0, b)\}$

$\delta(q_1, a, a) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, b, b) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, z_0) \Rightarrow \{(q_1, \varepsilon)\}$ [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 5.5.2 SPPU - May 16, Dec. 16, 5 Marks

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 :

$$\underbrace{ab}_{w}\underbrace{ba}_{w^R},\quad \underbrace{aba}_{w}\underbrace{aba}_{w^R},\quad \underbrace{aa}_{w}\underbrace{aa}_{w^R}$$

Centre Centre Centre

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 :
   $\delta(q_0, a, a) \Rightarrow (q_1, \varepsilon)$
   $\delta(q_0, b, b) \Rightarrow (q_1, \varepsilon)$
   must be identical
   (b) It belongs to first half.
   Push the current input
   $\delta(q_0, a, \varepsilon) \Rightarrow (q_0, a)$ $\delta(q_0, b, \varepsilon) \Rightarrow (q_0, b)$
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 :

$\delta(q_0, a, z_0) \Rightarrow \{(q_0, az_0)\}$

$\delta(q_0, b, z_0) \Rightarrow \{(q_0, bz_0)\}$

$\delta(q_0, a, a) \Rightarrow \{(q_0, aa)\ (q_1, \varepsilon)\}$

$\delta(q_0, a, b) \Rightarrow \{(q_0, ab)\}$

$\delta(q_0, b, a) \Rightarrow \{(q_0, ba)\}$

$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, \varepsilon)\}$

$\delta(q_1, a, a) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, b, b) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, z_0) \Rightarrow \{(q_1, \varepsilon)\}$ [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 5.5.3 SPPU - Dec. 12. 8 Marks

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 :

$\delta(q_0, a, z_0) \Rightarrow \{(q_1, z_0), (q_0, az_0)\}$

$\delta(q_0, b, z_0) \Rightarrow \{(q_1, z_0), (q_0, bz_0)\}$

$\delta(q_0, a, a) \Rightarrow \{(q_0, aa)\ (q_1, a), (q_1, \varepsilon)\}$

$\delta(q_0, a, b) \Rightarrow \{(q_0, ab), (q_1, b)\}$

$\delta(q_0, b, a) \Rightarrow \{(q_0, ba), (q_1, a)\}$

$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, b), (q_1, \varepsilon)\}$

$\delta(q_1, a, a) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, b, b) \Rightarrow \{(q_1, \varepsilon)\}$

$\delta(q_1, \varepsilon, z_0) \Rightarrow \{(q_1, \varepsilon)\}$

[Accept through an empty stack].

Details of important transitions

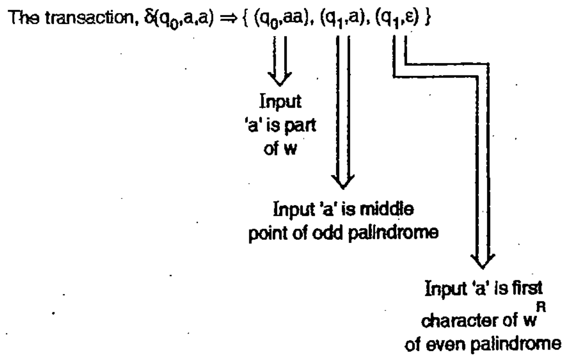


Fig. Ex. 5.5.3

The transition rule for δ (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 waw^R
3. Input 'a' is the first character of w^R.

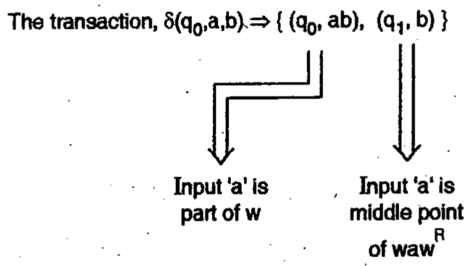


Fig. Ex. 5.5.3(a)

Example 5.5.4

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

$$\delta(q_0, a, z_0) = \{(q_1, aa), (q_2, z_0)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_3, \varepsilon)\}$$
$$\delta(q_3, b, z_0) = \{(q_f, z_0)\}$$
$$\delta(q_3, c, a) = \{(q_f, \varepsilon)\}$$
$$\delta(q_3, b, a) = \{(q_3, \varepsilon)\}$$
$$\delta(q_2, a, z_0) = \{(q_2, \varepsilon)\}$$
$$\delta(q_2, b, z_0) = \{(q_4, bz_0)\}$$
$$\delta(q_4, b, b) = \{(q_4, bb)\}$$
$$\delta(q_4, c, b) = \{(q_4, \varepsilon)\}$$
$$\delta(q_4, \varepsilon, b) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, b, \varepsilon) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, c, \varepsilon) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, \varepsilon, \varepsilon) = \{(q_f, \varepsilon)\}$$

Syllabus Topic
PDA and Context Free Language, Equivalence of PDA and CFG

5.6 Pushdown 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.

- It is possible to find a PDA for a CFG
- It is possible to find a CFG for a PDA.

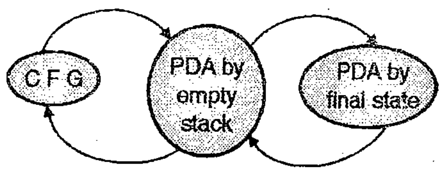


Fig. 5.6.1 : Equivalence of PDA and CFG

5.6.1 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 :

$$M = (\{q\}, T, V \cup T, \delta, q, S, \phi)$$

[M is a PDA for L(G)]

Where δ is defined by :

1. For each variable $A \in V$, include a transition,
$\delta(q, \varepsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha$ is a production in G$\}$
2. For each terminal $a \in T$, include a transition
$\delta(q, a, a) \Rightarrow \{(q, \varepsilon)\}$

Example 5.6.1

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 :

$$\delta(q, \varepsilon, S) = \{(q, 0S1), (q, 00), (q, 11)\}$$
$$\delta(q, 0, 0) = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Example 5.6.2 SPPU - May 12, 5 Marks

Convert the grammar $S \rightarrow 0S1 \mid A \quad A \rightarrow 1A0 \mid S \mid \varepsilon$ to PDA that accepts the same language by empty stack.

Solution :

Step 1 : For each variable $A \in V$, include a transition
$\delta(q, \varepsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha$ is a production in G $\}$;

$$\delta(q, \varepsilon, S) \Rightarrow \{(q, 0S1), (q, A)\}$$
$$\delta(q, \varepsilon, A) \Rightarrow \{(q, 1A0), (q, S), (q, \varepsilon)\}$$

Step 2 : For each terminal $a \in T$, include a transition
$\delta(q, a, a) \Rightarrow (q, \varepsilon)$

$$\therefore \delta(q, 0, 0) = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Therefore, the PDA is given by :

$$M = (\{q\}, \{0, 1\}, \{S, A, 0, 1\}, \delta, q, S, \phi)$$

where δ is : $\delta(q, \varepsilon, S) = \{(q, 0S1), (q, A)\}$

$\delta(q, \varepsilon, A) = \{(q, 1A0), (q, S), (q, \varepsilon)\}$

$\delta(q, 0, 0) = \{(q, \varepsilon)\}$

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

Example 5.6.3

Let G be the grammar given by S →aABB I aAA, A → aBB I a, B → bBB I 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 :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, aABB), (q, aAA)\}$
$\delta(q, \varepsilon, A) \Rightarrow \{(q, aBB), (q, a)\}$
$\delta(q, \varepsilon, B) \Rightarrow \{(q, bBB), (q, A)\}$ — For each production in given grammar

$\delta(q, a, a) \Rightarrow (q, \varepsilon)$
$\delta(q, b, b) \Rightarrow (q, \varepsilon)$ — For each terminal in T.

Example 5.6.4 SPPU - May 12, May 13. 5 Marks

Construct a PDA equivalent to the following CFG. S → 0BB B → 0S I 1S I 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 :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, 0BB)\}$
$\delta(q, \varepsilon, B) \Rightarrow \{(q, 0S), (q, 1S), (q, 0)\}$ — For each production in the given grammar

$\delta(q, 0, 0) \Rightarrow \{(q, \varepsilon)\}$
$\delta(q, 1, 1) \Rightarrow \{(q, \varepsilon)\}$ — For each terminal.

Acceptance of 010^4 by M

$\delta(q, 010000, S) \xrightarrow{\delta(q, \varepsilon, S) = (q, 0BB)} (q, 010000, 0BB)$

$\xrightarrow{\delta(q, 0, 0) = (q, \varepsilon)} (q, 10000, BB)$

$\xrightarrow{\delta(q, \varepsilon, B) = (q, 1S)} (q, 10000, 1SB)$

$\xrightarrow{\delta(q, 1, 1) = (q, \varepsilon)} (q, 0000, SB)$

$\xrightarrow{\delta(q, \varepsilon, S) = (q, 0BB)} (q, 0000, 0BBB)$

$\xrightarrow{\delta(q, 0, 0) = (q, \varepsilon)} (q, 000, BBB)$

$\xrightarrow{\delta(q, \varepsilon, B) = (q, 0)} (q, 000, 0BB)$

$\xrightarrow{\delta(q, 0, 0) = (q, \varepsilon)} (q, 00, BB)$

$\xrightarrow{\delta(q, \varepsilon, B) = (q, 0)} (q, 00, 0B)$

$\xrightarrow{\delta(q, 0, 0) = (q, \varepsilon)} (q, 0, B)$

$\xrightarrow{\delta(q, \varepsilon, B) = (q, 0)} (q, 0, 0)$

$\xrightarrow{\delta(q, 0, 0) = (q, \varepsilon)} (q, \varepsilon, \varepsilon)$

Thus the string 010^4 is accepted by M using an empty stack.

$\therefore \quad 010^4 \in L$

Example 5.6.5

Design a PDA to recognize the language generated by the following grammar :

S → S + S I S * S I 4 I 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 :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, S + S), (q, S * S), (q, 4), (q, 2)\}$ I for every production in G

$\delta(q, +, +) = \{(q, \varepsilon)\}$
$\delta(q, *, *) = \{(q, \varepsilon)\}$
$\delta(q, 2, 2) = \{(q, \varepsilon)\}$
$\delta(q, 4, 4) = \{(q, \varepsilon)\}$ — for every terminal in T.

Acceptance of 2 + 2 * 4 by this PDA

$\delta(q, 2+2*4, S) \xrightarrow{\delta(q, \varepsilon S) \Rightarrow (q, S+S)} (q, 2+2*4, S+S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, 2)} (q, 2+2*4, 2+S)$

$\xrightarrow{\delta(q, 2, 2) \Rightarrow (q, \varepsilon)} (q, +2*4, +S)$

$\xrightarrow{\delta(q, +, +) \Rightarrow (q, \varepsilon)} (q, 2*4, S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, S*S)} (q, 2*4, S*S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, 2)} (q, 2*4, 2*S)$

$\delta(q, 2, 2) \Rightarrow (q, \varepsilon)$

$\longrightarrow (q, *4, *S)$

$\delta(q, *, *) \Rightarrow (q, \varepsilon)$

$\longrightarrow (q, 4, S)$

$\delta(q, \varepsilon, S) = (q, 4)$

$\longrightarrow (q, 4, 4)$

$\delta(q, 4, 4) = (q, \varepsilon)$

$\longrightarrow (q, \varepsilon, \varepsilon)$

Example 5.6.6

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,

$\delta(q, \varepsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$

2. For each terminal $a \in T$, include a transition

$$\delta(q, a, a) \Rightarrow \{(q, \varepsilon)\}$$

- The transitions of M are designed to simulate a leftmost derivation of a string.

1. The transition of the form

$$\delta(q, \varepsilon, A) \Rightarrow (q, \alpha)$$

is for expansion of the topmost variable of the stack.

2. The transition of the form

$$\delta(q, a, a) \Rightarrow (q, \varepsilon)$$

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 $w \in L(G)$.

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.

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = w$$

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.

Let $\alpha_i = x_i \beta_i$ where $x_i \in T^*$ and $B_i \in (V \cup T)^*$

The string α_i corresponds to a unique configuration of PDA given by a pair :

$$(y_i, \beta_i)$$

The corresponding diagram is shown in Fig. Ex. 5.6.6.

- 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.

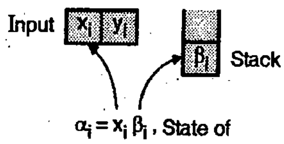
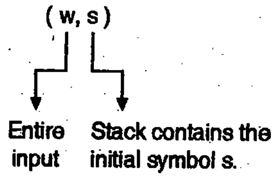


Fig. Ex. 5.6.6 : Unique configuration of PDA for α_i

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 $(i + 1)_{th}$ 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

$$\delta(q, a_{i+1}, a_{i+1}) \Rightarrow (q, \varepsilon)$$

Thus, there is one-to-one correspondence between the string α_i and the configuration of PDA. Thus a string $w \in L(G)$ will be accepted by the PDA M.

5.6.2 Construction of CFG from PDA

SPPU - May 12

University Question

Q. Describe in brief : Construction of CFG from PDA. **(May 2012, 5 Marks)**

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 :

$[p^X q]$, where $p, q \in Q$ 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.

   $S \rightarrow [q_0{}^z q_i]$ for each $q_i \in Q$, where z 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 $(\Sigma \cup \varepsilon)$

   (c) B and C belong to $(\Gamma \cup \varepsilon)$

   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 belongs to $(\Sigma \cup \varepsilon)$

   (c) B, C_1 and C_2 belongs to Γ

   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 5.6.7 SPPU - Dec. 12, 10 Marks

Convert PDA to CFG. PDA is given by

M = ({p, q}, {0, 1}, {x, z}, δ, q, z), transition function δ is defined by :

$\delta(q, 1, z) \Rightarrow \{(q, xz)\}$

$\delta(q, 1, x) \Rightarrow \{(q, xx)\}$

$\delta(q, \varepsilon, x) \Rightarrow \{(q, \varepsilon)\}$

$\delta(q, 0, x) \Rightarrow \{(p, x)\}$

$\delta(p, 1, x) \Rightarrow \{(p, \varepsilon)\}$

$\delta(p, 0, z) \Rightarrow \{(q, z)\}$

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

$\delta(q, \varepsilon, x) \Rightarrow \{(q, \varepsilon)\}$

$[q^x q] \rightarrow \varepsilon$

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

$\delta(p, 1, x) \Rightarrow \{(p, \varepsilon)\}$

$[p^x p] \rightarrow 1$

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
$[q \overset{z}{} q]$	A
$[q \overset{z}{} p]$	B
$[p \overset{z}{} q]$	C
$[p \overset{z}{} p]$	D
$[q \overset{x}{} q]$	E
$[q \overset{x}{} p]$	F
$[p \overset{x}{} q]$	G
$[p \overset{x}{} p]$	H

The set of productions can be written as :

$S \rightarrow A \mid B$
$A \rightarrow 1EA \mid 1FC$
$B \rightarrow 1EB \mid 1FD$
$E \rightarrow 1EE \mid 1FG$
$F \rightarrow 1EF \mid 1FH$
$E \rightarrow \varepsilon$
$E \rightarrow 0G$
$F \rightarrow 0H$
$H \rightarrow 1$
$C \rightarrow 0A$
$D \rightarrow 0B$

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 :

$S \rightarrow A \mid B$
$A \rightarrow 1EA \mid 1FC$
$B \rightarrow 1EB \mid 1FD$
$E \rightarrow 1EE \mid \varepsilon$
$F \rightarrow 1EF \mid 1FH \mid 0H$
$H \rightarrow 1$
$C \rightarrow 0A$
$D \rightarrow 0B$

Example 5.6.8 SPPU - Dec. 13, May 15, May 16, Dec. 16, 9/12 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 :

$\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$
$\delta(q_0, 1, x) = \{(q_0, xx)\}$, $\delta(q_0, 0, x) = \{(q_1, x)\}$
$\delta(q_0, \varepsilon, z_0) = \{(q_0, \varepsilon)\}$, $\delta(q_1, 1, x) = \{(q_1, \varepsilon)\}$
$\delta(q_1, 0, z_0) = \{(q_0, z_0)\}$

Solution :

Step 1 : Add productions for the start symbol

$$S \rightarrow [q_0 \overset{z_0}{} q_0]$$
$$S \rightarrow [q_0 \overset{z_0}{} q_1]$$

Step 2 : Add productions for $\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$

$$[q_0 \overset{z_0}{} q_0] \rightarrow 1\,[q_0 \overset{x}{} q_0]\,[q_0 \overset{z_0}{} q_0]$$
$$[q_0 \overset{z_0}{} q_0] \rightarrow 1\,[q_0 \overset{x}{} q_1]\,[q_1 \overset{z_0}{} q_0]$$
$$[q_0 \overset{z_0}{} q_1] \rightarrow 1\,[q_0 \overset{x}{} q_0]\,[q_0 \overset{z_0}{} q_1]$$
$$[q_0 \overset{z_0}{} q_1] \rightarrow 1\,[q_0 \overset{x}{} q_1]\,[q_1 \overset{z_0}{} q_1]$$

Step 3 : Add productions for $\delta(q_0, 1, x) \Rightarrow \{(q_0, xx)\}$

$$[q_0 \overset{x}{} q_0] \rightarrow 1\,[q_0 \overset{x}{} q_0]\,[q_0 \overset{x}{} q_0]$$
$$[q_0 \overset{x}{} q_0] \rightarrow 1\,[q_0 \overset{x}{} q_1]\,[q_1 \overset{x}{} q_0]$$
$$[q_0 \overset{x}{} q_1] \rightarrow 1\,[q_0 \overset{x}{} q_0]\,[q_0 \overset{x}{} q_1]$$
$$[q_0 \overset{x}{} q_1] \rightarrow 1\,[q_0 \overset{x}{} q_1]\,[q_1 \overset{x}{} q_1]$$

Step 4 : Add productions for $\delta(q_0, 0, x) \Rightarrow \{(q_1, x)\}$

$$[q_0 \overset{x}{} q_0] \rightarrow 0\,[q_1 \overset{x}{} q_0]$$
$$[q_0 \overset{x}{} q_1] \rightarrow 0\,[q_1 \overset{x}{} q_1]$$

Step 5 : Add productions for $\delta(q_0, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

$$[q_0 \overset{z_0}{} q_1] \rightarrow \varepsilon$$

Step 6 : Add production for $\delta(q_1, 1, x) \Rightarrow \{(q_1, \varepsilon)\}$

$$[q_1 \overset{x}{} q_1] \rightarrow 1$$

Step 7 : Add productions for $\delta(q_1, 0, z_0) \Rightarrow \{(q_0, z_0)\}$

$$[q_1 \overset{z_0}{} q_0] \Rightarrow 0\,[q_0 \overset{z_0}{} q_0]$$
$$[q_1 \overset{z_0}{} q_1] \Rightarrow 0\,[q_0 \overset{z_0}{} q_1]$$

Example 5.6.9 SPPU - May 14, Dec. 15, 8/12 Marks

Consider the PDA with the following moves :

$\delta(q_0, a, z_0) = \{(q_0, az_0)\}$, $\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$, $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

Obtain CFG equivalent to PDA.

Solution :

Step 1 : Add productions for the start symbol.

$$S \rightarrow [q_0 \overset{z_0}{} q_0]$$
$$S \rightarrow [q_0 \overset{z_0}{} q_1]$$

Step 2 : Add productions for $(q_0, a, a) = \{(q_0, aa)\}$

$$[q_0 \overset{a}{} q_0] \rightarrow a\,[q_0 \overset{a}{} q_0]\,[q_0 \overset{a}{} q_0]$$
$$[q_0 \overset{a}{} q_0] \rightarrow a\,[q_0 \overset{a}{} q_1]\,[q_1 \overset{a}{} q_0]$$
$$[q_0 \overset{a}{} q_1] \rightarrow a\,[q_0 \overset{a}{} q_0]\,[q_0 \overset{a}{} q_1]$$
$$[q_0 \overset{a}{} q_1] \rightarrow a\,[q_0 \overset{a}{} q_1]\,[q_1 \overset{a}{} q_1]$$

Step 3 : Add productions for $\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$

$$[q_0 \overset{a}{} q_1] \rightarrow b$$

Step 4 : Add productions for $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$

$$[q_1 \ ^{a}\ q_1] \quad \rightarrow b$$

Step 5 : Add productions for $\delta(q_1, \varepsilon, z_0) \rightarrow \{(q_1, \varepsilon)\}$

$$[q_1 \ ^{z_0}\ q_1] \quad \rightarrow \varepsilon$$

Example 5.6.10

For the PDA

$(\{ q_0, q_1 \}, \{ 0, 1 \}, \{ 0, 1, z_0 \}, \delta, q_0, z_0, \phi)$ where δ is

$\delta(q_0, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

$\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$

$\delta(q_0, 0, 0) = \{(q_0, 00)\}$

$\delta(q_0, 1, 0) = \{(q_0, 10)\}$

$\delta(q_0, 1, 1) = \{(q_0, 11)\}$

$\delta(q_0, 0, 1) = \{(q_1, \varepsilon)\}$

$\delta(q_1, 0, 1) = \{(q_1, \varepsilon)\}$

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

$\delta(q_1, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

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.	Productions due to start symbol S.	$S \rightarrow [q_0 \ ^{Z_0}\ q_0]$
		$S \rightarrow [q_0 \ ^{Z_0}\ q_1]$
2.	$\delta(q_0, \varepsilon, Z_0) = (q_1, \varepsilon)$	$[q_0 \ ^{Z_0}\ q_1] \rightarrow \varepsilon$
3.	$\delta(q_0, 0, Z_0) = (q_0, 0Z_0)$	$[q_0 \ ^{Z_0}\ q_0] \rightarrow 0\,[q_0 \ ^{0}\ q_0]\,[q_0 \ ^{Z_0}\ q_0]$
		$[q_0 \ ^{Z_0}\ q_0] \rightarrow 0\,[q_0 \ ^{0}\ q_1]\,[q_1 \ ^{Z_0}\ q_0]$
		$[q_0 \ ^{Z_0}\ q_1] \rightarrow 0\,[q_0 \ ^{0}\ q_0]\,[q_0 \ ^{Z_0}\ q_1]$
		$[q_0 \ ^{Z_0}\ q_1] \rightarrow 0\,[q_0 \ ^{0}\ q_1]\,[q_1 \ ^{Z_0}\ q_1]$
	$\delta(q_0, 0, 0) = (q_0, 00)$	$[q_0 \ ^{0}\ q_0] \rightarrow 0\,[q_0 \ ^{0}\ q_0]\,[q_0 \ ^{0}\ q_0]$
		$[q_0 \ ^{0}\ q_0] \rightarrow 0\,[q_0 \ ^{0}\ q_1]\,[q_1 \ ^{0}\ q_0]$
		$[q_0 \ ^{0}\ q_1] \rightarrow 0\,[q_0 \ ^{0}\ q_0]\,[q_0 \ ^{0}\ q_1]$
		$[q_0 \ ^{0}\ q_1] \rightarrow 0\,[q_0 \ ^{0}\ q_1]\,[q_1 \ ^{0}\ q_1]$
	$\delta(q_0, 1, 0) = (q_0, 10)$	$[q_0 \ ^{0}\ q_0] \rightarrow 1\,[q_0 \ ^{1}\ q_0]\,[q_0 \ ^{0}\ q_0]$
		$[q_0 \ ^{0}\ q_0] \rightarrow 1\,[q_0 \ ^{1}\ q_1]\,[q_1 \ ^{0}\ q_0]$
		$[q_0 \ ^{0}\ q_1] \rightarrow 1\,[q_0 \ ^{1}\ q_0]\,[q_0 \ ^{0}\ q_1]$
		$[q_0 \ ^{0}\ q_1] \rightarrow 1\,[q_0 \ ^{1}\ q_1]\,[q_1 \ ^{0}\ q_1]$
	$\delta(q_0, 1, 1) = (q_0, 11)$	$[q_0 \ ^{1}\ q_0] \rightarrow 1\,[q_0 \ ^{1}\ q_0]\,[q_0 \ ^{1}\ q_0]$
		$[q_0 \ ^{1}\ q_0] \rightarrow 1\,[q_0 \ ^{1}\ q_1]\,[q_1 \ ^{1}\ q_0]$
		$[q_0 \ ^{1}\ q_1] \rightarrow 1\,[q_0 \ ^{1}\ q_0]\,[q_0 \ ^{1}\ q_1]$
		$[q_0 \ ^{1}\ q_1] \rightarrow 1\,[q_0 \ ^{1}\ q_1]\,[q_1 \ ^{1}\ q_1]$
	$\delta(q_0, 0, 1) = (q_1, \varepsilon)$	$[q_0 \ ^{1}\ q_1] \rightarrow 0$
	$\delta(q_1, 0, 1) = (q_1, \varepsilon)$	$[q_1 \ ^{1}\ q_1] \rightarrow 0$
	$\delta(q_1, 0, 0) = (q_1, \varepsilon)$	$[q_1 \ ^{0}\ q_1] \rightarrow 0$
	$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$	$[q_1 \ ^{Z_0}\ q_1] \rightarrow \varepsilon$

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 :

$S \rightarrow A \mid B$	$B \rightarrow \varepsilon$
$A \rightarrow 0EA \mid 0FC$	$B \rightarrow 0EB \mid 0FD$
$E \rightarrow 0EE \mid 0FG$	$F \rightarrow 0EF \mid 0FH$
$E \rightarrow 1IE \mid 1JG$	$F \rightarrow 1IF \mid 1JH$
$I \rightarrow 1II \mid 1JK$	$J \rightarrow 1IJ \mid 1JL$
$J \rightarrow 0$	$L \rightarrow 0$
$H \rightarrow 0$	$D \rightarrow \varepsilon$

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 :

$S \rightarrow B$

$B \rightarrow 0F$

$F \rightarrow 0FH \mid 1JH$

$H \rightarrow 0$

$J \rightarrow 1JL \mid 0$

$L \rightarrow 0$

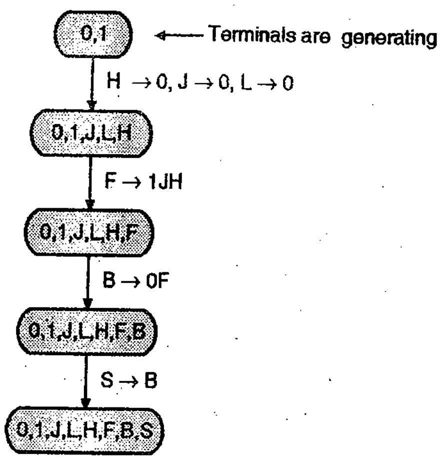


Fig. Ex. 5.6.10

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 :

$S \rightarrow 0F$ $\quad$ $F \rightarrow 0FH \mid 1JH$

$H \rightarrow 0$ $\quad$ $J \rightarrow 1JL \mid 0$

$L \rightarrow 0$

Language accepted by the PDA : The language accepted by the PDA is given by :

$$L = \{0^{n} 1^{m} 0^{n+m} \mid n, m \geq 1\} \cup \varepsilon$$

Example 5.6.11

Design a PDA and then corresponding CFG for the language that accepts the simple palindrome. $L = \{xcx^R \mid x \in \{a,b\}^*\}$

Solution : Transitions for the PDA are given by :

$\delta(q_0, a, z) = (q_0, az)$

$\delta(q_0, b, z) = (q_0, bz)$

$\delta(q_0, c, z) = (q_0, \varepsilon)$

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

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, a) = (q_0, ba)$

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

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

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

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

$\delta(q_1, b, b) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, z) = (q_1, \varepsilon)$

- 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

$\delta(q_0, c, z) = (q_0, \varepsilon)$

$[q_0 \, ^z \, q_0] \rightarrow c$

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 $\delta(q_0, a, b) = (q_0, ab)$

$[q_0 \, ^b \, q_0] \rightarrow a \, [q_0 \, ^a \, q_0] \; [q_0 \, ^b \, q_0]$

$[q_0 \, ^b \, q_0] \rightarrow a \, [q_0 \, ^a q_1] \; [q_1 \, ^b \, q_0]$

$[q_0 \, ^b \, q_1] \rightarrow a \, [q_0 \, ^a \, q_0] \; [q_0 \, ^b \, q_1]$

$[q_0 \, ^b \, q_1] \rightarrow a \, [q_0 \, ^a q_1] \; [q_1 \, ^b \, q_1]$

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

$$[q_1\ {}^{a}\ q_1] \rightarrow a$$

Step 11 : Add productions for $\delta(q_1, b, b) = (q_1, \varepsilon)$

$$[q_1\ {}^{b}\ q_1] \rightarrow b$$

Step 12 : Add production for $\delta(q_1, \varepsilon, z) = (q_1, \varepsilon)$

$$[q_1\ {}^{z}\ q_1] \rightarrow \varepsilon$$

Step 13 : Add productions for $\delta(q_0, b, z) = (q_0, bz)$

$$[q_0\ {}^{z}\ q_0] \rightarrow b[q_0\ {}^{b}\ q_0]\ [q_0\ {}^{z}\ q_0]$$

$$[q_0\ {}^{z}\ q_0] \rightarrow b[q_0\ {}^{b}\ q_1]\ [q_1\ {}^{z}\ q_0]$$

$$[q_0\ {}^{z}\ q_1] \rightarrow b[q_0\ {}^{b}\ q_0]\ [q_0\ {}^{z}\ q_1]$$

$$[q_0\ {}^{z}\ q_1] \rightarrow b[q_0\ {}^{b}\ q_1]\ [q_1\ {}^{z}\ q_1]$$

Example 5.6.12

For the PDA $(\{q_0, q_1\}, (0, 1), \{z_0, x\}, \delta, q_0, z_0, \phi)$ where δ is

$\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$

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

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

$\delta(q_0, \varepsilon, z_0) = \{(q_0, \varepsilon)\}$

$\delta(q_1, 1, x) = \{(q_1, \varepsilon)\}$

$\delta(q_1, 0, z_0) = \{(q_0, z_0)\}$

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.	Productions due to start symbol S.	$S \rightarrow [q_0\ {}^{z_0}\ q_0]$
		$S \rightarrow [q_0\ {}^{z_0}\ q_1]$
2.	$\delta(q_0, 1, z_0) = (q_0, xz_0)$	$[q_0\ {}^{z_0}\ q_0] \rightarrow 1[q_0\ {}^{x}\ q_0][q_0\ {}^{z_0}\ q_0]$
		$[q_0\ {}^{z_0}\ q_0] \rightarrow 1[q_0\ {}^{x}\ q_1][q_1\ {}^{z_0}\ q_0]$
		$[q_0\ {}^{z_0}\ q_1] \rightarrow 1[q_0\ {}^{x}\ q_0][q_0\ {}^{z_0}\ q_1]$
		$[q_0\ {}^{z_0}\ q_1] \rightarrow 1[q_0\ {}^{x}\ q_1][q_1\ {}^{z_0}\ q_1]$
3.	$\delta(q_0, 1, x) = (q_0, xx)$	$[q_0\ {}^{x}\ q_0] \rightarrow 1[q_0\ {}^{x}\ q_0][q_0\ {}^{x}\ q_0]$
		$[q_0\ {}^{x}\ q_0] \rightarrow 1[q_0\ {}^{x}\ q_1][q_1\ {}^{x}\ q_0]$
		$[q_0\ {}^{x}\ q_1] \rightarrow 1[q_0\ {}^{x}\ q_0][q_0\ {}^{x}\ q_1]$
		$[q_0\ {}^{x}\ q_1] \rightarrow 1[q_0\ {}^{x}\ q_1][q_1\ {}^{x}\ q_1]$
4.	$\delta(q_0, 0, x) = (q_1, x)$	$[q_0\ {}^{x}\ q_0] \rightarrow 0[q_1\ {}^{x}\ q_0]$
		$[q_0\ {}^{x}\ q_1] \rightarrow 0[q_1\ {}^{x}\ q_1]$
5.	$\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$	$[q_0\ {}^{z_0}\ q_0] \rightarrow \varepsilon$
6.	$\delta(q_1, 1, x) = (q_1, \varepsilon)$	$[q_1\ {}^{x}\ q_1] \rightarrow 1$
7.	$\delta(q_1, 0, z_0) = (q_1, z_0)$	$[q_1\ {}^{z_0}\ q_0] \rightarrow 0[q_0\ {}^{z_0}\ q_0]$
		$[q_1\ {}^{z_0}\ q_1] \rightarrow 0[q_0\ {}^{z_0}\ q_1]$

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

$[q_0\ {}^{x}\ q_0]$ – 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 :

$S \rightarrow A \mid B$

$A \rightarrow 1EA \mid 1FC$

$B \rightarrow 1EB \mid 1FD$

$E \rightarrow 1EE \mid 1FG$

$F \rightarrow 1EF \mid 1FH$

$E \rightarrow 0G$

$F \rightarrow 0H$

$A \rightarrow \varepsilon$

$H \rightarrow 1$

$C \rightarrow 0A$

$D \rightarrow 0B$

1. Removing ε-production.

   Nullable set $= \{S, A\}$

ε-productions are removed with resulting set of productions as given below :

$S \rightarrow A \mid B$

$A \rightarrow 1EA \mid 1FC \mid 1E$

$B \rightarrow 1EB \mid 1\ FD$

$C \rightarrow 0A \mid 0$

$D \rightarrow 0B$

$E \rightarrow 1EE \mid 1\ FG \mid 0G$

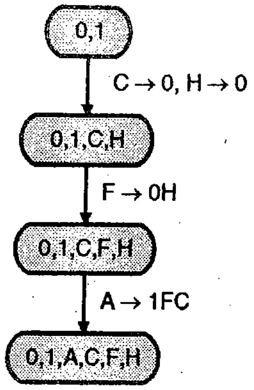
$F \rightarrow 1EH \mid 1FH \mid 0H$

$H \rightarrow 0$

2. Removing non-generating symbols

   Following symbols are non-generating
   = (B, D, E, G)

   Set of productions after elimination of non-generating symbols



   $S \to A$
   $A \to 1FC$
   $C \to 0A \mid 0$
   $F \to 1FH \mid 0H$
   $H \to 0$

3. Unit production $S \to A$ is removed, the resulting set of productions :

   $S \to 1FC$
   $A \to 1FC$
   $C \to 0A \mid 0$
   $F \to 1FH \mid 0H$
   $H \to 0$

   The language accepted by the PDA is given by :

   $$L = \{1^n 0 1^n 0 \ldots 1^n 0 \mid n \geq 1\} \cup \varepsilon$$

Example 5.6.13

For the PDA

1. $\delta(q_0, a, z_0) = (q_0, az_0)$
2. $\delta(q_0, a, a) = (q_0, aa)$
3. $\delta(q_0, b, a) = (q_1, a)$
4. $\delta(q_1, b, a) = (q_1, a)$
5. $\delta(q_1, a, a) = (q_2, \in)$
6. $\delta(q_2, a, a) = (q_2, \in)$
7. $\delta(q_2, \in, z_0) = (q_2, \in)$

Write CFG.

Solution :

1. Productions due to start symbol S.

$$S \to [q_0\, z_0\, q_0] \mid [q_0\, z_0\, q_1] \mid [q_0\, z_0\, q_2]$$

2. Productions due to $\delta(q_0, a, z_0) = (q_0, az_0)$

$$[q_0\, a\, q_0] \to a\, [q_0\, a\, q_0]\, [q_0\, z_0\, q_0]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, z_0\, q_0]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, z_0\, q_0]$$
$$[q_0\, a\, q_1] \to a\, [q_0\, a\, q_0]\, [q_0\, z_0\, q_1]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, z_0\, q_1]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, z_0\, q_1]$$
$$[q_0\, a\, q_2] \to a\, [q_0\, a\, q_0]\, [q_0\, z_0\, q_2]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, z_0\, q_2]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, z_0\, q_2]$$

3. Productions due to $\delta(q_0, a, a) = (q_0, aa)$

$$[q_0\, z_0\, q_0] \to a\, [q_0\, a\, q_0]\, [q_0\, a\, q_0]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, a\, q_0]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, a\, q_0]$$
$$[q_0\, z_0\, q_1] \to a\, [q_0\, a\, q_0]\, [q_0\, a\, q_1]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, a\, q_1]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, a\, q_1]$$
$$[q_0\, z_0\, q_2] \to a\, [q_0\, a\, q_0]\, [q_0\, a\, q_2]$$
$$\to a\, [q_0\, a\, q_1]\, [q_1\, a\, q_2]$$
$$\to a\, [q_0\, a\, q_2]\, [q_2\, a\, q_2]$$

4. Productions due to $\delta(q_0, b, a) = (q_1, a)$

$$[q_0\, a\, q_0] \to b\, [q_1\, a\, q_0]$$
$$[q_0\, a\, q_1] \to b\, [q_1\, a\, q_1]$$
$$[q_0\, a\, q_2] \to b\, [q_1\, a\, q_2]$$

5. Productions due to $\delta(q_0, a, a) = (q_2, \in)$

$$[q_1\, a\, q_2] \to a$$

6. Productions due to $\delta(q_2, a, a) = (q_2, \in)$

$$[q_2\, a\, q_2] \to a$$

7. Productions due to $\delta(q_2, \in, z_0) = (q_2, \in)$

$$[q_2\, z_0\, q_2] \to \in$$

Example 5.6.14

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,

   $\delta(q_i, x, b) \Rightarrow (q_j, c)$ [pop b and push c]

   can be replaced by couple of transitions.

1. $\delta(q_i, x, b) \Rightarrow \delta(q_{temp}, \varepsilon)$
2. $\delta(q_{temp}, \varepsilon, \varepsilon) \Rightarrow \delta(q_j, c)$

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 5.6.15

Describe the language L(M) in English for push down automation.

$M = (K, \Sigma, \Gamma, \Delta, S, F)$

where $k = \{S, F\}$ $F = \{F\}$

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

$\Delta = \{((S, a, \varepsilon), (S, a)), ((S, b, \varepsilon), (S, a)), ((S, a, \varepsilon), (F, \varepsilon)), ((F, a, a), (F, \varepsilon)), ((F, b, a), (F, \varepsilon))\}$

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.

$$((S, a, \varepsilon),(S,a)),((S,b,\varepsilon),(S,a))$$

- 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.

Example 5.6.16 SPPU - Dec. 14, 9 Marks

Construct PDA for the following CFG :

$S \rightarrow aAB$ $A \rightarrow bA \mid b$ $B \rightarrow aB \mid bA \mid a$

Is the resultant PDA deterministic (DPDA) or non-deterministic (NPDA) ? Justify your answer.

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

$\delta(q, \epsilon, S) = \{(q, aAB)\}$	For each production.
$\delta(q, \epsilon, A) = \{q, bA\}, (q, b)\}$	
$\delta(q, \epsilon, B) = \{(q, aB), (q, bA), (q, a)\}$	
$\delta(q, a, a) = (q, t)$	For each terminal symbol.
$\delta(q, b, b) = (q, \epsilon)$	

The resultant PDA is NPDA due to non-deterministic moves.

Syllabus Topic : Deterministic PDA

5.7 Deterministic Push Down Automata (DPDA) SPPU - Dec. 13

University Question

Q. Compare deterministic PDA with non deterministic PDA. **(Dec. 2013, 5 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 :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F),$$

where $\delta(q, a, x)$ has one move for any $q \in Q, X \in \Gamma$ and $a \in \Sigma$.

5.7.1 Regular Language and DPDA SPPU - May 12

University Question

Q. Describe in brief : Regular language and DPDA. **(May 2012, 5 Marks)**

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 5.7.1

Design a DPDA for a binary number divisible by 3.

Solution :

Step 1 : Construction of DFA.

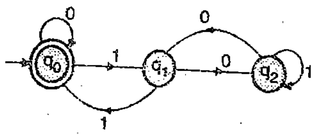


Fig. Ex. 5.7.1

Step 2 : The DPDA is given by :

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, 1, z_0\}, \delta, q_0, z_0, \{q_0\})$$

where the transition function δ is :

$$\delta(q_0, 0, z_0) = (q_0, z_0)$$
$$\delta(q_0, 1, z_0) = (q_1, z_0)$$
$$\delta(q_1, 0, z_0) = (q_2, z_0)$$
$$\delta(q_1, 1, z_0) = (q_0, z_0)$$
$$\delta(q_2, 0, z_0) = (q_1, z_0)$$
$$\delta(q_2, 1, z_0) = (q_2, z_0)$$

Example 5.7.2

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 5.7.3

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′.

$$M' = (Q, \Sigma, \Gamma, \delta, q_0, z_0, Q\text{-}F)$$

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.

$$(q_0, \omega, z_0) \overset{*}{\underset{M}{\vdash}} (p, -), \text{ where } p \in F.$$

Since the state of $p \notin Q\text{-}F$, 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.

$$(q_0, \omega, z_0) \overset{*}{\underset{M}{\vdash}} (r, -), \text{ where } r \notin F.$$

Since, the state $r \in Q\text{-}F$, it will be accepted by M′.

Example 5.7.4

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.

5.8 Application of PDA

SPPU - May 15, May 16, Dec. 16

University Question

Q. Give the application of PDA. **(May 2015, May 2016, Dec. 2016, 2 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 used for giving proofs of lemma on CFL.

Syllabus Topic
Parsing-Top-Down Parsing, Top-Down Parsing using Deterministic PDA, Bottom-Up Parsing

5.9 Parsing

We can always find a method for determining whether a particular string is generated by a CFG. Parsing a string is nothing but finding a derivation of the string in the given grammar G. A great deal of work has been done in finding an efficient algorithm for parsing. These algorithms depend on specific properties of the grammar.

PDA is a machine for CFG. A PDA can be used for parsing. A PDA can be enhanced to record its moves, so that the sequence of moves leading to an acceptance of the string can be remembered.

5.9.1 Top-Down Parsing :

A top down parser for a given grammar G tries to derive a string through a sequence of derivations starting with the start symbol.

A top down parser, normally uses leftmost derivation to derive an input string.

- **Recursive-descent parser** is a general top down parser.
- A recursive descent parser may require backtracking and repeated scan of input string. working of a recursive descent parser is being explained with the help of an example.

Example : Consider the grammar

$S \rightarrow aXb$

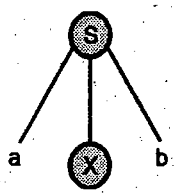
$X \rightarrow ab \mid b$

and the input string abb.

The parse tree of abb can be constructed as given below :

Step 1 : We create a parse tree with single node S. S is the start symbol.

Step 2 : We use the first production $S \rightarrow aXb$ to expand the node S



The leftmost leaf, labelled a, matches the first symbol of input string abb. Input pointer is advanced to the next symbol b of abb.

Step 3 : Now, X is expanded using $X \rightarrow ab$ and if it fails to generate remaining input symbols, we backtrack and try the next production $X \rightarrow b$.

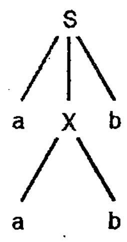


Fig. 5.9.1

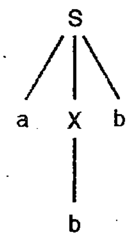


Fig. 5.9.2

The first tree will not generate the input string abb. The leftmost symbol 'a' of subtree, rooted at X does not match the next input symbol 'b'. We must backtrack and try the next production $X \rightarrow b$, as shown in Fig. 5.9.2.

- A recursive-descent parser may enter an infinite loop.
- We can write a predictive parser (without backtracking) by modifying the grammar :
  1. Left recursion should be eliminated.
  2. Apply left factoring to the grammar.

We can easily write a program for a recursive descent parser.

- There must be a function corresponding to each variable.
- A global variable 'position' points to the current input character.
- If the current input character matches the terminal in the production then the next input character is read by advancing the variable 'position' to the next location.

Example 5.9.1

Write a program for recursive descent parser for the following grammar.

$E = E + T \mid T$

$T = T * F \mid F$

$F = (E) \mid a \mid b$

Solution :

Step 1 : Removing left recursion from the grammar we get :

$$\begin{aligned} E &= TE_1 \\ E_1 &= +TE_1 \mid \varepsilon \\ T &= FT_1 \\ T_1 &= *FT_1 \mid \varepsilon \\ F &= (E) \mid a \mid b \end{aligned}$$

Step 2 : The program will have a function for each variable.

Function corresponding to $E = TE_1$

```
E( )
 { T( );
     E1( );
 }
```

Function corresponding to $E_1 = + T E_1 \mid \varepsilon$

```
E1( )
     {    if (input [position] == '+')
          {      match( );
                 T( );
                 E1( );
          }
     }
```

//The function match ( ) will increment the current position pointer for input string by 1.

Function corresponding to $T = F T_1$

```
T( )
{    F( );
     T1( );
}
```

Function corresponding to $T_1 = *F T_1 \mid \varepsilon$

```
T1( )
{    if(input [position] == '*')
          { match( );
            F( );
            T1( );
          }
}
```

Function corresponding to $F = (E) \mid a \mid b$

```
F( )
{
          if(input[position] == 'C')
          {      match( );
                 E ( );
                 if (input[position] == ')')
                      match( );
                 else
                      error =1;
          }
          else
           if(input [position] == 'a' ¦¦ input
          [position] == 'b')
                      match( );
}
```

```
// main program
char input [30];
int error = 0, position = 0;
void main( )
 {
      printf("\n enter a string :");
      gets(input);
      E( );
      if(error == 1 ¦¦ input [position]! = '\0')
           printf("\n invalid string");
      else
           printf("\n valid string");
 }
```

5.9.2 Bottom-Up Parsing :

In bottom up parsing, the source string is reduced to the start symbol of the grammar. The bottom up parsing is also known as shift-reduce parsing.

- A shift reduce parser constructs a parse tree by beginning at the leaves and then working up towards the root.
- A shift reduce parser can be constructed using a stack.

Implementation of shift reduce parser :

- A stack is taken to hold grammar symbols.
- An array is taken to store input string to be parsed.
- Initially, the stack is empty.

At each step the parser can take one of the following two steps :

1. Reduce : The process of reduction is shown in the Fig. 5.9.3.

   If there is a production $X \rightarrow \alpha$ and the string α is found at the top end of the stack then α should be replaced by X.

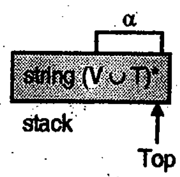


Fig. 5.9.3

2. If the reduction step cannot be carried out then the next input symbol is shifted on the stack.

The parser repeats this cycle of shift-reduce until it has detected an error or until the stack contains the start symbol and the input is empty.

Example 5.9.2

Consider the grammar

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

and the input string id + id * id. Show the working of the shift reduce parser.

Solution :

Sr. No.	Stack	Input	Action
1.	empty	id + id *id	Shift
2.	id	+ id * id	Reduce by $F \rightarrow id$
3.	F	+ id * id	Reduce by $T \rightarrow F$
4.	T	+ id * id	Reduce by $E \rightarrow T$
5.	E	+ id * id	Shift
6.	E +	id * id	Shift
7.	E + id	* id	Reduce by $F \rightarrow id$
8.	E + F	* id	Reduce by $T \rightarrow F$
9.	E + T	* id	Shift
10.	E + T*	id	Shift
11.	E + T * id	–	Reduce by $F \rightarrow id$
12.	E + T * F	–	Reduce by $T \rightarrow T * F$
13.	E + T	–	Reduce by
14.	E	–	Accept

❑❑❑

Unit IV

CHAPTER 6

Turing Machine (TM)

Syllabus

Turing Machine Model, Representation of Turing Machines, Design of TM, Description of TM, Techniques for TM Construction, Variants of Turing Machines, The Model of Linear Bounded Automata.

Syllabus Topic : Turing Machine Model

6.1 Introduction to Turing Machine

SPPU - May 13, May 16

University Questions

Q. Define Turing machine. **(May 2013, 2 Marks)**

Q. What is a Turing Machine ? **(May 2016, 3 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 :

FA ≤ DPDA ≤ NPDA ≤ Post machine

= Turing machine

Turing machine is capable of performing computations on inputs and producing a new result.

An abstract model of a turing machine is shown in Fig. 6.1.1.

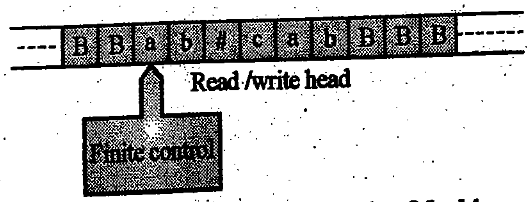


Fig. 6.1.1 : An example of a Turing Machine

- 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 6.1.1 SPPU - Dec. 12. 8 Marks

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. 6.1.1: 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. 6.1.1(c).

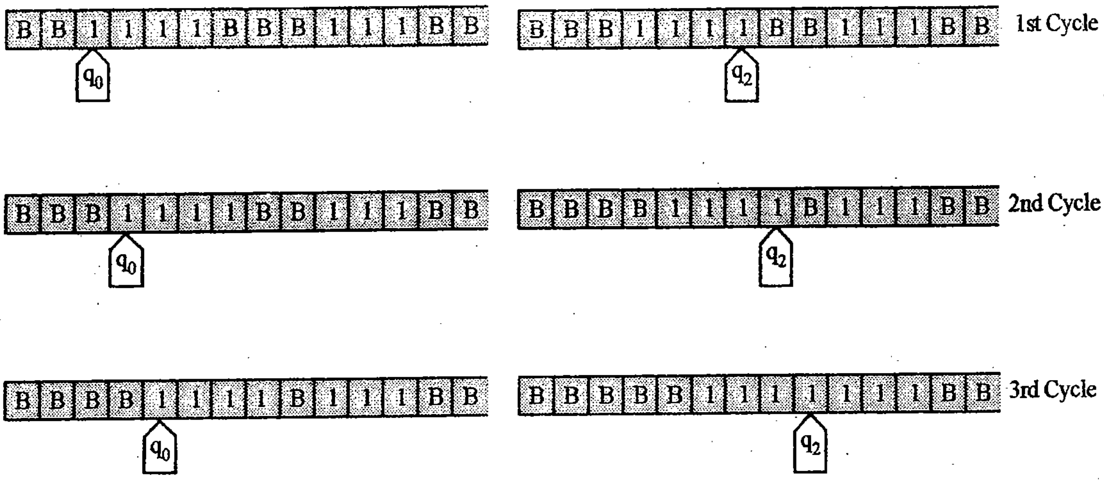


Fig. Ex. 6.1.1(c) : Cycles of computation

The transition behaviour of the turing machine is shown in Fig. Ex. 6.1.1(d).

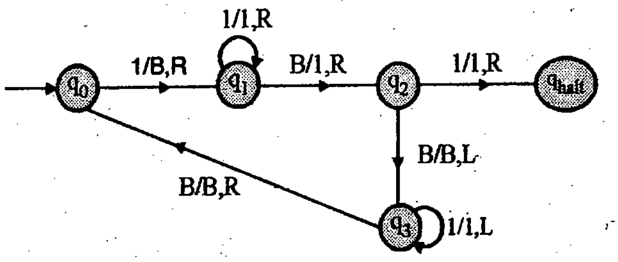
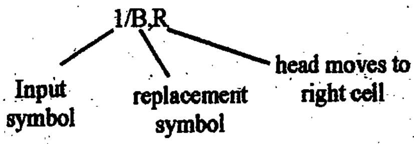


Fig. Ex. 6.1.1(d) : Transition diagram

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.



Meaning of various transitions and their effect on input tape

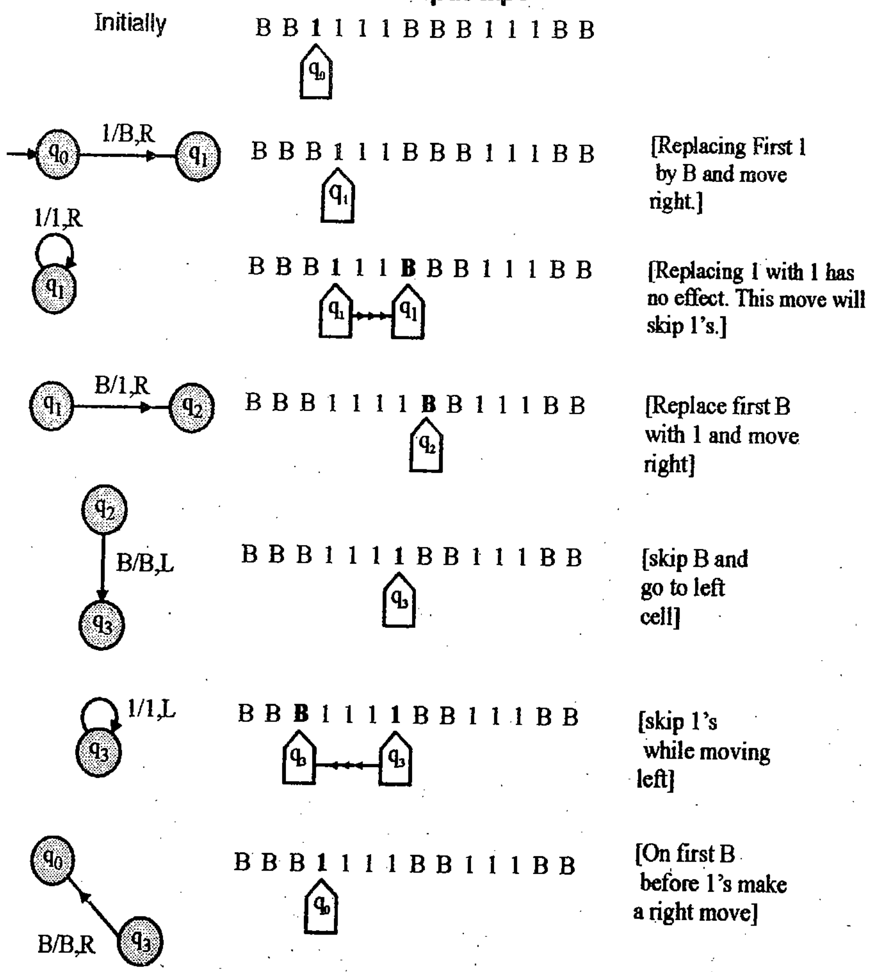


Fig. Ex. 6.1.1(e) : Transitions in detail for first cycle

Syllabus Topic : Representation of Turing Machines, Design of TM, Description of TM, Techniques for TM Construction

6.2 The Formal Definition of Turing Machine

SPPU - Dec. 12. May 16

University Questions

Q. Define Turing Machine. **(Dec. 2012, 2 Marks)**

Q. Give the formal definition of TM. **(May 2016, 3 Marks)**

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 $\rightarrow$ Move to left cell

R $\rightarrow$ Move to right cell

N $\rightarrow$ Remain in the current cell (No movement)

Example 6.2.1

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 :

$\delta(q_0,a) = (q_0,B,R)$ [erase 'a' and move right]

$\delta(q_0,b) = (q_0,B,R)$ [erase 'b' and move right]

$\delta(q_0,B) = (q_1,B,N)$ [stop in state q_1]

The transition function can also be given in a tabular form as shown in Fig. Ex. 6.2.1.

	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. 6.2.1 : Transition table

The transition function can also be given as transition diagram as shown in Fig. Ex. 6.2.1(a).

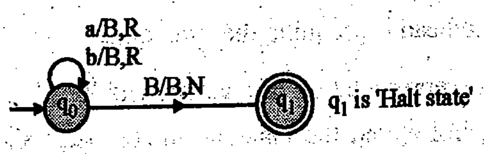


Fig. Ex. 6.2.1(a) : Transition diagram

6.2.1 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.

$$(q_0,\omega) \underset{M}{\overset{*}{\vdash}} \text{Halt-state}$$

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 6.2.2 SPPU - May 2013, 8 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^3b^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. 6.2.2(a).

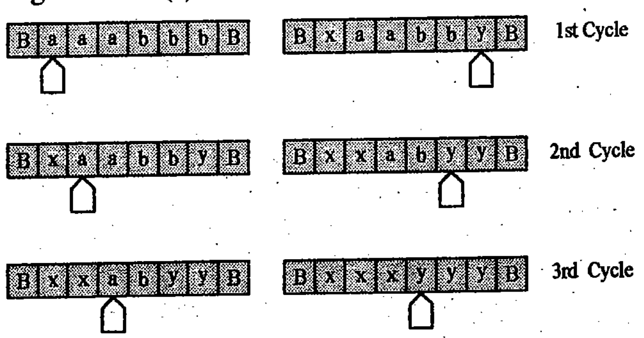


Fig. Ex. 6.2.2(a) : Cycles of computation for a^3b^3

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. 6.2.2(b) and Ex. 6.2.2(c).

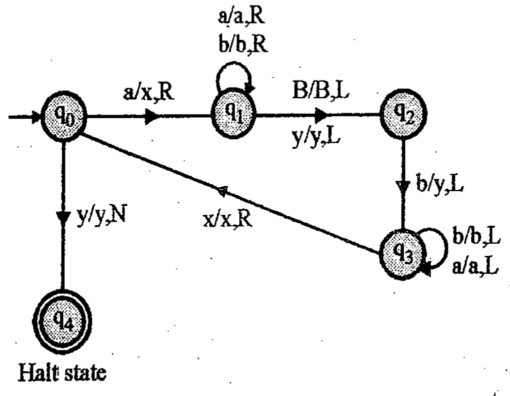


Fig. Ex. 6.2.2(b) : Transition diagram

	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.6.2.2(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 6.2.3

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 6.2.2. In Fig. Ex. 6.2.2(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 $a^i b^j$.

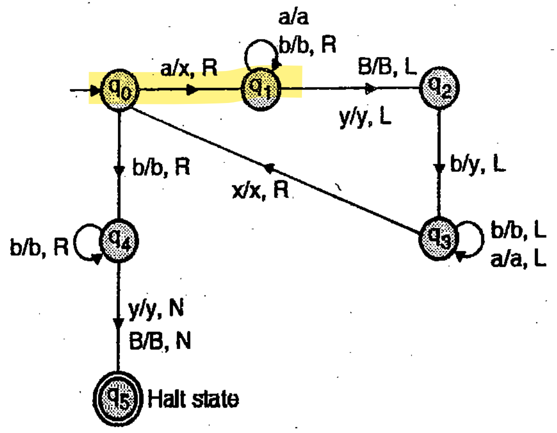


Fig. Ex. 6.2.3

6.2.2 Instantaneous Descriptions for Turing Machines SPPU - Dec. 12

University Question

Q. Define instantaneous description w.r.t. TM **(Dec. 2012, 2 Marks)**

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 $a_1 a_2 \ldots a_{i-1} \boxed{q\ a_i} a_{i+1} \ldots a_n$ 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. 6.2.1.

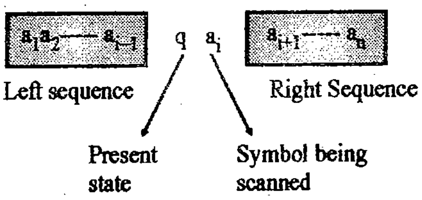


Fig. 6.2.1 : Instantaneous description of TM

Example 6.2.4

A turing machine is represented in Fig. Ex. 6.2.4 show the processing sequence for the input string aaabbb.

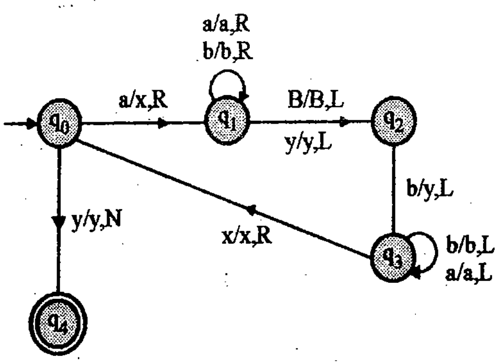


Fig. Ex. 6.2.4 : TM under consideration

Solution :

aaabbbB ⊢ xaabbbB ⊢ xaabbbB
(q_0, q_1, q_1)

⊢ xaabbbB ⊢ xaabbbB ⊢ xaabbbB
(q_1, q_1, q_1)

⊢ xaabbbB ⊢ xaabbbB ⊢ xaabbyB
(q_1, q_2, q_3)

⊢ xaabbyB ⊢ xaabbyB ⊢ xaabbyB
(q_3, q_3, q_3)

⊢ xaabb yB ⊢ xaabbyB
(q_3, q_0)

⊢ xxabb yB ⊢ xxabbyB ⊢ xxabbyB
(q_1, q_1, q_1)

⊢ xxabbyB ⊢ x xabbyB ⊢ xxabyyB
(q_1, q_2, q_3)

⊢ xxabyyB ⊢ xxabyyB ⊢ xxabyyB
(q_3, q_3, q_0)

⊢ xxxbyyB ⊢ xxxbyyB ⊢ xxxbyyB
(q_1, q_1, q_2)

⊢ xxxyyyB ⊢ xxxyyyB ⊢ xxxyyyB
(q_3, q_0, q_4)

[Accept]

Example 6.2.5

Let T be the Turing Machine defined by the 5-tuples,

(S_0, 0, 0, S_1, L)

(S_0, 1, 0, S_0, L)

(S_0, B, B, Halt, L)

(S_1, 0, 1, S_0, L)

(S_1, 1, 1, S_0, R)

For each of the following initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

1) 110B 2) 0011B 3) 0101B

Solution :

1) 110B

B110B ⊢ B010B [As per (S_0, 1, 0, S_0, L)]
(S_0, S_0)

⊢ BB010B [If there is less then 2 blanks on the left then the tape head halt with a crash]
(Halt)

2) 0011B

B0011B ⊢ B0011B [As per (S_0, 0, 0, S_1, L)]
(S_0, S_1)

The machine will fail at this point. There is no transition for input B in state S_1

3) 0101B

B0101B ⊢ B0011B [As per (S_0, 0, 0, S_1, L)]
(S_0, S_1)

This machine will fail at this point. There is no transition for input B in state S_1.

Example 6.2.6

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 is shown below :

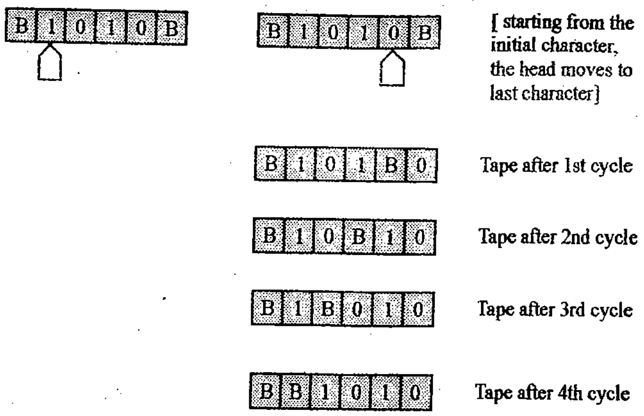


Fig. Ex. 6.2.6 : Cycle of computation

Turing machine is shown in Fig. Ex. 6.2.6(a) and 6.2.6(b).

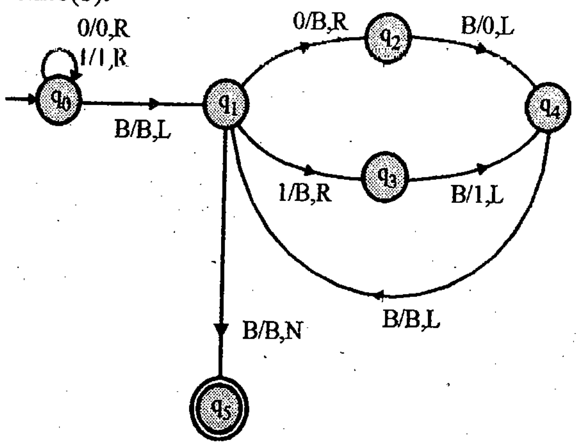


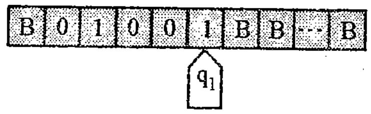
Fig. Ex. 6.2.6 (a) : Transition diagram

	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

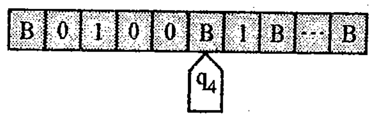
Fig. Ex. 6.2.6 (b) : Transition table

Meaning of various states

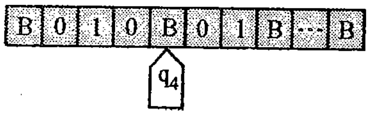
- Initial state q_0 is being used to skip the string of 0's and 1's so that the head can be positioned on last character.
- On seeing a B (blank), the head takes a left turn and positions itself on the last character. Machine enters state q_1.



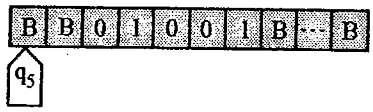
- A path through $q_1 \rightarrow q_3 \rightarrow q_4$ is for right shifting 1.



- A path through $q_1 \rightarrow q_2 \rightarrow q_4$ is for right shifting 0.

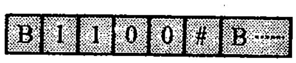


- Machine halts in state q_5.



Example 6.2.7 SPPU - May 14, 8 Marks

Design a TM to make a copy of a string over {0,1}



Input

Output

Solution :

- Two copies of the strings are separated by #.
- Machine is described in Fig. Ex. 6.2.7(a).

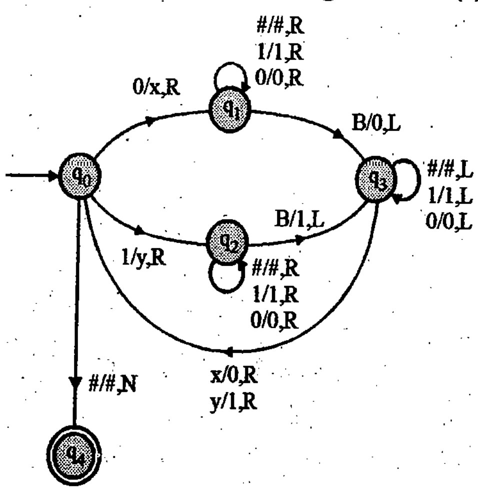


Fig. Ex. 6.2.7(a) : Transition diagram

	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. 6.2.7(b) : Transition table

– The turing machine M is given by :

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

where

$$Q = \{q_0,q_1,q_2,q_3,q_4\}$$
$$\Sigma = \{0,1\}$$
$$\Gamma = \{0,1,x,y,\#,B\}$$
$$q_0 = \text{initial state}$$
$$B = \text{blank symbol}$$
$$F = \{q_4\}$$

– Working of machine for an input 011 is being given below :

```
B011#B |- Bx11#B |- Bx11#B |- Bx11#B |- Bx11#B
 ↑          ↑           ↑          ↑          ↑
 q0         q1          q1         q1         q1
|- Bx11#0 |- Bx11#0 |- Bx11#0 |- Bx11#0
       ↑         ↑        ↑       ↑
       q3        q3       q3      q3
|- B011#0 |- B0y1#0 |- B0y1#0 |- B0y1#0B
     ↑           ↑          ↑          ↑
     q0          q2         q2         q2
|- B0y1#0B |- B0y1#01 |- B0y1#01
         ↑         ↑          ↑
         q2        q3         q3
|- B0y1#01 |- B0y1#01 |- B011#01 |- B01y#01B
      ↑         ↑          ↑              ↑
      q3        q3         q0             q2
|- B01y#01B |- B01y#01B |- B01y#01B
        ↑           ↑            ↑
        q2          q2           q2
|- B01y#011 |- B01y#011 |- B01y#011
         ↑          ↑          ↑
         q3         q3         q3
|- B01y#011 |- B011#011 |- B011#011
      ↑            ↑           ↑
      q3           q0          q4
```

Example 6.2.8 SPPU - May 12, May 14, 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.

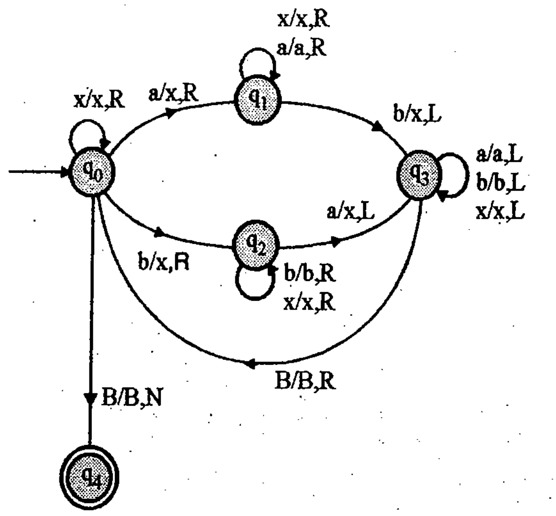


Fig. Ex. 6.2.8(a) : State transition diagram

	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	q_4 ← Halting state

Fig. Ex. 6.2.8(b) : Transition table

The turing machine M is given by :

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

Where, $Q = \{q_0,q_1,q_2,q_3,q_4\}$

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

Γ = {a,b,X,B}

q_0 = initial state

B = blank symbol

F = {q_4}

Working of machine for an input abba is being given below.

|– B **a** b b a B |– B x **b** b a B |– B **x** x b a B |– **B** x x b a B
q_0 q_1 q_3 q_3

|– B **x** x b a B |– B x **x** b a B |– B x x **b** a B |– B x x x **a** B
q_0 q_0 q_0 q_2

|– B x x **x** x B |– B x **x** x x B |– B **x** x x x B |– **B** x x x x B
q_3 q_3 q_3 q_3

|– B **x** x x x B |– B x **x** x x B |– B x x **x** x B |– B x x x **x** B
q_0 q_0 q_0 q_0

|– B x x x x **B** |– B x x x x **B**
q_0 q_4

Example 6.2.9 SPPU - Dec. 14. 9 Marks

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 below.

Input string is assumed to be (()))().

Cycle No.	Tape
Initial	B (())() B
1.	B (XX())() B
2.	B (XXXX)() B
3.	B XXXXXX() B
4.	B XXXXXXXX B

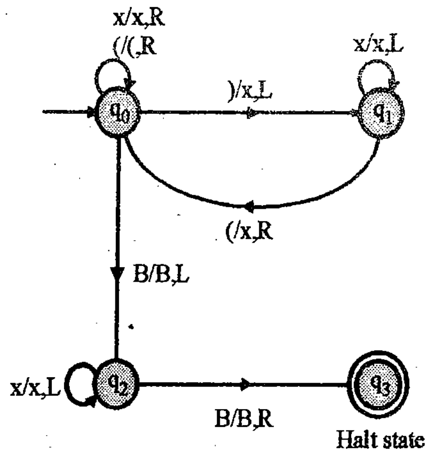


Fig Ex. 6.2.9(a) : State transition diagram

	(	)	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. 6.2.9(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}

Σ = {(,)}

Γ = {(,),x,B}

δ is given in Fig. Ex. 6.2.9(a) or 6.2.9(b)

q_0 = initial state

B = blank symbol

F = {q_3}, halting state

Making of the machine for input (()())() is given below :

B(()())()B ⊢ B(()())()B ⊢ B(()())()
q_0 q_0 q_0

⊢ B((x())()B ⊢ B(xx())()B ⊢ B(xx())()B
q_1 q_0 q_0

⊢ B(xx())()B ⊢ B(xx(x)()B ⊢ B(xxxx)()B
q_0 q_1 q_0

⊢ B(xxxx)()B ⊢ B(xxxxx()B ⊢ B(xxxxx()B
q_0 q_1 q_1

⊢ B(xxxxx()B ⊢ B(xxxxx()B ⊢ B(xxxxx()B
q_1 q_1 q_1

⊢ Bxxxxxx()B ⊢ Bxxxxxx()B ⊢ Bxxxxxx()B
q_0 q_0 q_0

⊢ Bxxxxxx()B ⊢ Bxxxxxx()B ⊢ Bxxxxxx()B
q_0 q_0 q_0

⊢ Bxxxxxx()B ⊢ Bxxxxxx(xB
q_0 q_1

⊢ BxxxxxxxxB ⊢ BxxxxxxxxB
q_0 q_0

⊢ BxxxxxxxxB ⊢ BxxxxxxxxB ⊢ BxxxxxxxxB
q_2 q_2 q_2

⊢ BxxxxxxxxB ⊢ BxxxxxxxxB ⊢ BxxxxxxxxB
q_2 q_2 q_2

⊢ BxxxxxxxxB ⊢ BxxxxxxxxB ⊢ BxxxxxxxxB
q_2 q_2 q_2

⊢ BxxxxxxxxB
q_3

Example 6.2.10

(a) **Design a TM that recognizes a string containing aba as a substring.**

(b) **Draw finite automata and corresponding turing machine for language L of following description. L = {x ← {a, b}* | x ends with aba}**

Solution :

(a) 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. 6.2.10(a).
- The equivalent TM is shown in Fig. Ex. 6.2.10(b).
- The head scans the input from left to right without modifying it.

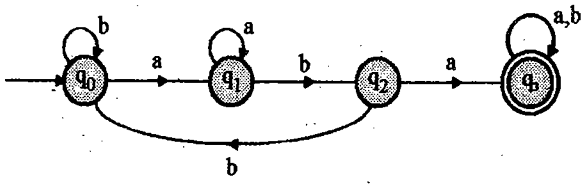


Fig. Ex. 6.2.10(a) : DFA for substring aba

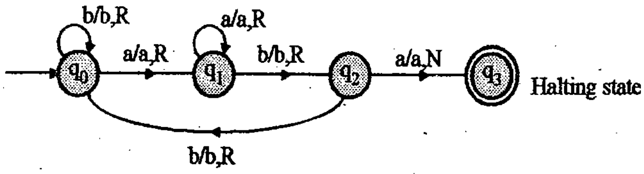


Fig. Ex. 6.2.10(b) : TM obtained from the DFA of Fig. Ex. 6.2.10(a)

The Turing machine M is given by :

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

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

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

$\Gamma = \{a,b,B\}$

δ is given in Fig. Ex. 6.2.10(b)

q_0 = initial state

B = blank symbol,

F = $\{q_0\}$, Halting state

(b)

Step 1 : DFA for the given language.

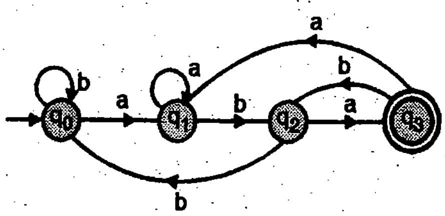


Fig. Ex. 6.2.10(c)

Step 2 : TM from DFA.

A TM can simulate the behaviour of DFA, through a left to right scan without modifying the input.

The equivalent TM is given below.

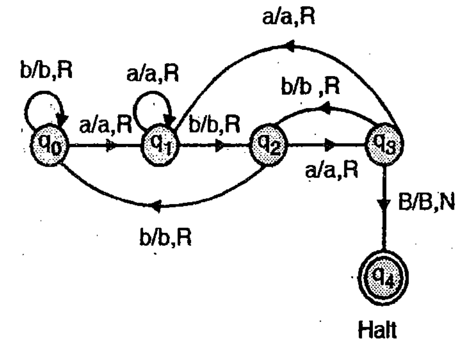


Fig. Ex. 6.2.10(d)

Example 6.2.11 SPPU - May 16, 3 Marks

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 $q_2 \to q_3 \to q_4 \to q_5$.

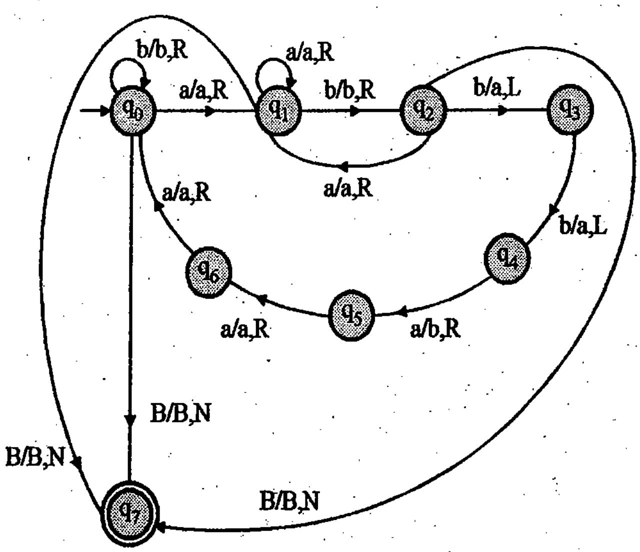


Fig. Ex. 6.2.11(a) : Transition diagram

- The head is positioned after the converted baa, by a transition through $q_5 \to q_6 \to q_0$.
- Above cycle is repeated to replace every occurrence of abb by baa.

	a	b	B
$\to q_0$	(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	q_7 ← Halting state

Fig. Ex. 6.2.11(b) : State transition table

The Turing machine M is given by

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

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

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

$\Gamma = \{a,b,B\}$

δ is given is Fig. Ex. 6.2.11(a) or 6.2.11(b)

q_0 = initial state

B = blank symbol

F = $\{q_7\}$, Halting state

Working of the machine for input bbabbaabbab is given below :

B**b**babbaabbabB ⊢ Bb**b**abbaabbabB ⊢ Bbb**a**bbaabbabB
q_0 q_0 q_0

⊢ Bbba**b**baabbabB ⊢ Bbbab**b**aabbabB ⊢ Bbba**b**aaabbabB
q_1 q_2 q_3

⊢ Bbb**a**aaaabbabB ⊢ Bbbb**a**aaabbabB
q_4 q_5

⊢ Bbbba**a**aabbabB ⊢ Bbbbaa**a**abbabB ⊢ Bbbbaaa**a**bbabB
q_6 q_0 q_1

⊢ Bbbbaaaa**b**babB ⊢ Bbbbaaaab**b**abB
q_1 q_2

⊢ Bbbbaaaa**b**aabB ⊢ Bbbbaaa**a**aaabB
q_3 q_4

⊢ Bbbbaaab**a**aabB ⊢ Bbbbaaaba**a**abB
q_5 q_6

⊢ Bbbbaaabaa**a**bB ⊢ Bbbbaaabaaa**b**B
q_0 q_1

⊢ Bbbbaaabaaab**B** ⊢ Bbbbaaabaaab**B**
q_2 q_7

Example 6.2.12

(a) Construct Turing Machine that recognizes the language :
$L = \{x \in \{0,1\}^* \mid x \text{ ends in } 00\}$

(b) Construct Turing machine that recognizes the language :
$L = \{0^n 1^m \mid n, m >= 0\}$

Solution : (a) 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.

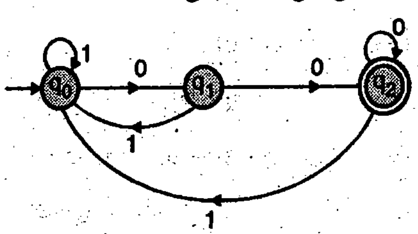


Fig. Ex. 6.2.12

Step 2 : TM from the DFA.

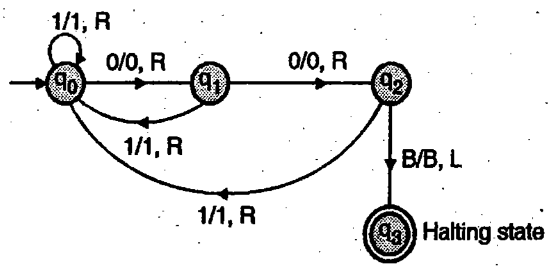


Fig. Ex. 6.2.12(a)

– 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, $Q = \{q_0, q_1, q_2, q_3\}$

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

Γ = {0,1,B}

δ = Transition function is given in step 2

q_0 = initial state

B = Blank symbol for the tape

F = $\{q_3\}$, halting state

(b) 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.

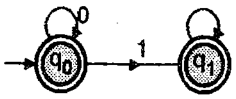


Fig. Ex. 6.2.12(b)

Step 2 : TM from the DFA

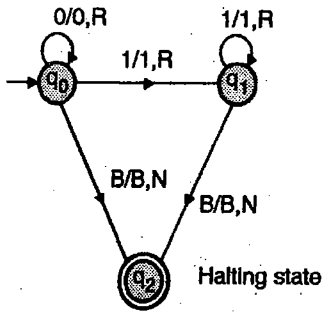


Fig. Ex. 6.2.12(c)

– An input symbol '0' in the state q_1 will imply rejection.

The Turing machine M is given by :

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

where,

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

Σ = {0, 1}

Γ = {0, 1, B}

δ = Transition function is given in step 2

q_0 = Initial state

B = Blank symbol for the tape

F = $\{q_2\}$, halting state.

Example 6.2.13

Design a turing machine to replace string 110 by 101 in binary input string.

Solution :

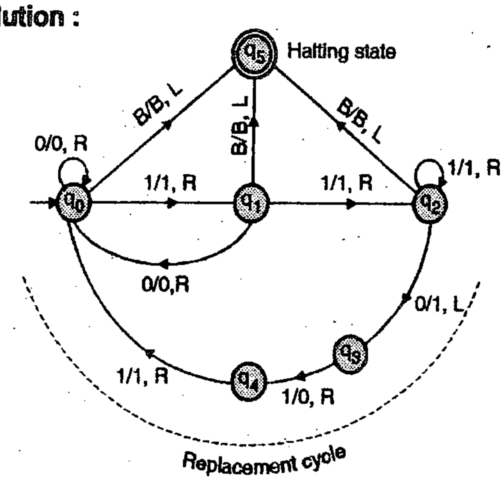


Fig. Ex. 6.2.13

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.

The turing machine M is given by :

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

Where, Q = $\{q_0,q_1,q_2,q_3,q_4,q_5\}$

Σ = {0,1}

Γ = {0,1,B}

δ = Transition function is shown using the transition diagram

B = Blank symbol for the tape

F = $\{q_5\}$, halting state

Working of the machine for input 0101101 is shown below :

0101101B ⊢ 0101101 ⊢ 0101101B ⊢ 0101101
↑ q_0 ↑ q_0 ↑ q_1 ↑ q_0

⊢ 0101101B ⊢ 0101101B
↑ q_1 ↑ q_2

⊢ 0101111B ⊢ 0101011B
↑ q_3 ↑ q_4

⊢ 010111B ⊢ 010111B ⊢ 010111B
↑ q_0 ↑ q_1 ↑ q_5 (halt)

Example 6.2.14

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.

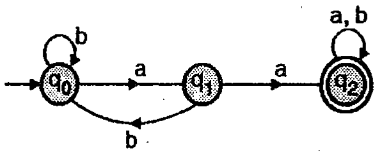


Fig. Ex. 6.2.14

Step 2 : TM from the DFA

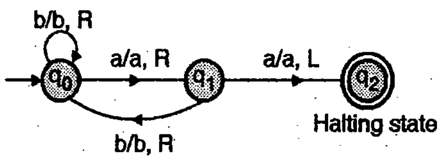


Fig. Ex. 6.2.14(a)

Example 6.2.15 SPPU - Dec. 13. 6 Marks

Design a TM which recognizes words of the form $a^n b^n c^n \mid n \geq 1$.

Solution :

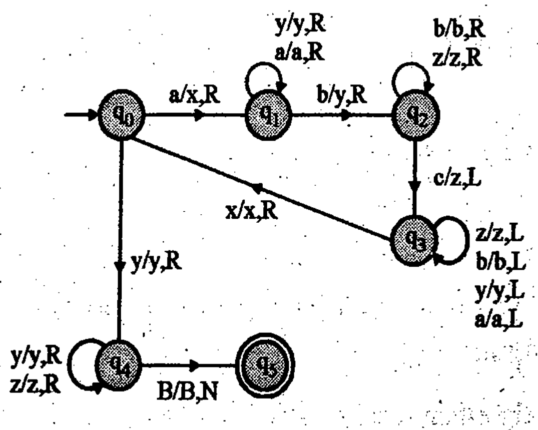


Fig. Ex. 6.2.15(a) : Transition diagram

	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	q_5	q_5	q_5	q_5	q_5	q_5

Hating state

Fig. Ex. 6.2.15(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 in Fig. Ex. 6.2.15(a) or 6. 2.15(b)

q_0 = is the initial state

B = is a 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 $a^3b^3c^3$ is given below :

B a a a b b b c c c B ⊢ B x a a b b b c c c B ⊢ B x a a b b b c c c B
q_0 q_1 q_1

⊢ B x a a b b b c c c B ⊢ B x a a y b b c c c B ⊢ B x a a y b b c c c B
q_1 q_2 q_2

⊢ B x a a y b b c c c B ⊢ B x a a y b b z c c B ⊢ B x a a y b b z c c B
q_2 q_3 q_3

⊢ B x a a y b b z c c B ⊢ B x a a y b b z c c B ⊢ B x a a y b b z c c B
q_3 q_3 q_3

⊢ B x a a y b b z c c B ⊢ B x a a y b b z c c B ⊢ B x x a y b b z c c B
q_3 q_0 q_1

⊢ B x x a y b b z c c B ⊢ B x x a y b b z c c B ⊢ B x x a y y b z c c B
q_1 q_1 q_2

⊢ B x x a y y b z c c B ⊢ B x x a y y b z c c B ⊢ B x x a y y b z z c B
q_2 q_2 q_3

⊢ B x x a y y b z z c B ⊢ B x x a y y b z z c B ⊢ B x x a y y b z z c B
q_3 q_3 q_3

⊢ B x x a y y b z z c B ⊢ B x x a y y b z z c B ⊢ B x x a y y b z z c B
q_3 q_3 q_0

⊢ B x x x y y b z z c B ⊢ B x x x y y b z z c B ⊢ B x x x y y b z z c B
q_1 q_1 q_1

⊢ B x x x y y y z z c B ⊢ B x x x y y y z z c B ⊢ B x x x y y y z z c B
q_2 q_2 q_2

⊢ B x x x y y y z z z B ⊢ B x x x y y y z z z B ⊢ B x x x y y y z z z B
q_3 q_3 q_3

⊢ B x x x y y y z z z B ⊢ B x x x y y y z z z B ⊢ B x x x y y y z z z B
q_3 q_3 q_3

Example 6.2.16 SPPU - Dec. 13, May 14, 6 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.

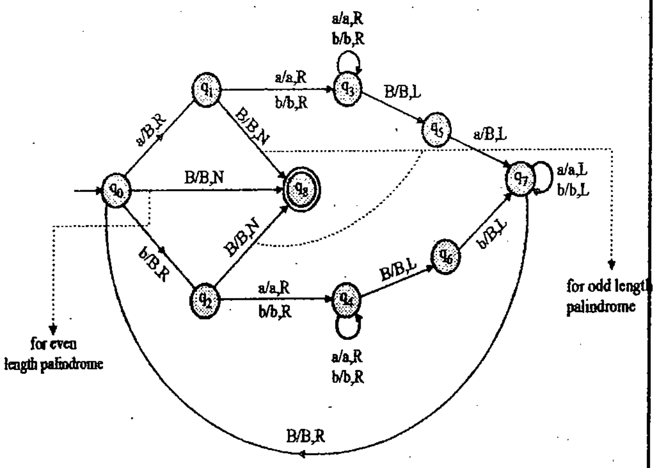


Fig. Ex. 6.2.16 : Transition diagram

- 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,

$$Q = \{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8\}$$

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

$$\Gamma = \{a,b,B\}$$

The transition function δ is given in Fig. Ex. 6.2.16.

q_0 = initial state

B = blank symbol

F = $\{q_8\}$, halting state

Working of TM for input abbabba is given below :

B**a**bbabbaB ⊢ BB**b**babbaB ⊢ BBb**b**abbaB ⊢ Bbb**a**bbaB
q_0 q_1 q_3 q_3

⊢ Bbba**b**baB ⊢ Bbbab**b**aB ⊢ Bbbabb**a**B ⊢ Bbbabba**B**
q_3 q_3 q_3 q_3

⊢ Bbbabb**a**B ⊢ Bbbab**b**BB ⊢ Bbba**b**bB ⊢ Bbb**a**bbB
q_5 q_7 q_7 q_7

⊢ Bb**b**abbB ⊢ B**b**babbB ⊢ **B**bbabbB ⊢ B**b**babbB
q_7 q_7 q_7 q_0

⊢ BB**b**abbB ⊢ Bb**a**bbB ⊢ Bba**b**bB ⊢ Bbab**b**B
q_2 q_4 q_4 q_4

⊢ Bbabb**B** ⊢ Bbab**b**B ⊢ Bba**b**BB ⊢ Bb**a**bB
q_4 q_6 q_7 q_7

⊢ B**b**abB ⊢ **B**babB ⊢ B**b**abB ⊢ BB**a**bB
q_7 q_7 q_0 q_2

⊢ Ba**b**B ⊢ Bab**B** ⊢ Ba**b**B ⊢ B**a**BB ⊢ **B**aB
q_4 q_4 q_6 q_7 q_7

⊢ B**a**B ⊢ BB**B** ⊢ BB**B**
q_0 q_1 q_8

Example 6.2.17

Draw a transition diagram for a Turing machine accepting the following language.

L = The language of all non-palindromes over {a,b}

Solution :

The solution follows from Example 6.2.16.

In Fig. Ex. 6.2.16, 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 :

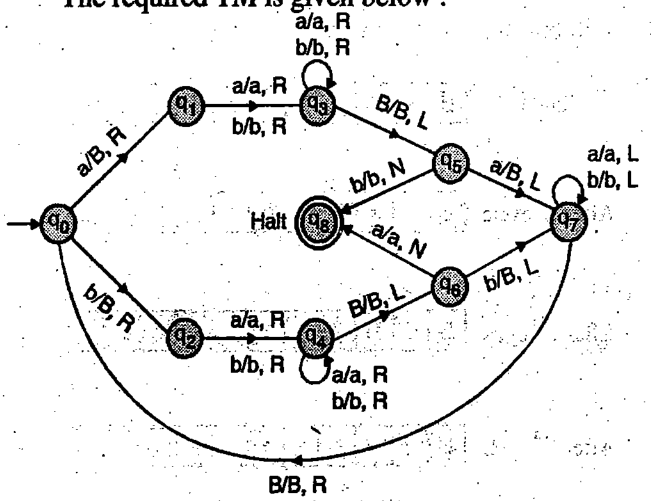


Fig. Ex. 6.2.17

Example 6.2.18

Design a TM which recognizes words of the form ωωω over alphabet {a,b}.

Solution :

Step 1 : Three ω's of ωωω 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	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. 6.2.18

1.2 Locate the last alphabet of ωωω 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 type for ω = abb is shown below.

Step 2 : Fist ω of ωωω 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

Fig. Ex. 6.2.18(a)

Step 3 : After step 2, the tape contains ωω, 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	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

Fig. Ex. 6.2.18(b)

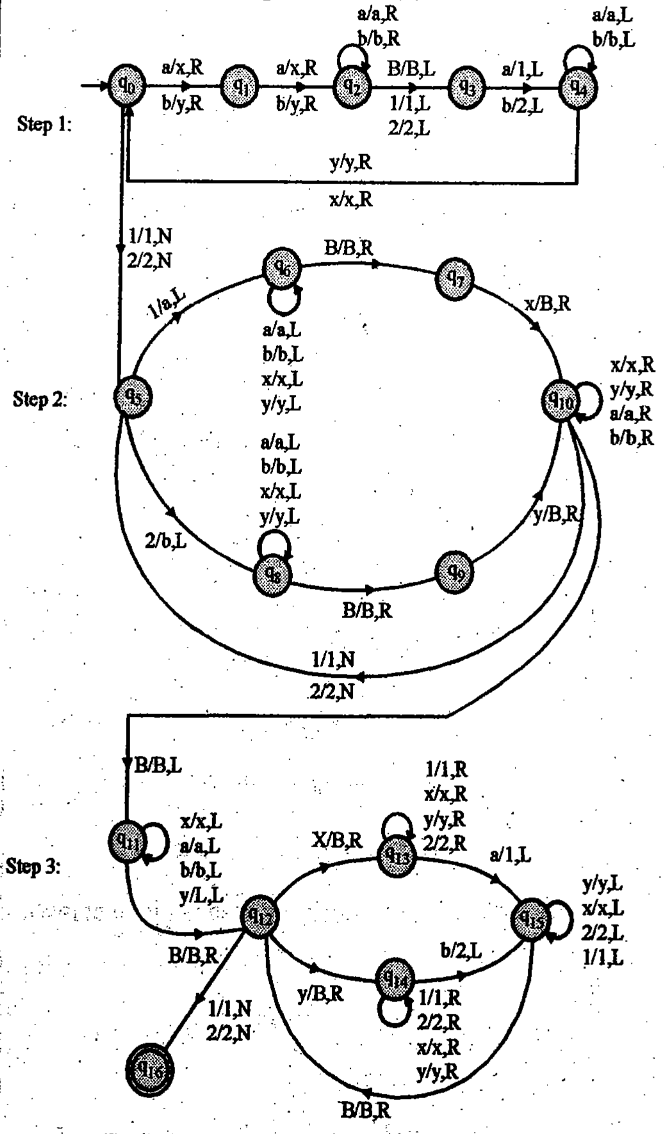


Fig. Ex. 6.2.18(c) : Transition diagram

Example 6.2.19 SPPU - Dec. 14. 9 Marks

Construct a turing machine for reversing a string.

Solution : The TM will require several cycles to reverse a string. The given string can be copied a in the reverse order to produce a string which is reverse of the original string.

Let us try to understand the design process with the help of the string 011.

Original string	B011B
After 1^{st} cycle	B01X1B
After 2^{nd} cycle	B0XX11B
After 3^{rd} cycle	BXXX110B

Subsequently, X's can be erased.

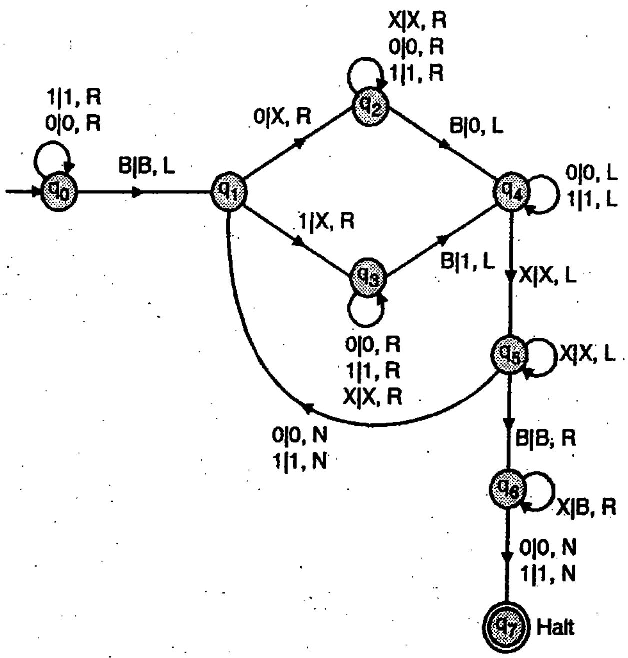


Fig. Ex. 6.2.19

- Initially, the head is positioned on the left most symbol.
- The head is advanced to the right most symbol in the state q_1.

Example 6.2.20 SPPU - Dec. 14. 9 Marks

Design TM that recognizes occurrence of substring '101' and replaces it with 110.

Solution : The Turing machine will look for every occurrence of the string 101.

- The state q_2 is for previous two symbols as 10.
- Next symbol as 1 in state q_2, will initiate the replacement process to replace 101 by 110.

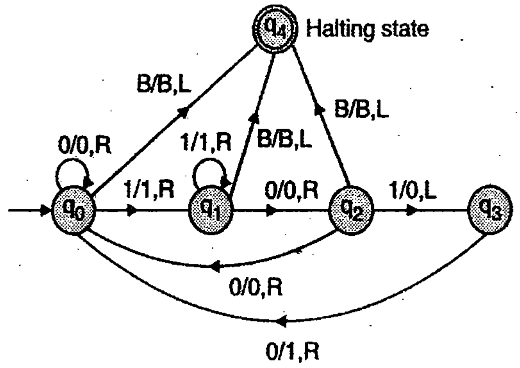


Fig. Ex. 6.2.20

- The Turing machine M is given by :

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

Where,
$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, B\}$
δ = transition function is shown using the transition diagram.
B = Blank symbol for the tape
F = $\{q_1\}$ halting state

Example 6.2.21 SPPU - Dec. 15. 6 Marks

Design a turing machine to perform right shift operation on a binary number.

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 is shown below :

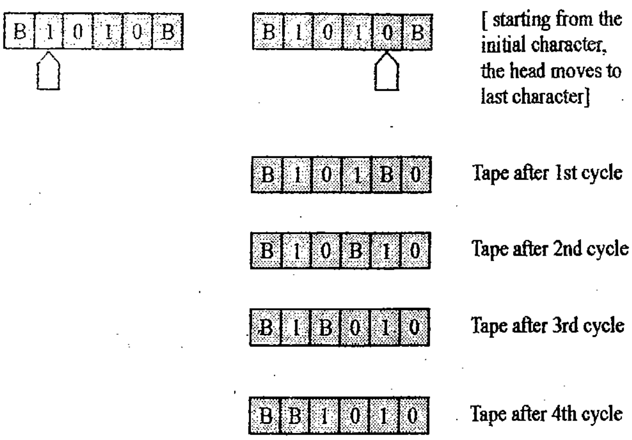


Fig. Ex. 6.2.21: Cycle of computation

Turing machine is shown in Fig. Ex. 6.2.21(a) and Fig. Ex. 6.2.21(b).

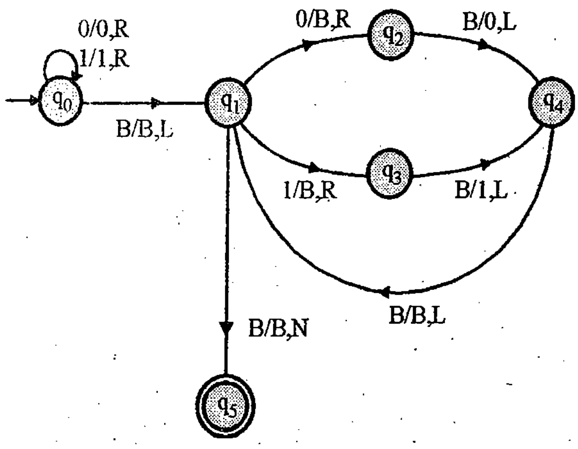


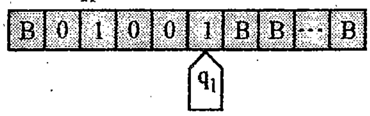
Fig. Ex. 6.2.21(a): Transition diagram

	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

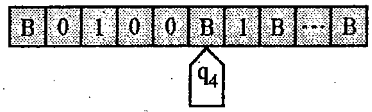
Fig. Ex. 6.2.21(b): Transition table

Meaning of various states

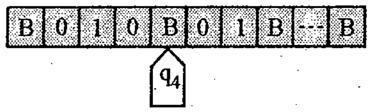
- Initial state q_0 is being used to skip the string of 0's and 1's so that the head can be positioned on last character.
- On seeing a B (blank), the head takes a left turn and positions itself on the last character. Machine enters state q_1.



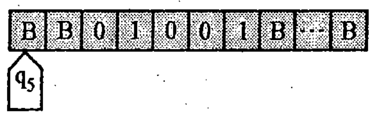
- A path through $q_1 \rightarrow q_3 \rightarrow q_4$ is for right shifting 1.



- A path through $q_1 \rightarrow q_2 \rightarrow q_4$ is for right shifting 0.



- Machine halts in state q_5.



6.3 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 :

$$(q_0, B\omega B) \overset{*}{\underset{M}{\vdash}} (q_f, B\mu B),$$

Where $\omega \in \Sigma_1^*$ and $\mu \in \Sigma_2^*$ satisfying

$$f(\omega) = \mu \quad \text{and} \quad q_f \in F$$

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 6.3.1 SPPU - May 13, 6 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 :

$$\underbrace{010110}_{\text{Complement every bit}}\ \underbrace{1000}_{\text{no change}} \Longrightarrow 1010011000$$

Algorithm

1. Locate the last bit (right most)
2. Move towards left till the first 1.
3. Complement remaining bits, while moving towards left.

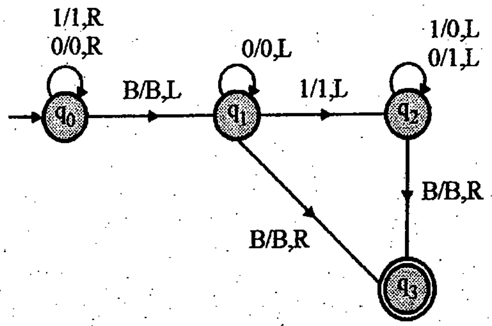


Fig. Ex. 6.3.1(a) : Transition diagram

	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	← Halting state

Fig. Ex. 6.3.1(b) : Transition table

The Turing machine M is givens by :

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

Where,

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

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

$$\Gamma = \{0,1,B\}$$

The transition function δ is given in Fig. Ex. 6.3.1(a) or Ex. 6.3.1(b),

q_0 = initial state

B = blank symbol

F = $\{q_3\}$, halting state

Working of TM for input 0101101000 is given below :

B**0**101101000B ⊢ B0**1**01101000B ⊢ B01**0**1101000B
q_0 q_0 q_0

⊢ B010**1**101000B ⊢ B0101**1**01000B ⊢ B01011**0**1000B
q_0 q_0 q_0

⊢ B010110**1**000B ⊢ B0101101**0**00B ⊢ B01011010**0**0B
q_0 q_0 q_0

⊢ B010110100**0**B ⊢ B0101101000**B** ⊢ B010110100**0**B
q_0 q_0 q_1

⊢ B01011010**0**0B ⊢ B0101101**0**00B ⊢ B010110**1**000B
q_1 q_1 q_1

⊢ B01011**0**1000B ⊢ B0101**1**11000B
q_2 q_2

⊢ B010**1**011000B ⊢ B01**0**0011000B ⊢ B0**1**10011000B
q_2 q_2 q_2

⊢ B**0**010011000B ⊢ **B**1010011000B ⊢ B**1**010011000B
q_2 q_2 q_3

Example 6.3.2 SPPU - May 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 systems, 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 :

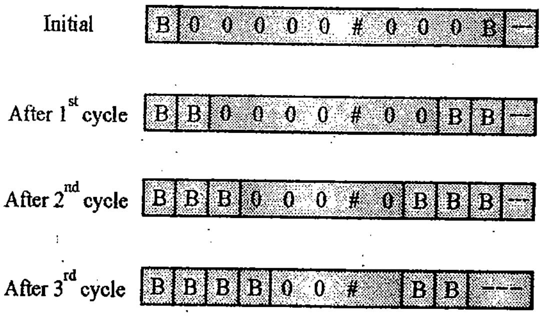


Fig. Ex. 6.3.2

Transition diagram and transition table are given in Fig. Ex. 6.3.2(a) and Ex. 6.3.2(b).

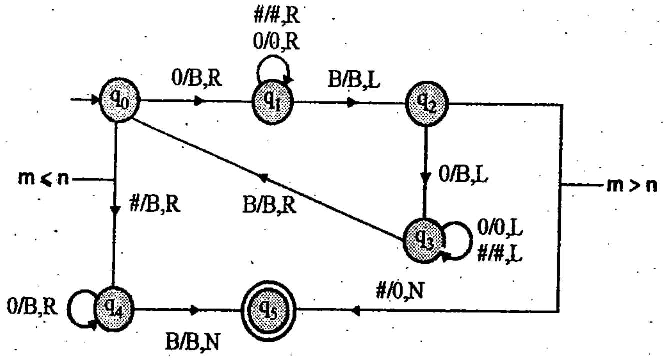


Fig. Ex. 6.3.2(a) : Transition diagram

	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 ← Halting state

Fig. Ex. 6.3.2(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. 6.3.2(a) or Ex. 6.3.2(b)

q_0 = initial state,

B = blank symbol

F = $\{q_5\}$, Halting state

The working of TM is being simulated for 5-3 :

```
B00000#000B ⊢ BB0000#000B ⊢ BB0000#000B
q0            q1            q1
⊢ BB 0000#000B ⊢ BB0000#000B ⊢ BB0000#000B
  q1             q1            q1
⊢ BB 0000#000B ⊢ BB0000#000B ⊢ BB0000#000B
  q1             q1            q1
⊢ BB 0000#000B ⊢ B0000#000B ⊢ B0000#00BB
  q1             q2           q3
⊢ B0000#00B ⊢ B0000#00B ⊢ B0000#00B ⊢ B0000#00B
  q3          q3          q3          q3
⊢ B0000#00B ⊢ B0000#00B ⊢ B0000#00B ⊢ B0000#00B
  q3          q3          q3          q0
⊢ BB000#00B ⊢ BB000#00B ⊢ BB000#00B ⊢ B000#00B
  q1          q1          q1          q1
⊢ B000#00B ⊢ B000#00B ⊢ B000#00B ⊢ B000#00B
  q1         q1         q1         q2
⊢ B000#0BB ⊢ B000#0B ⊢ B000#0B ⊢ B000#0B
  q3         q3        q3        q3
⊢ B000#0B ⊢ B000#0B ⊢ B000#0B ⊢ BB00#0B
  q3        q3        q0        q1
⊢ B00#0B ⊢ B00#0B ⊢ B00#0B ⊢ B00#0B ⊢ B00#0B
  q1       q1       q1       q1       q2
⊢ B00#BB ⊢ B00#B ⊢ B00#B ⊢ B00#B
  q3       q3      q3      q3
⊢ B00#B ⊢ BB0#B ⊢ BB0#B
  q0      q1      q1
⊢ B0#B ⊢ B0#B ⊢ B00B
  q1     q2     q5
```

Example 6.3.3

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 be require three cycles.

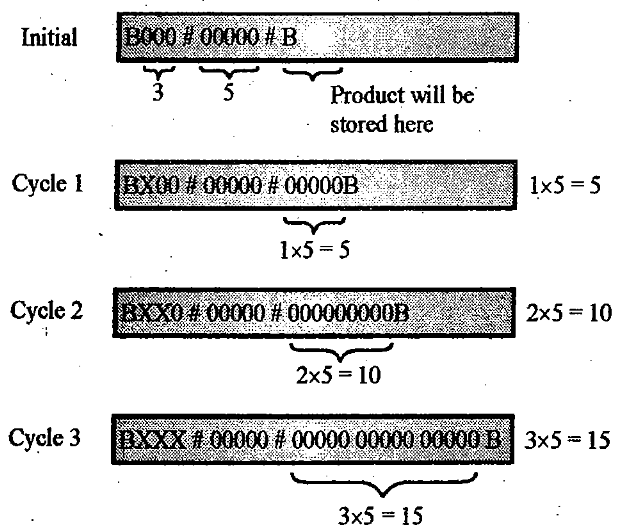


Fig. Ex. 6.3.3

- 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.

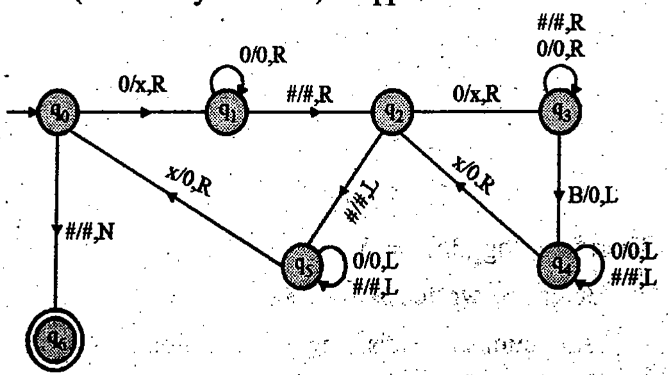


Fig. Ex. 6.3.3(a) : Transition diagram for TM of example 6.3.3

- 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 0's.

  # separates ω_1 and ω_2, ω_2 and ω_3.
- In the TM shown in Fig. Ex. 6.3.3, 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 given below :

B00#00#B $\vdash$ Bx0#00#B $\vdash$ Bx0#00#B $\vdash$ Bx0#00#B
$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_1$ $\uparrow q_2$

$\vdash$ Bx0#x0#B $\vdash$ Bx0#x0#B
$\uparrow q_3$ $\uparrow q_3$

$\vdash$ Bx0#x0#B $\vdash$ Bx0#x0#0B
$\uparrow q_3$ $\uparrow q_4$

$\vdash$ Bx0#x0#0B $\vdash$ Bx0#x0#0B
$\uparrow q_4$ $\uparrow q_4$

$\vdash$ Bx0#00#0B $\vdash$ Bx0#0x#0B
$\uparrow q_2$ $\uparrow q_3$

$\vdash$ Bx0#0x#0B $\vdash$ Bx0#0x#0B
$\uparrow q_3$ $\uparrow q_3$

$\vdash$ Bx0#0x#00B $\vdash$ Bx0#0x#00B
$\uparrow q_4$ $\uparrow q_4$

$\vdash$ Bx0#0x#00B $\vdash$ Bx0#00#00B
$\uparrow q_4$ $\uparrow q_2$

$\vdash$ Bx0#00#00B $\vdash$ Bx0#00#00B $\vdash$ Bx0#00#00B
$\uparrow q_5$ $\uparrow q_5$ $\uparrow q_5$

$\vdash$ Bx0#00#00B $\vdash$ Bx0#00#00B $\vdash$ B00#00#00B
$\uparrow q_5$ $\uparrow q_5$ $\uparrow q_0$

$\vdash$ B0x#00#00B $\vdash$ B0x#00#00B
$\uparrow q_1$ $\uparrow q_2$

$\vdash$ B0x#x0#00B $\vdash$ B0x#x0#00B

↑ q_3 ↑ q_3

$\vdash$ B0x#x0#00B $\vdash$ B0x#x0#00B

↑ q_3 ↑ q_3

$\vdash$ B0x#x0#00B $\vdash$ B0x#x0#000B

↑ q_3 ↑ q_4

$\vdash$ B0x#x0#000B $\vdash$ B0x#x0#000B

↑ q_4 ↑ q_4

$\vdash$ B0x#x0#000B $\vdash$ B0x#x0#000B

↑ q_4 ↑ q_4

$\vdash$ B0x#00#000B $\vdash$ B0x#0x#000B

↑ q_2 ↑ q_3

$\vdash$ B0x#0x#000B $\vdash$ B0x#0x#000B

↑ q_3 ↑ q_3

$\vdash$ B0x#0x#000B $\vdash$ B0x#0x#000B

↑ q_3 ↑ q_3

$\vdash$ B0x#0x#0000B $\vdash$ B0x#0x#0000B

↑ q_4 ↑ q_4

$\vdash$ B0x#0x#0000B $\vdash$ B0x#0x#0000B

↑ q_4 ↑ q_4

$\vdash$ B0x#0x#0000B $\vdash$ B0x#00#0000B

↑ q_4 ↑ q_4

$\vdash$ B0x#00#0000B $\vdash$ B0x#00#0000B $\vdash$ B0x#00#0000D

↑ q_5 ↑ q_5 ↑ q_5

$\vdash$ B0x#00#0000B $\vdash$ B00#00#0000B $\vdash$ B00#00#$\underbrace{0000}_{\text{result}}$B

↑ q_5 ↑ q_0 ↑ q_6

$\vdash$ B0x0000B $\vdash$ B0xy000B $\vdash$ B0xy000B

↑ q_1 ↑ q_2 ↑ q_2

$\vdash$ B0xy000B $\vdash$ Bxxy000B $\vdash$ Bxxy000B $\vdash$ Bxxy000B

↑ q_0 ↑ q_1 ↑ q_1 ↑ q_1

$\vdash$ Bxxyy00B $\vdash$ Bxxyy00B $\vdash$ Bxxyy00B

↑ q_2 ↑ q_2 ↑ q_2

$\vdash$ Bxxyy00B $\vdash$ Bxxyy00B $\vdash$ Bxyy00B $\vdash$ Byy00B

↑ q_2 ↑ q_5 ↑ q_5 ↑ q_5

x y

$\vdash$ B0y00B $\vdash$ B0000B $\vdash$ B0000B

↑ q_5 ↑ q_5 ↑ q_0

$\vdash$ B0x00B $\vdash$ B0xy0B $\vdash$ B0xy0B $\vdash$ B0xy0B

↑ q_1 ↑ q_2 ↑ q_2 ↑ q_0

$\vdash$ Bxxy0B $\vdash$ Bxxy0B $\vdash$ Bxxy0B $\vdash$ BxxyyB

↑ q_1 ↑ q_1 ↑ q_1 ↑ q_2

$\vdash$ BxxyyB $\vdash$ BxxyyB $\vdash$ BxxyyB $\vdash$ BxxyyB

↑ q_2 ↑ q_2 ↑ q_2 ↑ q_5

$\vdash$ BxyyB $\vdash$ ByyB $\vdash$ B0yB $\vdash$ B00B $\vdash$ B00B

↑ q_5 ↑ q_5 ↑ q_5 ↑ q_5 ↑ q_6

Result = 00, which is 2.

Example 6.3.8

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

$$n = 36$$

$$2^5 \leq 36 < 2^6$$

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.

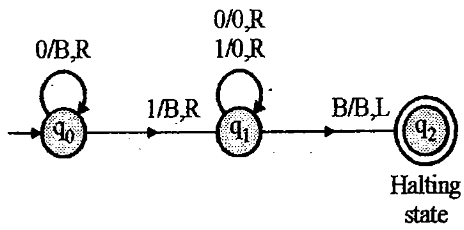


Fig. Ex. 6.3.8(a) : Transition diagram

	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	← Halting state

Fig. Ex. 6.3.8(b) : Transition table

Working of TM for $(36)_{10}$ is shown below :

$$(36)_{10} = (0100100)_2$$

$$\underset{\uparrow q_0}{B}0100100B \vdash B\underset{\uparrow q_0}{1}00100B \vdash B0\underset{\uparrow q_1}{0}100B$$

$$\vdash B00\underset{\uparrow q_1}{1}00B \vdash B000\underset{\uparrow q_1}{0}0B \vdash B0000\underset{\uparrow q_1}{0}0B$$

$$\vdash B00000\underset{\uparrow q_1}{0}B \vdash B000000\underset{\uparrow q_1}{B} \vdash B00000\underset{\uparrow q_2}{0}B$$

Example 6.3.9 SPPU - Dec. 12, 8 Marks

Design a TM to recognize the language

$\{a^n b^n c^m \mid n, m \geq 1\}$

Solution :

- a's are matched with b's
- After a's and b's are exhausted, the tape should contain some c's.
- In each cycle involving matching of a's and b's, 'a' is changed to 'x' and 'b' is changed to 'y'.

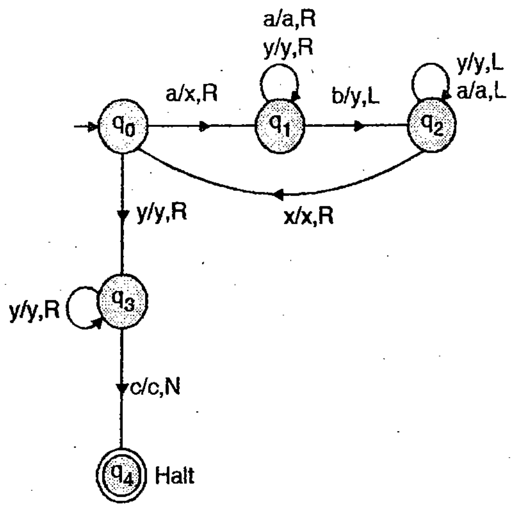


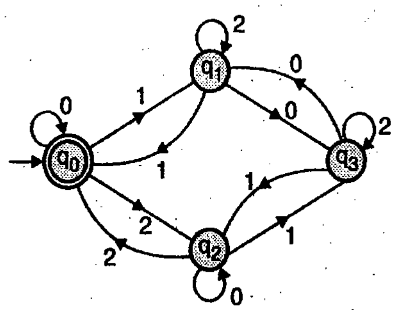
Fig. Ex. 6.3.9

Example 6.3.10 SPPU - May 15, 10 Marks

Design a Turing Machine to recognize an arbitrary string divisible by 4, given $\Sigma = \{0, 1, 2\}$

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 we have to design a DEA for a to ternary number divisible by y.



q_0 – reminder 0
q_1 – reminder 1
q_2 – reminder 2
q_3 – reminder 3

Fig. Ex. 6.3.10(a)

Step 2 : DFA to TM

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 = $\{q_4\}$, halting state

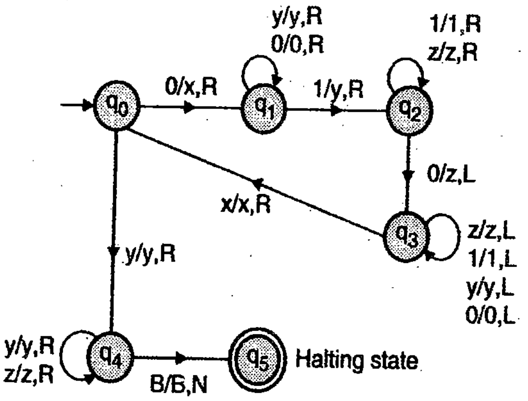


Fig. Ex. 6.3.10(b)

Example 6.3.11 SPPU - May 15, 8 Marks

Design a Turing Machine that accepts a language $L = \{0^n 1^n 0^n \mid n >= 1\}$

Solution :

Algorithm :

For a string $0^n 1^n 0^n$ the TM will need n cycles, in each cycle :

1. Leftmost '0' is written as x
2. Leftmost '1' is written as y
3. leftmost '0' in the 0's following is written as z.

At the end of n cycles, the tape should contain only x's, y's and z's.

The transition diagram is given in Fig. Ex. 6.3.11.

The turing machine M is given by :

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

Where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{0, 1, x, y, z, B\}$

δ = The transition is given above

q_0 = initial state

B = blank symbol

F = $\{q_5\}$, Halting state

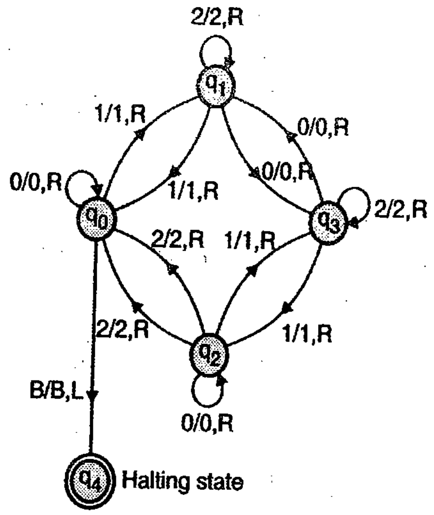


Fig. Ex. 6.3.11 : transition diagram

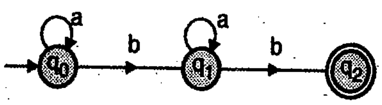
Example 6.3.12 SPPU - May 15, 6 Marks

Construct a TM that accepts a language L, a* ba*b.

Solution :

The language L is a regular language we can construct a DFA for L and the DFA so obtained can be converted into TM.

Step 1 : DFA for a*ba*b



Step 2 : DFA to TM

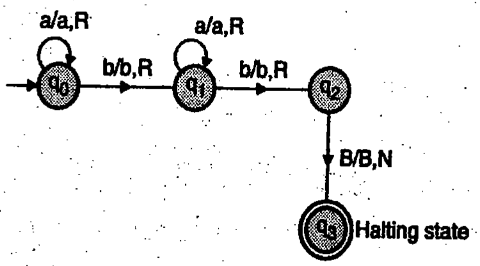


Fig. Ex.6.3.12 : Transition diagram

The turing machines for the above language is given by :

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

Where,

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

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

$\Gamma = \{a, b, B\}$

δ = the transition diagram is given above

q_0 = initial state

B = blank symbol

F = $\{q_3\}$ Halting state

Example 6.3.13 SPPU - Dec. 15. 5 Marks

Design turing machines for given two unary numbers, m and n, display, 'G', if m > n, 'E', if m = n, 'L', if m< n

Solution : It is assumed that the two numbers are separated by the symbol # and B is the blank symbol. The required TM is given in Fig. Ex. 6.3.13.

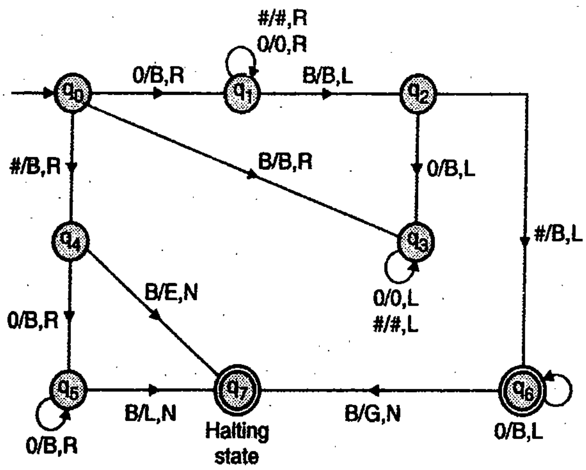


Fig. Ex. 6.3.13

Syllabus Topic : Variants of Turing Machines

6.4 Extension of Turing Machine :

SPPU - Dec. 13. May 16

University Questions

Q. Explain in detail concept of extensions to basic Turing Machine. (Dec. 2013, 3 Marks)

Q. What are the different ways for extension of TM ? Explain. (May 2016, 3 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.

6.4.1 Two-way Infinite Turing Machine

In a standard turing machine number of positions for leftmost blanks are 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.

6.4.2 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.

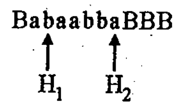


Fig. 6.4.1 : A Turing machine with two heads

The transition behavior of 2-head one tape Turing machine can be defined as given below :

δ (state, symbol under H_1, symbol under H_2)

= (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.

6.4.3 Multi-Tape Turing Machine

SPPU - May 12. Dec. 14

University Questions

Q. What is the concept of multi-tape Turing Machine ? (May 2012, 8 Marks)

Q. Write a note on multitape TM. (Dec. 2014, 2 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. 6.4.2.

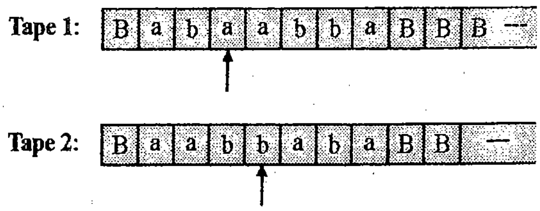


Fig. 6.4.2 : A two-tape turing machine

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 6.4.1

Construct a two-tape turing machine to recognize words of the form ωωω over alphabet {a,b}.

Solution : Let the initial string be placed on tape 1 and tape 2 may contain all blanks. Initial configuration for ω = abb is shown in Fig. Ex. 6.4.1(a).

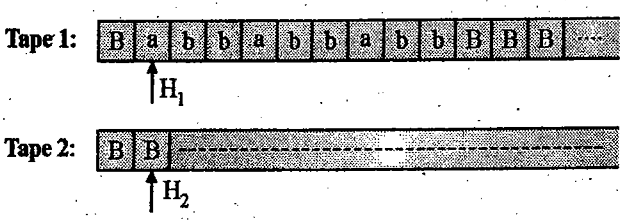


Fig. Ex. 6.4.1(a) : Initial configuration

Step 1: This step will copy the last ω from ωωω 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.

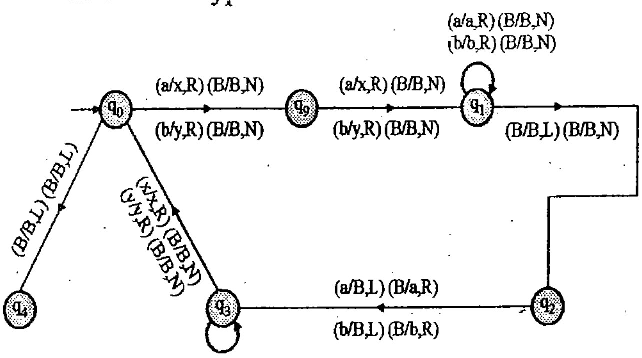


Fig. Ex. 6.4.1(b)

Contents of tapes after step 1 is shown in Fig. Ex. 6.4.1(c)

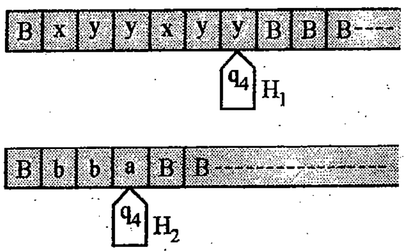


Fig. Ex. 6.4.1(c) : Last ω from tape 1 is copied as ω^R to tape 2

Step 2 : H_2 is moved to leftmost character on tape 2.

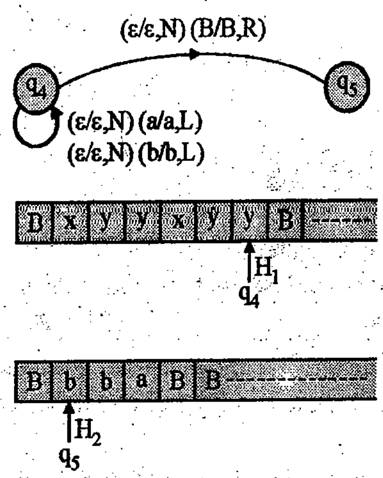


Fig. Ex. 6.4.1(d) : Situation after step 2

Step 3 : Two sets of xyy are matched with bba in the reverse order.

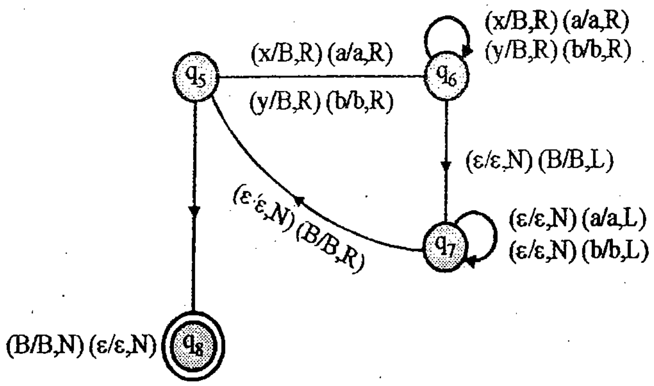


Fig. Ex. 6.4.1(e) : Two sets of xyy are matched with bba in the reverse order

---

Example 6.4.2 SPPU - May 16, 4 Marks

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.

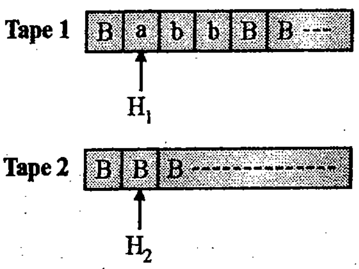
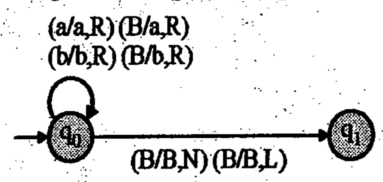


Fig. Ex. 6.4.2(a) : Initial configuration

Step 1 : String abb is copied to tape 2.



Contents of tapes after step1 are shown in Fig. Ex. 6.4.2(b).

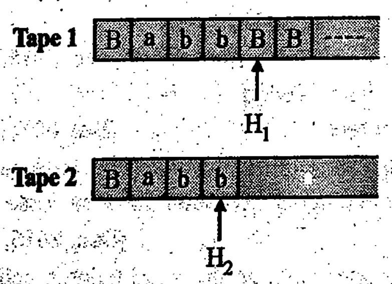
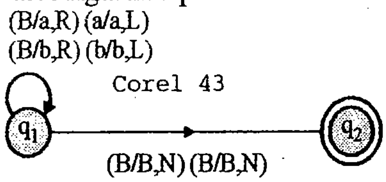


Fig. Ex. 6.4.2(b) : Contents of tapes after step 1

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. 6.4.2(c)

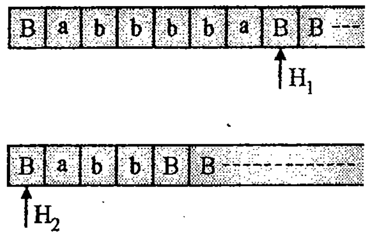


Fig. Ex. 6.4.2(c) : Contents of tapes after step 2

Transition diagram of the turing machine is shown in Fig. Ex. 6.4.2(d)

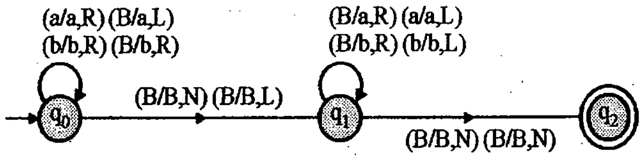


Fig. Ex. 6.4.2(d) : Transition diagram for example 6.4.2

6.4.4 Non-Deterministic Turing Machine

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 .

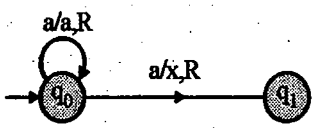


Fig. 6.4.3 : A sample move for NDTM

The transition behaviour for state q_0 for TM of Fig. 6.4.3 can be written as

$$\delta(q_0,a) = \{(q_0,a,R)\ (q_1,x,R)\}$$

Example 6.4.3

Construct NDTM to recognize words of the form ωω over alphabet {a,b}

Solution : First character of the second ω in ωω 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 ωω. Contents of tape after step 1 is shown below.

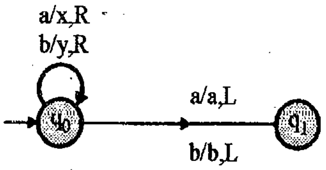


Fig. Ex. 6.4.3

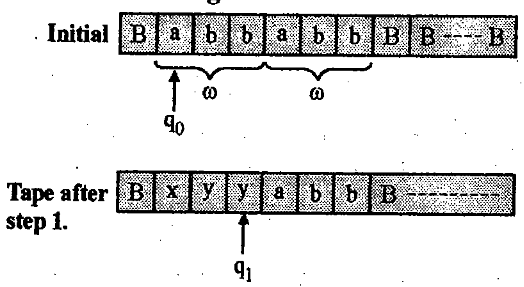


Fig. Ex. 6.4.3(a)

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.

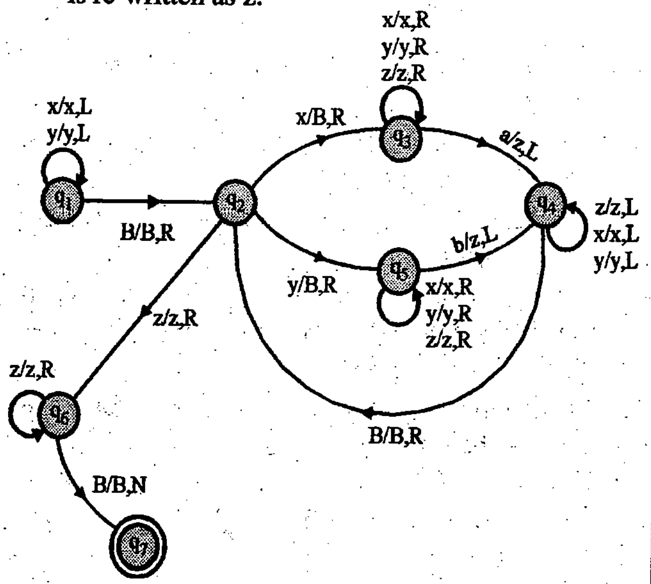


Fig. Ex. 6.4.3(b)

- 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 ∈ 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 ωω has been matched.
- q_7 is the final state.
- The final TM is shown in Fig. Ex. 6.4.3(c).

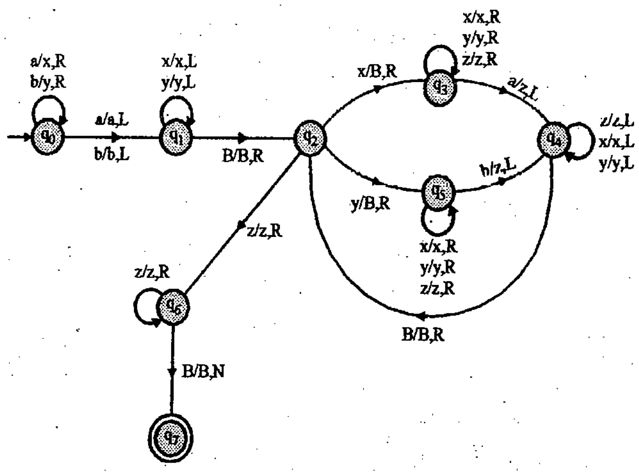


Fig. Ex. 6.4.3(c) : NDTM for ωω

6.5 Universal Turing Machine :

SPPU - May 12. Dec. 12. May 13. Dec. 13. Dec. 14. May 15. Dec. 15. May 16. Dec. 16

University Questions

Q. Describe Universal Turing machine in brief. **(May 2012, 8 Marks)**

Q. Define Universal TM. **(Dec. 2012, 2 Marks)**

Q. Explain in detail concept of Universal Turing Machine. **(Dec. 2013, 6 Marks)**

Q. Write a notes on Universal TM. **(Dec. 2014, Dec. 2016, 3 Marks)**

Q. How can Turing Machines be compared to computers ? **(May 2015, 6 Marks)**

Q. Justify how a turing machine can simulate a general purpose computer and vice-versa. **(Dec. 2015, 8 Marks)**

Q. Write short note on universal turing machine along with example. **(May 2013, May 2016, 6 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 complier.
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

$\{(q_i,a_i,q_j,a_j,m_f) : q_i,q_j,\in Q ; a_i,a_j,\in \Gamma ;$
$m_f \in$ {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.

Syllabus Topic : The Model of Linear Bounded Automata

6.6 Linear Bounded Automata

A linear bounded automata(LBA) restricts the length of the tape. The LBA is important as the set of context-sensitive languages is accepted by it.

A linear bounded automata is same as a non-deterministic Turing machine except in the following :

1. There is a special symbol < (say), marking the left end of the tape.
2. There is a special symbol > (say), marking the right end of the tape.
3. The head is not allowed to move beyond these end-markers.
4. The head is not allowed to change these end-markers.

An LBA can be denoted by :

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

Where < and > are symbols in Σ, the left and right end-markers. Q, Σ, Γ, δ, q_0 and F are same as for a non-deterministic TM.

□□□

CHAPTER 7 Unit VI

Undecidability and Intractable Problems

Syllabus

A Language that is not recursively enumerable TM & Type 0 grammars, An un-decidable problem that is RE, Language Acceptability by Turing Machines , TM's Halting Problem, Post Correspondence Problem, The Classes P and NP : Problems Solvable in Polynomial Time, An Example: Kruskal's Algorithm, Nondeterministic Polynomial Time, An NP Example: The Traveling Salesman Problem, Polynomial-Time Reductions NP Complete Problems, An NP-Complete Problem: The Satisfiability Problem, Tractable and Intractable Representing Satisfiability, Instances, NP Completeness of the SAT Problem, A Restricted Satisfiability Problem: Normal Forms for Boolean Expressions, Converting Expressions to CNF, The Problem of Independent Sets, The Node-Cover Problem.

Syllabus Topic : A Language that is not Recursively Enumerable, TM & Type 0 Grammars

7.1 Recursively Enumerable and Recursive Language

SPPU - May 12, Dec. 13, May 14

University Questions

Q. Define recursive and recursively enumerable languages and specify relationship between them. **(May 2012, 8 Marks)**

Q. Define recursive and recursively enumerable language along with example. **(Dec. 2013, 4 Marks)**

Q. Define and explain Recursive and Recursively Enumerable Languages. **(May 2014, 8 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.

Syllabus Topic : Language Acceptability by Turing Machines

7.1.1 Turing Acceptable Language

SPPU - Dec. 16

University Question

Q. Write a short note on language accepted/decided by TM. **(Dec. 2016)**

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 7.1.1 SPPU - May 12, May 13, Dec. 13, May 14, Dec. 15, Dec. 16, 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. 7.1.1(a).

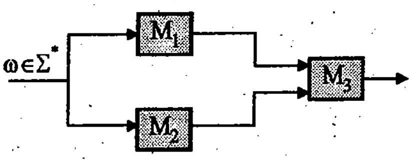


Fig. Ex. 7.1.1(a) : A turing machine for $L_1 \cup L_2$

- 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. 7.1.1(b).

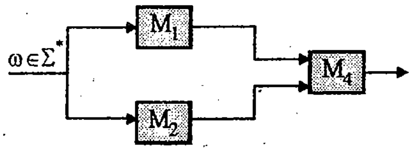


Fig. Ex. 7.1.1.(b) : A turing machine for $L_1 \cap L_2$

- 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 decides L_1.

Let us construct a turing machine M_5 as shown in Fig. Ex. 7.1.1(c).

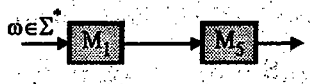


Fig. Ex. 7.1.1(c) : A turing machine for L_1'

- 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 7.1.2

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 right most b and H_3 is positioned on the right most 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 7.1.3

If L_1 and L_2 are two recursive languages and if L is defined as : $L = \{\omega \mid \omega$ is in L_1 and not in L_2, or ω is in L_2 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. 7.1.3.

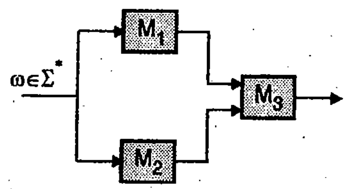


Fig. Ex. 7.1.3

- 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 output of both M_1 and M_2 is 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 7.1.4 SPPU - Dec. 12. 8 Marks

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. 7.1.4.

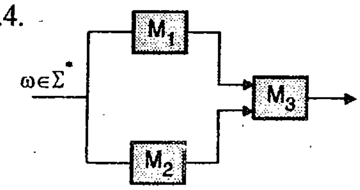


Fig. Ex. 7.1.4

- 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 $\omega \in L$ 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.

Example 7.1.5

Prove the theorem – "If L is accepted by a nondeterministic TM say T, and every possible sequence of moves of T causes it to halt, then L is recursive".

Solution :

Proof : Non-determinism means that at some stage, there may be a multiple moves. Thus, non-determinism can be handled through parallel computation. An NDTM can be simulated through n deterministic TMs.

Let us say that the given NDTM is equivalently represented by the following TMs running in Fig. Ex. 7.1.5.

$T_1, T_2, \ldots T_n$

Let us make the following construction :

Let the TM T is activated after every matching T_1 to T_3 has generated its output ("Y" or "N"). These outputs are written on the tape of T.

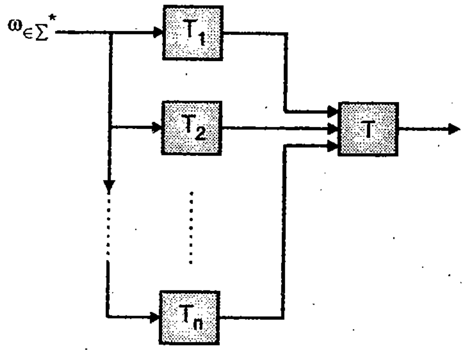


Fig. Ex. 7.1.5

Since, every sequence of move causes the TM to stop. Every TM from T_1 to T_n will ultimately half with output "Y" or "N".

Now, the TM T will scan its tape and generate an output "Yes", if it finds "Y" everywhere on its tape. Otherwise the TM T will generate an output "No".

Thus the above construction is clearly Turing decidable.

Example 7.1.6

Prove the theorem – "If L_1 and L_2 are recursively enumerable languages over Σ then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable".

Solution :

Let L_1 is Turing acceptable by the TM M_1 and L_2 is Turing acceptable by the TM M_2.

Let us make the following construction.

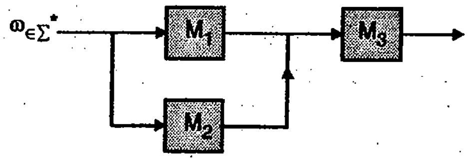


Fig. Ex. 7.1.6

- If, at any stage, there is an output from either M_1 or M_2, then on the first output from M_1 or M_2, the M_3 is activated.

1. Proving that $L_1 \cup L_2$ is recursively enumerable.

Case 1 : $\omega \in L_1$ or $\omega \in L_2$. If $\omega \in L_1$ or $\omega \in L_2$, the corresponding machine will obviously generate an answer "Y". On seeing "Y", the M3 will halt with answer yes.

Case 2 : $\epsilon \notin L_1$ and $\omega \notin L_2$.

In this case the machine M_1 and M_2 may loop forever and hence the TM M_3 will not be activated.

Thus the union $L_1 \cup L_2$ is Turing acceptable.

2. **Proving that $L_1 \cap L_2$ is recursively enumerable.**

   The TM M_3 waits for generation of outputs from both M_1 and M_2. If it finds "Y" form M_1 and "Y" from M_2 then it halts with answer yes.

   If $\omega \notin L_1 \cap L_2$ then M_3 can loop forever. Therefore, $L_1 \cap L_2$ is Turing acceptable.

Example 7.1.7 SPPU - Dec. 12, 8 Marks

Prove that every recursive language is recursively enumerable.

Solution : Every recursive language is Turing decidable. A language $L \subseteq \Sigma^*$ representing a problem over Σ is said to be Turing Decidable, if there is a TM, M which always halts given any input $w \in \Sigma^*$, whether $w \in L$ or $w \notin L$.

Every recursively enumerable language is Turing Acceptable. A language $L \subseteq \Sigma^*$ is said to be Turing acceptable language if there is a Turing Machine M which when given an input $w \in \Sigma^*$, such that $w \in L$, then halts with an output y. However, if $w \notin L$, then M may not halt.

Any recursive language is also recursively enumerable. Let us assume that the Turing machine M decides a language L. We can always construct a Truing machine M_1 from M that accepts the language L. All that we have to do is to make the rejecting states of M as non-halting state, from which the machine is guaranteed to never halt.

- Thus for any string $w \in L$, both M and M_1 will halt.
- Thus for any string $w \notin L$, M will halt but M_1 will loop forever.

Syllabus Topic : An Un-decidable Problem that is RE

7.1.2 An Un-decidable Problem that is RE (Recursively Enumerable)

In particular, recursive (decidable) languages are subset of the recursively enumerable languages. So, anything that is not recursively enumerable is not recursive (decidable).

- Halting problem of a Turing machine is recursively enumerable but it is **undecidable**
- The recursive set of languages is subset of recursively enumerable one. There are some recursively enumerable languages that are outside the recursive set. Some recursively enumerable languages are decidable and some are un-decidable.
- There are some problems which are not recursively enumerable. The complement of the halting problem is not recursively enumerable.

7.2 Enumerating a Language

SPPU - May 16, Dec. 16

University Question

Q. Write short note on : Recursively Enumerable Languages. **(May 2016, Dec. 2016, 3 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 multitape 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 :

     $$\omega_1 \# \omega_2 \# .... \# \omega_n \# \omega \#$$

     Where, the strings $\omega_1, \omega_2 ... \omega_n$ 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 $\omega_{i+1} = \omega_i \cdot \Sigma$ 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.

7.2.1 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 7.2.1 SPPU - Dec. 12, 4 Marks

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 :

$$f(1), f(2), f(3) \ldots$$

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 : $f(i_1), f(i_2) \ldots$

Now, we can define a function

$g : N \rightarrow S$ such that $g(n) = f(i_n)$

Thus, g is one-to-one and onto.

Hence, S is countable.

Example 7.2.2

Show that an infinite recursively enumerable set has an infinite recursive sub set.

Solution : Let $n_1, n_2, n_3, \ldots$ be a recursive enumeration of an infinite set S. For each n_i there must be, in this sequence, a later $n_j > n_i$.

$$\begin{aligned} \text{Let} \quad m_1 &= n_1, \\ m_2 &= n_{i_2}, \text{ the first } n_i \text{ greater than } n_1 \\ m_3 &= n_{i_3}, \text{ the first } n_i \text{ beyond } n_{i_2} \text{ greater than } n_{i_2}. \\ &\vdots \end{aligned}$$

The sequence $m_1, m_2, m_3 \ldots$ is a recursive enumeration of a subset of S without repetition in order of magnitude. This subset is infinite and recursive.

Example 7.2.3

Show that if S is uncountable and T is countable than S-T is uncountable.

Solution : We can prove this by contradiction.

Suppose **S-T** is countable set, $S\text{-}T = \{\omega_1, \omega_2 \ldots \omega_n\}$

$$\begin{aligned} \text{where,} \quad \omega_1 &= a_{11}, a_{12} \ldots \ldots \\ \omega_2 &= a_{21}, a_{22} \ldots \ldots \\ &\vdots \\ \omega_n &= a_{n1}, a_{n2} \ldots \ldots \end{aligned}$$

It is given that the set T is countable.

$$T = \{x_1, x_2 \ldots x_m\}$$

Since, the set **S** is uncountable, we can always find a word y such that

$y \in S$, $y \notin S\text{-}T$ and $y \notin T$

Hence, a contradiction.

Therefore, S-T is uncountable.

Example 7.2.4

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.

$$T = \{\omega_1, \omega_2, \ldots\}$$

Obviously, there exists a bijection between the set **N** of natural numbers and the set **T**.

$f : N \rightarrow T$, $f(n) = \omega$ for $\omega \in T$ 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 7.2.5 SPPU - Dec. 12, 4 Marks

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.

$$S = \{\omega_1, \omega_2 \ldots \omega_n\}$$

where,

$$\omega_1 = a_{11}\, a_{12}\, a_{13} \ldots\ldots ;$$
$$\omega_2 = a_{21}\, a_{22}\, a_{23} \ldots\ldots ;$$
$$\omega_n = a_{n1}\, a_{n2}\, a_{n3} \ldots\ldots$$

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 $\omega_k = x_{11}\, x_{22}\, x_{33} \ldots$, such that $\omega_k \notin S$.

$$x_{ii} = a_{ii} + 1, \quad \text{if } a_{ii} < 5$$
$$= 4, \quad \text{if } a_{ii} \geq 5$$

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.

7.3 Chomsky Hierarchy

SPPU - May 12, Dec. 12, Dec. 13

University Questions

Q. Explain the Chomsky Hierarchy of grammar and describe the machines that accept each type of grammar. **(May 2012, 8 Marks)**

Q. Explain in detail Chomsky classification of Languages with suitable examples. Clearly state the machines that accept each type of grammar. **(Dec. 2012, 8 Marks)**

Q. Explain in detail "Chomsky Hierarchy" stating the types of grammars, types of machines and types of languages. **(Dec. 2013, 6 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.

7.3.1 Type 3 or Regular Grammar

A grammar is called type 3 or regular grammar if all its productions are of the following forms :

$$A \to \varepsilon; \qquad A \to a$$

$$A \to aB; \qquad A \to Ba$$

where $a \in \Sigma$ and $A, B \in V$.

A language generated by type 3 grammar is known as regular language.

7.3.2 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

$A \to \alpha$ 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 is context free but not the reverse.

7.3.3 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.

$$\alpha \to \beta$$

where the β is atleast as long as α. α contains atleast one variable.

Example 7.3.1 : 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 :

$$P = \left\{ \begin{array}{l} S \to aSBC \mid aBC \\ CB \to BC,\ aB \to ab,\ bB \to bb, \\ bC \to bc,\ cC \to cc \end{array} \right\}$$

A string of the form $a^n b^n c^n$ can be generated as given below.

$S \overset{*}{\Rightarrow} a^{n-1} S(BC)^{n-1}$

[by applying $S \to aSBC$, $(n-1)$ times]

$\Rightarrow a^{n-1} aBC(BC)^{n-1}$

[by applying $S \to aBC$]

$\Rightarrow a^{n-1} abC(BC)^{n-1}$

[by applying $aB \to ab$]

$\Rightarrow a^{n-1} abB^{n-1} C^n$

[by applying $CB \to BC$, $n-1$ times]

$\Rightarrow a^n b^{n-1} bC^n$

[by applying $bB \to bb$ several times]

$\Rightarrow a^n b^{n-1} bcC^{n-1}$

[by applying $bC \to bc$]

$\Rightarrow a^n b^n c^n$

[by applying $cC \to cc$ several times]

Example 7.3.2

Write a grammar for : $L = \{a^{n^2} \mid n \geq 1\}$

Solution :

$$\begin{aligned} S &\to A_3A_4 \mid a \\ A_3 &\to A_1A_3A_2 \mid A_1A_2 \\ A_1 A_2 &\to a A_2 A_1 \\ A_1 a &\to a A_1 \\ A_2 a &\to a A_2 \\ A_1 A_4 &\to A_4 a \\ A_2 A_4 &\to A_5 a \\ A_2 A_5 &\to A_5 a \\ A_5 &\to a \end{aligned}$$

7.3.4 Type 0 or Unrestricted Grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the form $\alpha \to \beta$, then length of α could be more than length of β.

- Every grammar 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.

7.3.5 Compare Type 0, Type 1, Type 2 and Type 3 Grammars

Type	Name of Languages Generated	Restriction on Productions	Accepting Machine
0	Recursively enumerable	$\alpha_1 \to \alpha_2$ where $\alpha_1, \alpha_2 \in (V \cup T)^*$	Turing machine
1	Context sensitive	$\alpha_1 \to \alpha_2$ where $\alpha_1, \alpha_2 \in (V \cup T)^*$ and $\lvert\alpha_1\rvert \leq \lvert\alpha_2\rvert$	Turing machine with bounded tape. Length of the tape in finite

Type	Name of Languages Generated	Restriction on Productions	Accepting Machine
2	Context free	$X \to \alpha_1$ where $X \in V$ and $\alpha_1 \in (VUT)^*$	PDA
3	Regular	$X \to aY \mid a \mid \varepsilon \mid Ya$ where, $X, Y \in V$ and $a \in T$	FA

A general hierarchy of various languages is given in Fig. 7.3.1.

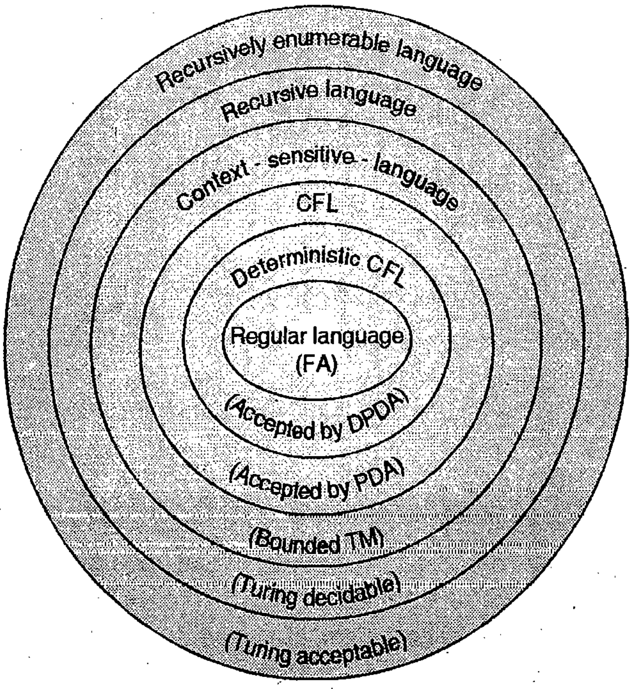


Fig. 7.3.1 : Hierarchy of languages

7.4 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 : TM's Halting Problem

7.4.1 Halting Problem of a Turing Machine

SPPU - May 12, Dec. 12, May 13, Dec. 13, May 15, Dec. 15, May 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, 6 Marks)**

Q. Justify "Halting Problem of Turing machine is undecidable". **(Dec. 2013, 5 Marks)**

Q. Prove that the halting problem in Turing Machines is undecidable. **(May 2015, 5 Marks)**

Q. Explain : "The halting problem in turing machines is undecidable". **(Dec. 2015, 5 Marks)**

Q. Write short note on :Halting Problem of Turing Machine. **(May 2016, 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

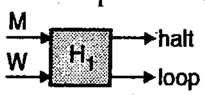
- Moves of a turing machine can be represented using a binary number. Thus, a Turing machine can be represented using a string over $\Sigma^*(0,1)$. This concept has already been explained in the Chapter 6.
- Unsolvability of halting problem of a Turing machine can be proved through the method of contradiction.

Step 1 : Let us assume that the halting problem of a Turing machine is solvable.

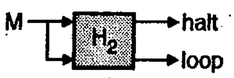
There exists a machine H_1(say). H_1 takes two inputs :

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.

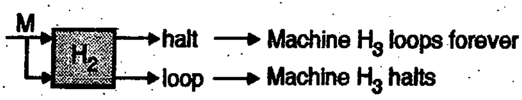


Step 2 : Let us revise the machine H_1 as H_2 to take M as both inputs and H_2 should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.



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.



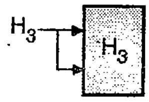
H_3 will do the opposite of the output of H_2.

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 7.4.1 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 : Post Correspondence Problem

7.4.2 Un-decidability of Post Correspondence Problem

SPPU - May 12, Dec. 12, May 13, Dec. 13, May 14

University Questions

Q. Write a short Note : Post correspondence Problem. **(May 2012, May 2014, 4 Marks)**

Q. Write a note on post correspondence problem. **(Dec. 2012, 8 Marks)**

Q. Explain in detail Post Correspondence Problem and Modified Post Correspondence Problem with suitable example. **(May 2013, 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 :

$$A = \{x_1, x_2, x_3 \ldots x_k\}$$

$$B = \{y_1, y_2, y_3 \ldots y_k\}$$

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 $x_i\, x_j \ldots x_m$ is equal to $y_i\, y_j \ldots y_m$.

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 7.4.2 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.

$$A = \{u_1, u_2, u_3 \ldots u_m\}$$

$$B = \{v_1, v_2, v_3 \ldots v_m\}$$

Let us take a new set of symbols $a_1, a_2 \ldots a_m$ such that

$$\{a_1, a_2 \ldots a_m\} \cap \Sigma = \phi.$$

Symbols $a_1, a_2 \ldots a_m$ 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 $u_i\, u_j\, u_k \ldots a_k\, a_j\, a_i$. Over alphabet $\Sigma \cup \{a_1, a_2, \ldots a_m\}$ can be defined using the set of productions :

$$G_A = \begin{Bmatrix} A \rightarrow u_1\, A\, a_1 \mid u_2\, A\, a_2 \mid \ldots \mid u_m\, Aa_m \\ u_1\, a_1 \mid u_2\, a_2 \mid \ldots \mid u_m\, a_m \end{Bmatrix}$$

Similarly a string of the form $v_i\, v_j\, v_k \ldots a_k\, a_j\, a_i$ over alphabet $\Sigma \cup \{a_1, a_2 \ldots a_m\}$ can be defined using the set of productions :

$$G_B = \begin{Bmatrix} B \rightarrow v_1\, A\, a_1 \mid v_2\, A\, a_2 \mid \ldots \mid v_m\, Aa_m \\ v_1\, a_1 \mid v_2\, a_2 \mid \ldots \mid v_m\, a_m \end{Bmatrix}$$

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}

$$S \rightarrow A$$

$$S \rightarrow B$$

- 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 $i_1, i_2, \ldots, i_m$ is a solution to this instance of PCP. Two derivations for the above string in G_{AB} is :

$$S \Rightarrow A \Rightarrow u_{i_1}\, A\, a_{i_1} \Rightarrow u_{i_1}\, u_{i_2}\, A\, a_{i_1}\, a_{i_2} \Rightarrow \ldots \Rightarrow$$

$$u_{i1}\, u_{i2} \ldots u_{im}\, a_{i1}\, a_{i2} \ldots a_{im}$$

$$S \Rightarrow B \Rightarrow v_{i_1}\, B\, a_{i_1} \Rightarrow v_{i_1}\, v_{i_2}\, B\, a_{i_1}\, a_{i_2} \Rightarrow \ldots \Rightarrow$$

$$v_{i1}\, v_{i2} \ldots v_{im}\, a_{i1}\, a_{i2} \ldots a_{im}$$

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 7.4.3

Prove that the blank tape halting problem is un-decidable.

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 7.4.4

Let G_1 and G_2 be context free grammars, and r be a regular expression. Then the following are un-decidable :

(a) $L(G_1) \cap L(G_2) = \phi$ (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) $L(G_1) \cap L(G_2) = \phi$ 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

$$L(G_2) = \{u_1u_2 \ldots u_n C v_n^R v_{n-1}^R \ldots v_1^R . \mid n \geq 1 \text{ and } (u_i, v_i) \in P\}$$

- G_1 and G_2 can be constructed easily.
- $L(G_1) \cap L(G_2) = \{\omega C \omega^R$ such that ω 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 $L(G_1) \cap L(G_2)$ is ϕ 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

$$L(G_1) \cap L(G_2) = \phi ?$$

(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 a 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 7.4.5

For following decision problem, determine whether it is decidable or un-decidable, prove the same.

(1) Given a TMT, 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.

7.4.3 Modified PCP Problem

SPPU - May 13, Dec. 13

University Question

Q. Explain in detail Modified Post Correspondence Problem with suitable example. **(May 2013, Dec. 2013, 3 Marks)**

The modified PCP problem is as follows :

Let us consider two lists A and B of K strings each. Each string is from Σ^*.

$$A = x_1, x_2, \ldots, x_k \text{ and } B = y_1, y_2, \ldots, y_k$$

does there exist a sequence of integers $i, i_2, \ldots, i_r$ such that

$$x_1 x_{i_2} x_{i_2} \ldots x_{i_k} = y_1, y_{i_1}, y_{i_2} \ldots y_{i_k} ?$$

In the modified PCP, the solution is required to start with the first string on each list.

7.5 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) = 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) = 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

$= 2(n-1) + 1$ moves

$\vdots$

Find the centre character c = 1 move

$\therefore$ Total number of moves

$= (2n+1) + (2(n-1)+1) + \ldots (2(n-(n-1))+1) + 1$

$= 2(1+2+\ldots+n) + n = \cancel{2} \times \dfrac{(n)(n+1)}{\cancel{2}} + n$

$= n^2 + 2n$

$$\therefore T(n) = n^2 + 2n = O(n^2)$$

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

$$S(n) = n + 1$$

[n – 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.

Syllabus Topic : Nondeterministic Polynomial Time

7.5.1 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.
2. 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 $P \subset NP$.

7.5.2 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 Classes P and NP

7.6 The Classes P and NP

SPPU - Dec. 14, Dec. 15

University Questions

Q. What do you mean by class NP problems ? **(Dec. 2014, 8 Marks)**

Q. Write note on P, NP with examples. **(Dec. 2015, 6 Marks)**

Q. Differentiate between P-class problems and NP-class problems. **(Dec. 2015, 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.

Syllabus Topic : Problem Solvable in Polynomial Time

7.6.1 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).

Syllabus Topic : An Example - Kruskal's Algorithm

7.6.2 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. 7.6.2 gives some of the spanning trees of the graph shown in Fig. 7.6.1.

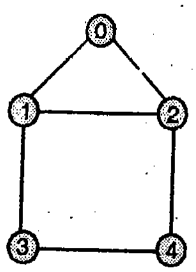
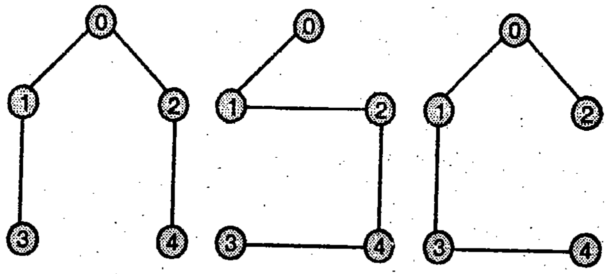


Fig. 7.6.1 : A sample connected graph

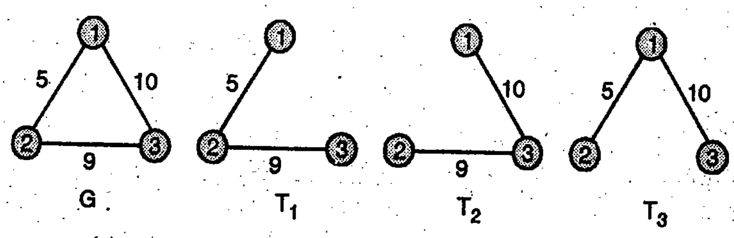


(a) (b) (c)

Fig. 7.6.2 : Spanning trees of the graph of Fig. 7.6.1

7.6.3 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.



(a) (b) (c) (d)

Fig. 7.6.3 : 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**

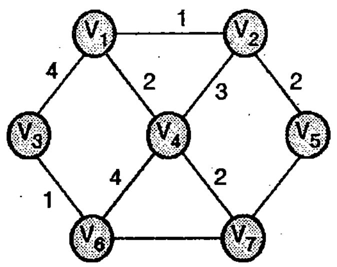
7.6.4 Kruskal's Algorithm

SPPU - May 16

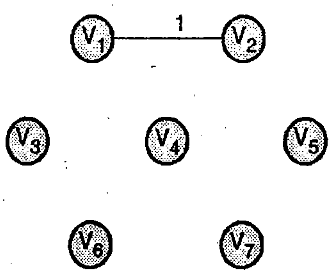
University Question

Q. What is Kruskal's Algorithm ? **(May 2016, 4 Marks)**

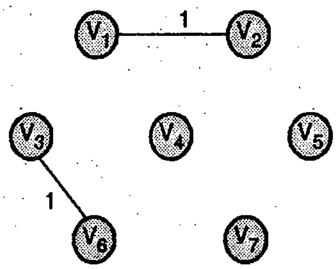
It is another method for finding the minimum cost spanning tree of the given graph. In Kruskal's algorithm, edges are added to the spanning tree in increasing order of cost. If any selected edge e forms a cycle in the spanning tree, it is discarded. Fig. 7.6.4 shows the sequence in which the edges are added to the spanning tree.



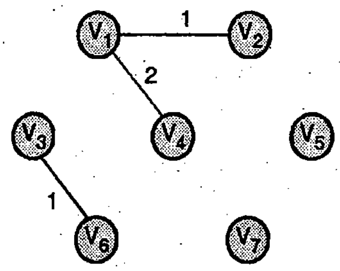
(a) Given graph G



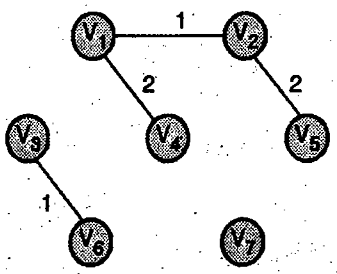
(b) Spanning tree, edge (V_1, V_2) is added



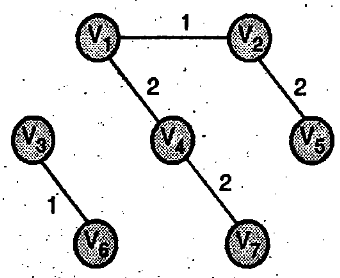
(c) Spanning tree, edge (V_3, V_6) is added



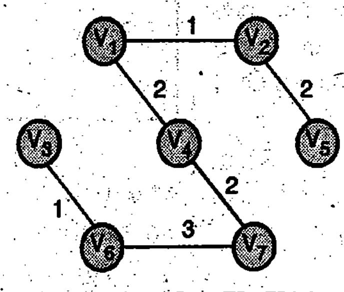
(d) Spanning tree, edge (V_1, V_4) is added



(e) Spanning tree, edge (V_2, V_5) is added



(f) Spanning tree, edge (V_4, V_7) is added



(g) Spanning tree, edge (V_6, V_7) is added

Fig. 7.6.4 : 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 $e_1, e_2, \ldots e_k$.
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 $(e_1, e_2, \ldots, e_k)$
   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.

7.6.5 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 it's 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.

Syllabus Topic : Polynomial-Time Reductions

7.6.6 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. **(Dec. 2015, 8 Marks)**

Q. What do you mean by Polynomial Time Reduction ? Explain with suitable example. **(May 2016, 8 Marks)**

Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. A polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ 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-Complete Problems

7.6.7 NP-Complete Problems

SPPU - May 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 Reduction? Explain with suitable example. **(May 2016, 8 Marks)**

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.

Syllabus Topic : An NP-Complete Problems

7.7 An NP-Complete Problem

SPPU - Dec. 14, May 16

> **University Questions**
>
> **Q.** Write note on NP-C problems with examples. **(Dec. 2014, 3 Marks)**
>
> **Q.** Why do we need to reduce existing problems to NP-Complete problems ? Explain with example. **(Dec. 2014, 8 Marks)**
>
> **Q.** What do you mean by NP-Problem ? **(May 2016, 4 Marks)**

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

Syllabus Topic : The Satisfiability Problem (SAT)

7.7.1 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 boolem 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.

Syllabus Topic : An NP Example : Traveling Salesman Problem

7.7.2 Traveling Salesman Problem (TSP)

Given a set of cities $C = \{ C_1, C_2, \ldots, C_n\}$ 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.

Syllabus Topic : Tractable and Intractable

7.7.3 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.

7.7.4 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 covert the clauses which do not contain three literals in to the one 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) $(x_1, x_2, \ )$.

If clause ha only one literal (x), we can recursively introduce literals u_1 and u_2 such that (x) corresponds to (x, u_1, u_2), $(x, u_1, \ )$, $(x, \ , u_2)$, $(x, \ , \ )$.

If the clause has more than three literals $(x_1, x_2, x_3, \ldots, x_n)$, we can rewrite it as follow:

$(x_1, x_2, u_1), (x_3, \quad , u_2), (x_4, \quad , u_3), \ldots, (x_{n-2}, \quad , u_{n-3}), (x_{n-1}, x_n, \quad)$

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 xi encounters in new arrangement. It can be seen that, if original clause is satisifiable, than 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 $(x_{n-1}, x_n, \quad)$ 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 SAT ≤ 3SAT, and hence 3SAT is NP complete problem.

Example 7.7.1 SPPU - Dec. 14, 4 Marks

What do you mean by class NP problems ? Justify why the Travelling Salesman Problem is a class NP problem.

Solution :

Travelling salesman problem

Given a set of cities $C = \{C_1, C_2, \ldots, C_n\}$ with $n > 1$ and a function d which assigns to each pair of cites (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.

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.

Syllabus Topic : Representing Satisfiability, Instances

7.8 Representing SAT Instances

A boolean expression is represented using the symbols ∧, ∨, ¬, 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 ∧, ∨ ¬ and parenthesis.

We can encode instances of SAT in a straightforward way :

1. The symbols ∧, ∨, ¬, (, 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.

Syllabus Topic : NP Completeness of the SAT Problem

7.8.1 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.

Syllabus Topic : A Restricted Satisfiability Problem

7.9 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. 7.9.1

Assuming P is already established as NP-complete, the NP-completeness of Q is established through a polynomial-time reduction from P to Q.

Syllabus Topic : Normal Forms for Boolean Expressions

7.9.1 Normal Forms for Boolean Expressions

There are two kinds of normal forms :

1. Conjunctive normal form
2. Disjunctive normal form

- A boolean expressional is said to be in conjunctive normal form if it is a conjunction (∧) of disjunctions (∨) of literals. A boolean expression in CNF is written in the form

$$\bigwedge_i \bigvee_j b_{i,j}$$

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 (∨) of conjunction (∧) of literals. A boolean expression in DNF is written in the form

$$\bigvee_i \bigwedge_j b_{i,j}$$

where $b_{i,j}$ is either a variable or its negation.

Syllabus Topic : Converting Expressions to CNF

7.9.2 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 ¬'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 low

(b) $\neg (E \vee F) \Rightarrow \neg (E) \wedge \neg (F)$ – De Morgan's low

(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.

7.9.3 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.

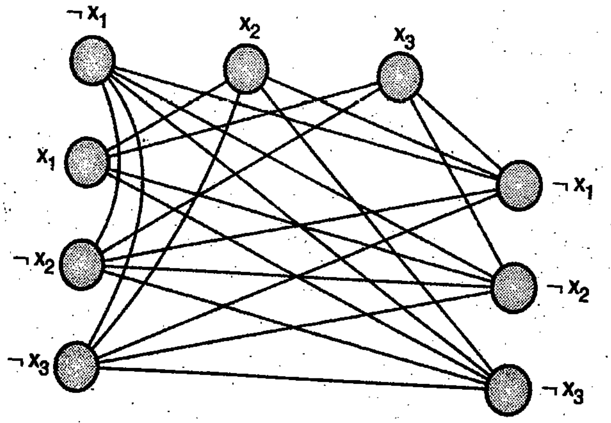


Fig. 7.9.2

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

$$\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

then G is the graph shown in Fig. 7.9.2.

In the given example, a satisfying argument of ϕ is ($x_1 = 0$, $x_2 = 0$, $x_2 = 0$, $x_3 = 1$). 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.

Syllabus Topic : The Problem of Independent Sets

7.9.4 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 = {< G, K > | G has an independent 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

$G_1 = (V, E_1)$, 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_1 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 n 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.

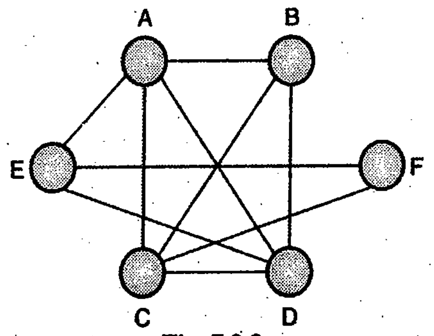


Fig. 7.9.3

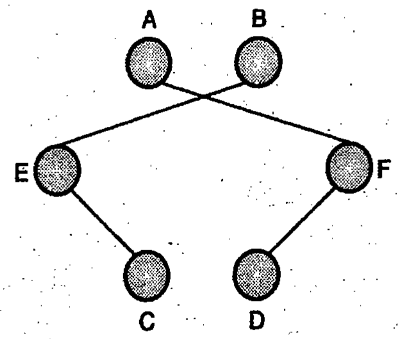


Fig. 7.9.4

For example, the graph G (V, E) has a clique, {A, B, C, D} given by Fig. 7.9.3. The complement of graph G is given by G_1, shown in Fig. 7.9.4 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.

Syllabus Topic : The Node-Cover Problem

7.9.5 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.

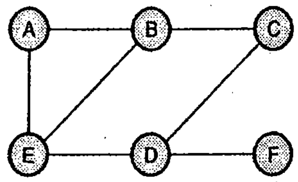


Fig. 7.9.5

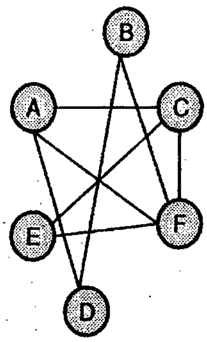


Fig. 7.9.6

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 = {<G, K> 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 $u \in V_1$. 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. 7.9.5. The complement of graph G is given by G_1 shown in Fig. 7.9.6 and has independent set given by {C, D, F}.

This proves the finding that the node cover is NP-complete.

7.9.6 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.

7.9.7 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.

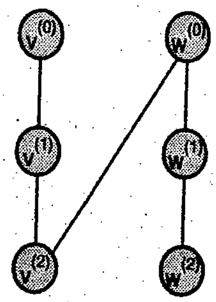


Fig. 7.9.7

Fig. 7.9.7 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 $V^{(1)}$ of G_u has only two edges, and therefore must appear in G Hamiltonian circuit.

□□□

Fully Solved University Question Papers

- Aug. 2017 (In Sem)
- Dec. 2017

August 2017 (In Sem.)

Q. 1(a) Compare DFA and NFA.

(Ans. : Refer section 2.6.5)

(Chap. 2, 3 Marks)

Ans. :

Difference between NFA and DFA

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.

Q. 1(b) Construct a DFA to accept strings of 0's and 1's having at least three consecutive 0's.

(Chap. 2, 3 Marks)

Ans. : [Hint : Refer Example as Ex. 2.2.28]

The required DFA is given below.

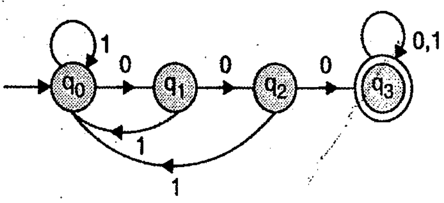


Fig. 1- Q. 1(b)

Q. 1(c) Construct an equivalent DFA for the following NFA. **(Chap. 2, 4 Marks)**

States/Σ	0	1
→ p	{p, q}	{q}
q*	{r}	{r}
R	-	{r}

Ans. : [Hint : Refer Example s 2.6.17(A)]

Step 1 : Transitions for state {p} are written

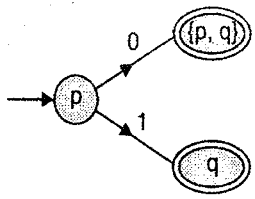


Fig. 1-Q. 1(c)

Step 2 : Transitions for state {q} are written

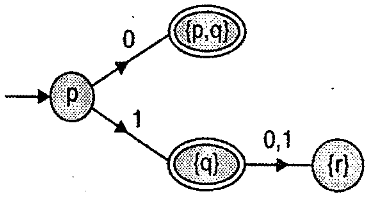


Fig. 2-Q. 1(c)

Step 3 : Transitions for the state {r} are written

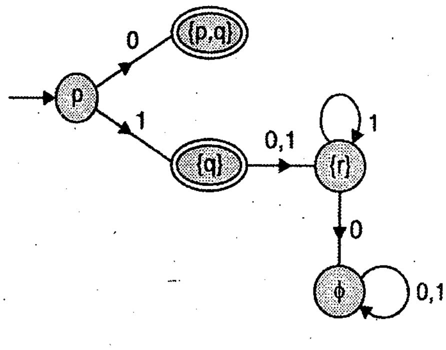


Fig. 3-Q. 1(c)

Step 4 : Transitions for the state {p, q} are written

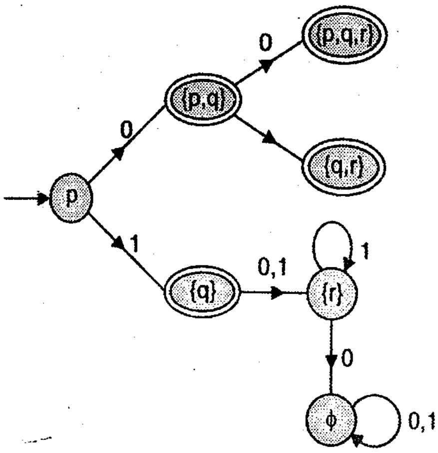


Fig. 4-Q. 1(c)

Step 5 : Transitions for state {p, q, r} and {p, q} are written.

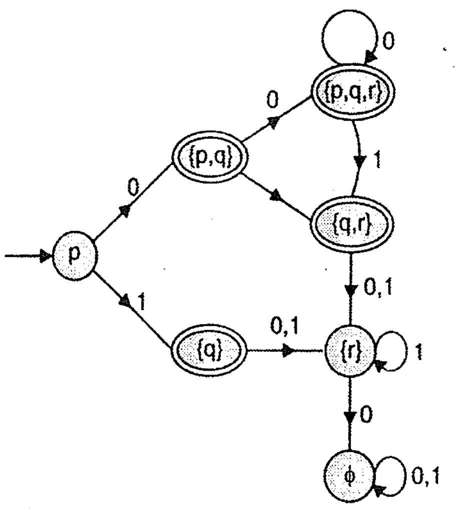


Fig. 5-Q. 1(c)

Q. 2(a) Compare NFA and NFA - ε.

(Chap. 2, 3 Marks)

Ans. : Both NFA and NFA - ε have the same power. We can always find an equivalent NFA for every NFA - ε. NFA - ε can make a transitions without any input. The above feature of NFA - ε makes its useful to find an equivalent NFA for any regular expression. Many a times it is easier to design NFA - ε for any regular language.

Q. 2(b) Construct a Mealy Machine which is equivalent to the Moore Machine given in the following table :

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

(Ans. : Refer section 2.8.1)

(Chap. 2, 3 Marks)

Ans. :

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. 1-Q. 2(b).

Present state	Next state		Output
	0	1	
→ p	s	q	0
q	q	r	1
r	r	s	0
s	s	p	0

Fig. 1-Q. 2(b) : 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	
→p	s	q	–
q	q	r	1
r	r	s	0
s	s	p, 0	0

Fig. 2-Q. 2(b)

(ii) Moving the output associated with q, which is 1 to transitions entering into state q, we get :

	0	1	
→ p	s	q, 1	–
q	q, 1	r	–
r	r	s	0
s	s	p, 0	0

Fig. 3-Q. 2(b)

(iii) Moving the output associated with r, which is 0 to transitions entering into the state r, we get :

	0	1	
→ p	s	q, 1	–
q	q, 1	r, 0	–
r	r, 0	s	–
s	s	p, 0	0

Fig. 4-Q. 2(b)

(iv) Moving the output associated with s, which is 0 to transitions entering the state s, we get :

	0	1
→ p	s, 0	q, 1
q	q, 1	r, 0
r	r, 0	s, 0
s	s, 0	p, 0

Fig. 5-Q. 2(b) : A trivial Mealy machine

Step 2 : The trivial Mealy machine obtained in Fig. 5-Q. 2(b) can not be minimized further. Final Mealy machine is given in Fig. 6-Q. 2(b) and Fig. 7-Q. 2(b).

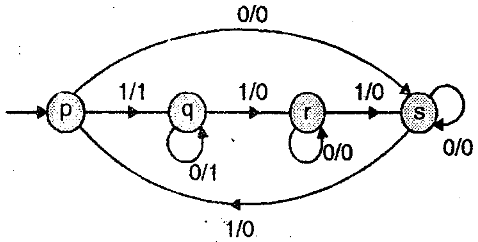


Fig. 7-Q. 2(b) : Transition diagram for final Mealy machine

	0	1
→ p	s, 0	q, 1
q	q, 1	r, 0
r	r, 0	s, 0
s	s, 0	p, 0

Fig. 6-Q. 2(b) : Transition table for final Mealy machine

Q. 2(c) Construct the DFA for the language of all strings that begin and end with same symbol over the alphabet $\Sigma = \{0, 1\}$.

(Chap. 2, 4 Marks)

Ans. : [Hint : Refer Example as Ex. 2.2.10(A)]

– The required DFA is given below.

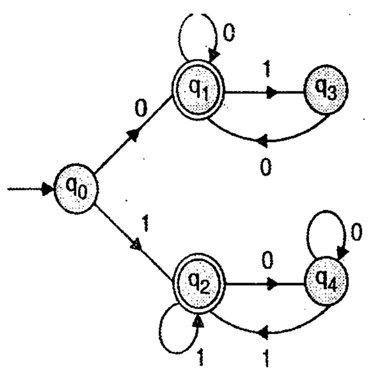


Fig. 1-Q. 2(c)

– The DFA will end in the final state q_1 if the starting and ending symbols are '0'.

– The DFA will end in the final state q_2 if the starting and ending symbols are '1'.

Q. 3(a) Define the following with suitable example.

(i) Regular expression and operations

(Ans. : Refer section 3.1)

(ii) Prove or disprove the following

$(rs + r)^*r = r(sr + r)^*$

(Chap. 3, 3 Marks)

Ans. : [Hint : (ii) Refer Example as Ex. 3.3.1(A)]

(i) Regular expression and operations

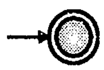
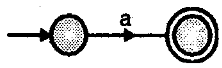
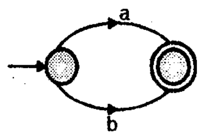
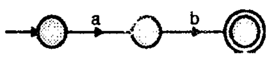
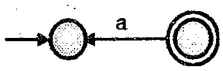
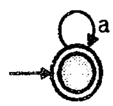
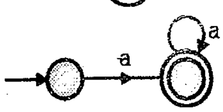
The set of strings accepted by finite automata is known as regular language. This language can also be described in a compact form using a set of operators. These operators are :

(1) + , union operator

(2) . , concatenation operator

(3) * , star or closure operator.

An expression written using the set of operators (+, · , *) and describing a regular language is known as regular expression. Regular expressions for some basic automata are given in Fig. 1-Q. 3(a).

Automata	Language	Regular expression
	{ε}	R.E. = ε
	{a}	R.E. = a
	{a, b}	R.E = a + b
	{ab}	R.E. = a · b or simply ab
	ϕ	R.E. = ϕ
	{ε, a, aa, aaa, ...}	R.E. = a*
	{a, aa, aaa, ...}	R.E. = aa* or a+

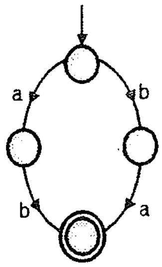
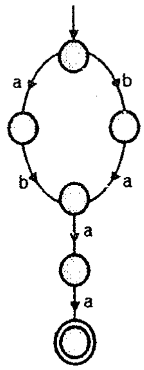
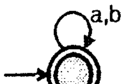
Automata	Language	Regular expression
	{ab, ba}	R.E. = ab + ba
	{abaa, baaa}	R.E. = (ab + ba) aa
	{ε, a, b, aa, ab, ba, bb, ...}	R.E. = $(a + b)^*$

Fig. 1-Q. 3(a) : 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 $|\omega| \geq 0$.

(ii) Prove or disprove the following $(rs + r)^*r = r(sr+r)^*$

Finite automata for LHS is given below.

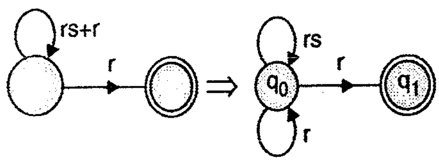


Fig. 2-Q. 3(a)

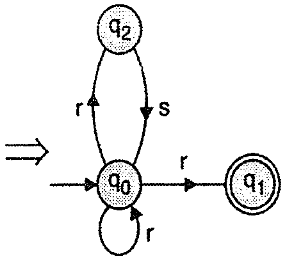


Fig. 3-Q. 3(a)

An equivalent DFA is given below.

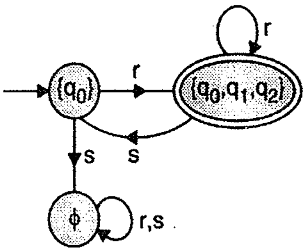


Fig. 4-Q. 3(a)

R.E. for the DFA can be written as

$r(sr + r)^* =$ R.H.S.

Hence the identity stands proved.

Q. 3(b) Construct the finite Automata defined over $\Sigma = \{0, 1\}$ for the following regular expression $1(01+10)^*+0(11+10)^*$

(Ans. : Refer Example 3.3.13(2))

(Chap. 3, 3 Marks)

Ans. :

F.A. for $1(01 + 10)^* + 0(11 + 10)^*$

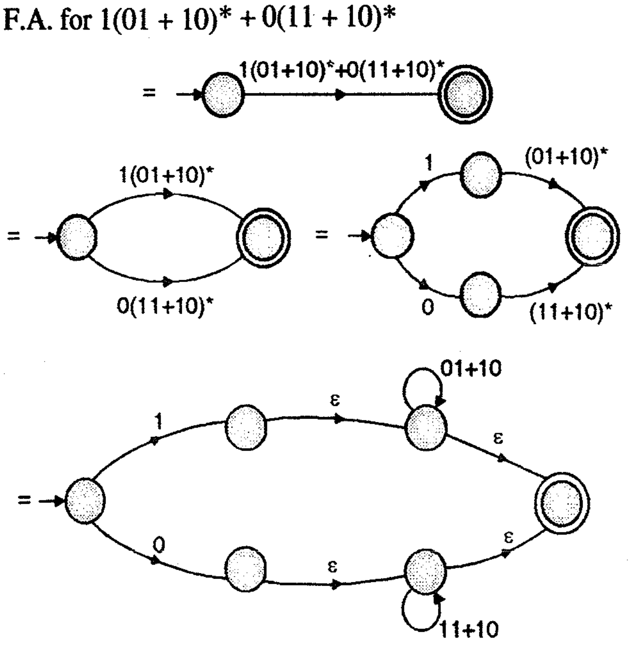


Fig. 1-Q. 3(b) (Contd...)

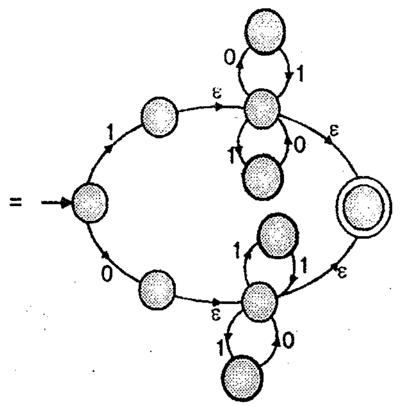


Fig. 1-Q. 3(b)

Q. 3(c) Using the pumping lemma for the regular set prove that $L = \{a^{i \text{ square}} \mid i \geq 1\}$ is not regular. *(Ans. : Refer Example 3.6.8)* **(Chap. 3, 4 Marks)**

Ans. :

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

$$|y| > 0$$

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.

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

$$|xy^2z| = |xyz| + |y|$$
$$= n^2 + (>0) \text{ as the length of } y > 0 \text{ and length of xyz is } n^2.$$

$$\therefore |xy^2z| > n^2 \quad \ldots(1)$$

again, $|xy^2z| = |xyz| + |y|$

Since the length of $|y| \leq n$ as $|xy| \leq n$ we can say that

$$|xy^2z| \leq n^2 + n$$

or, $|xy^2z| < n^2 + n + n + 1$ [n + 1 is added on the right of the inequality]

or, $|xy^2z| < (n+1)^2 \quad \ldots(2)$

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.

Q. 4(a) What are the algebraic laws of regular expression. *(Ans. : Refer section 3.2.2)* **(Chap. 3, 3 Marks)**

Ans. :

Algebraic Laws for Regular Expressions

There are a number of laws for algebraic laws, including :

1. Associativity and commutativity.
2. Identities and annihilators
3. Distributive laws
4. The idempotent law
5. Laws involving closures.

1. Associativity and commutativity

- The commutative law for union says that the union of two regular languages can be taken in any order.
- For any two languages L and M,

$$L + M = M + L$$

- The associativity law holds for union of regular languages.

or, $(L + M) + N = L + (M + N)$

2. Identities and annihilators

∈ is the identity and ϕ is the annihilator. There are three laws for regular expressions involving ∈ and ϕ :

1. $\phi + L = L + \phi = L$, ϕ is the identity for +
2. $\in \cdot L = L \cdot \in = L$, ∈ is the identity for concatenation
3. $\phi \cdot L = L \cdot \phi = \phi$, ϕ is annihilator for concatenation

3. Distributive laws

The left distributive law of concatenation over union holds for regular languages.

$$L(M + N) = LM + LN$$

The right distributive law of concatenation over union holds for regular languages.

$$(M + N)L = ML + NL$$

4. The idempotent law

It says that the union of two identical expression can be replaced by one copy of the expression.

$$L + L = L$$

5. Laws involving closures

These laws include :

1. $(L^*)^* = L$, closure of the closure does not changes the language
2. $\phi^* = \in$
3. $\in^* = \in$
4. $L^* = LL^* = L^*L$
5. $L^* = L^* + \in$

Q. 4(b) Convert the following regular expression to ε-NFA and find the ε-closure of all the states. (0+1)*.1.(0+1)

(Chap. 3, 3 Marks)

Ans. : [Hint : Refer Example as Ex. 3.3.20]

The $\in$ - NFA is given below.

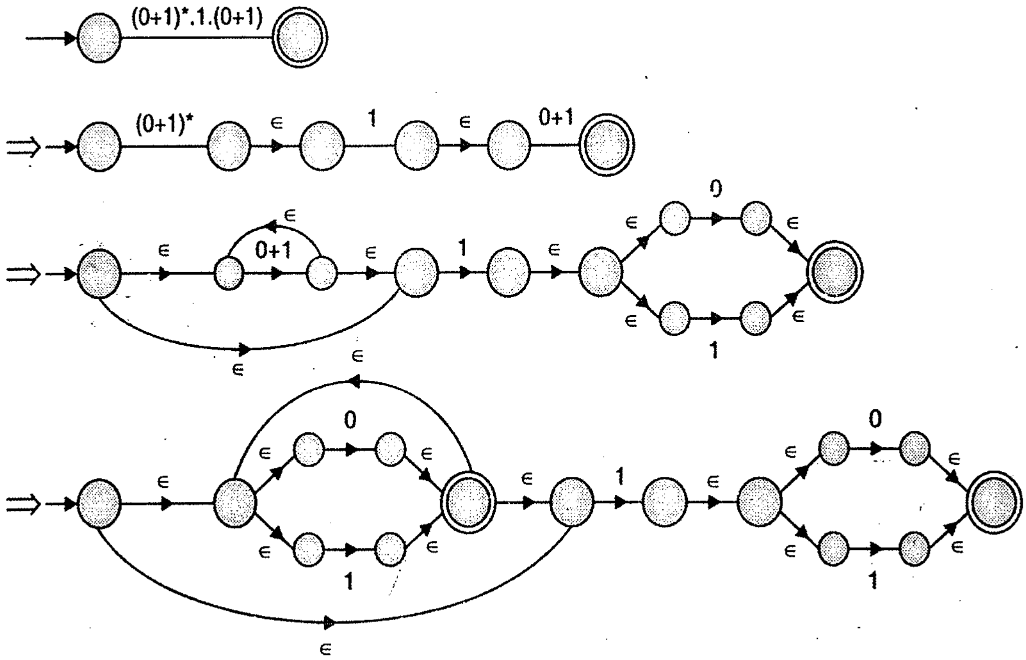


Fig. 1-Q. 4(b)

Removing unnecessary states, we get

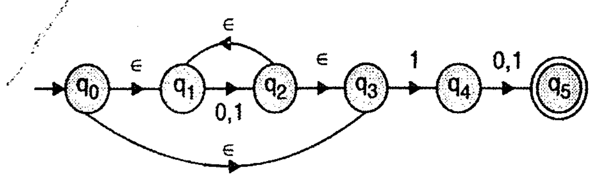


Fig. 2-Q. 4(b)

$\in$ - closure of states.

State	$\in$ - closure
q_0	$\{ q_0, q_1, q_3 \}$
q_1	$\{ q_1 \}$
q_2	$\{ q_2, q_3, q_1 \}$
q_3	$\{ q_3 \}$
q_4	$\{ q_4 \}$
q_5	$\{ q_5 \}$

Q. 4(c) Using the pumping lemma for the regular set, prove that $L = \{ a^m b^n \}$ is not regular.
(Ans. : Refer Example 3.6.2)

(Chap. 3, 4 Marks)

Ans. :

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

$$p + q > n \quad \text{and} \quad p = q + 1$$

Let us write w as xyz with

$$|y| > 0$$

and $\quad |xy| \le n$

y could take any of the given forms :

1. a^i .... case I
2. $a^i b^j$ ... case II
3. b^j ... case III

Step 3 : We want to find i so that $xy^iz \notin L$ 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.

$$xy^0z \notin L$$

Case II

xy^2z 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 ≥ 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.

Q. 5(a) Write in brief about "Sentential form" with reference to context free grammar.
(Ans. : Refer section 4.2.2.1)

(Chap. 4, 3 Marks)

Ans. :

Sentential Form

Let us consider a grammar given below :

$S \rightarrow A1B$ (Production 4-5)

$A \rightarrow 0A \mid \varepsilon$ (Production 4-6)

$B \rightarrow 0B \mid 1B \mid \varepsilon$ (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.

$S \rightarrow A1B$ [Starting production]

$\rightarrow 0A1B$ [Using the production $A \rightarrow 0A$]

$\rightarrow 00A1B$ [Using the production $A \rightarrow 0A$]

$\rightarrow 001B$ [Using the production $A \rightarrow \varepsilon$]

$\rightarrow 0010B$ [Using the production $B \rightarrow 0B$]

$\rightarrow 00101B$ [Using the production $B \rightarrow 1B$]

$\rightarrow 00101$ [Using the production $B \rightarrow \varepsilon$]

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.

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \mid \alpha \in (V \cup T)^*$$

- 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.

Q. 5(b) Write equivalent left linear grammar for the following right linear grammar.

$S \rightarrow 0A$

$A \rightarrow 10A|\epsilon$ **(Chap. 4, 3 Marks)**

Ans. : [**Hint :** Refer Example as Ex. 4.2.16(A)]

Transition system for two given right-linear grammar is given below.

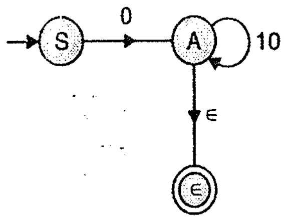


Fig. 1-Q. 5(b)

Interchanging the start state with the final state and reversing direction of transitions, we get

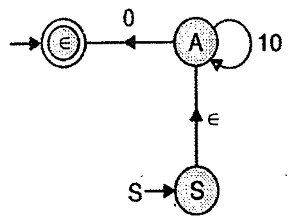


Fig. 2-Q. 5(b)

Left-linear grammar can be written from the above transition system.

$S \rightarrow A$

$A \rightarrow A\,10 \mid 0$

Q. 5(c) Write context free grammar for the following language 0(0+1)*01(0+1)*1.

(Chap. 4, 4 Marks)

Ans. : [**Hint :** Refer Example as Ex. 4.2.16(A)]

$S = 0X01X1$ [where X generates (0+1)*]

$X \rightarrow 0X \mid 1X \mid \epsilon$

Q. 6(a) Eliminate ϵ-productions from the grammar G

$A \rightarrow aBb|bBa$

$B \rightarrow aB|bB|\epsilon$ **(Chap. 4, 3 Marks)**

Ans. : [**Hint :** Refer Example as Ex. 4.4..6(A)]

The symbol 'B' is nullable. Re-writing the given grammar after elimination of ϵ - productions, we get

$A \rightarrow aBb \mid ab \mid bBa \mid ba$

$B \rightarrow aB \mid a \mid bB \mid b$

Q. 6(b) Write CFL for the following CFG

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

(Ans. : Refer Example 4.2.21)

(Chap. 4, 3 Marks)

Ans. : 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,

  $B \rightarrow bS$

  and $B \rightarrow aBB$

Thus the set of productions for the above language are given by :

$$P = \{ \quad S \to aB \mid bA$$
$$A \to aS \mid bAA \mid a$$
$$B \to bS \mid aBB \mid b$$
$$\}$$

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

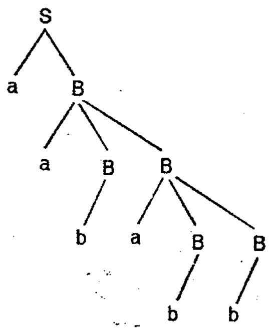


Fig. 1-Q. 6(b) : Parse tree for aababb

Q. 6(c) Write an equivalent left-linear grammar for the right-linear grammar.

$S \to 0A|1B$; $A \to 0C|1A|0$

$B \to 1B|1A|1$; $C \to 0|0A$

(Ans. : Refer Example 4.7.10)

(Chap. 4, 4 Marks)

Ans. :

Step 1 : Transition system for the given right linear grammar is given below :

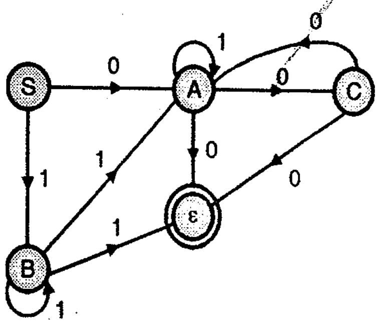


Fig. 1-Q. 6(c)

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get

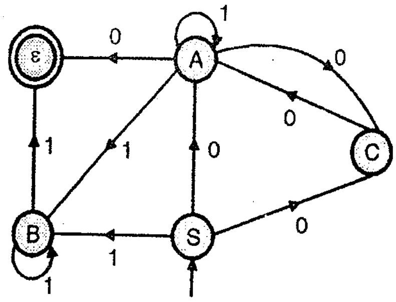


Fig. Ex. 4.7.10(a)

Step 3 : Writing of left linear grammar from the transition system, we get :

$S \to C0 \mid A0 \mid B1$

$A \to A1 \mid C0 \mid B1 \mid 0$

$B \to B1 \mid 1$

$C \to A0$

❑❑❑

Dec. 2017

Q. 1(a) Construct DFA for language defined by $\Sigma = \{0, 1\}$ where

S = {strings ending with 0 always}

S = {strings representing odd binary numbers}

S = {strings over Σ^* with total number of 0's even}

(Chap. 2, 6 Marks)

Ans. : [Hint : Refer Example as Ex. 2.2.4(A)]

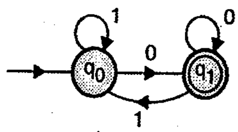


Fig. 1-Q. 1(a)

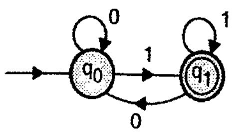


Fig. 2-Q. 1(a)

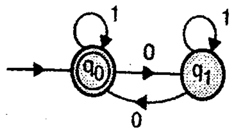


Fig. 3-Q. 1(a)

Q. 1(b) Let M = { $\{q_0, q_1\}$, {0, 1}, δ, q_0, $\{q_1\}$ } be an NFA **(Chap. 2, 6 Marks)**

Where

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

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

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

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

Construct an equivalent DFA.

Ans. : [Hint : Refer Example as Ex. 2.6.12]

DFA is drawn in stages.

Step 1 : Transitions from the starting state q_0

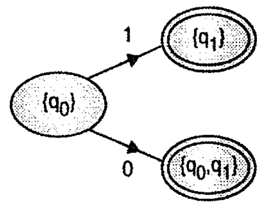


Fig. 1-Q. 1(b)

Step 2 : Transitions from the state { q_1 } are added

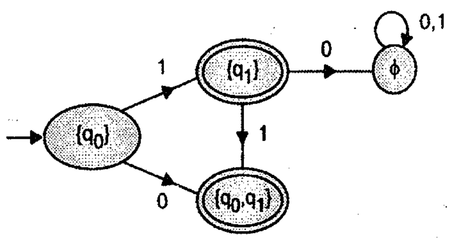


Fig. 2-Q. 1(b)

Step 3 : Transitions from the state { q_0, q_1 } are added

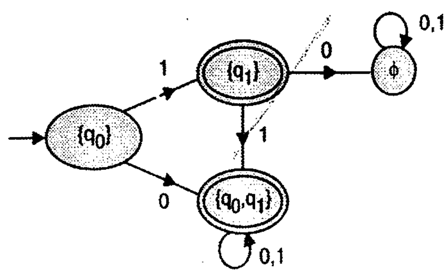


Fig. 3-Q. 1(b)

Q. 1(c) Write short notes on : **(Chap. 5, 8 Marks)**

(i) Chomsky Normal Form

(Ans. : Refer section 4.5.1)

(ii) Greibach Normal Form

(Ans. : Refer section 4.5.2)

Ans. :

(i) Chomsky Normal Form (CNF)

A Context Free Grammar (CFG) without ε-production is said to be in CNF if every production is of the form :

1. $A \to BC$, where $A, B, C \in V$.
2. $A \to a$, 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.

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 consists only of variables.

   i.e. every production of the form

   $A \to \alpha$ with $|\alpha| \geq 2$, α should consists only of variables.

   Example : Consider a production

   $A \to V_1V_2aV_3bV_4$.

   Terminal symbols a and b can be removed by rewriting the production

   $A \to V_1V_2aV_3bV_4$

   as $A \to V_1V_2C_aV_3C_bV_4$

   And adding two productions

   $C_a \to a$ and $C_b \to b$
3. Every production deriving 3 or more variables ($A \to \alpha$ 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 $A \to X_1X_2\ldots X_n$ where $n \geq 3$ and as X_i's are variables.

   The production $A \to X_1X_2\ldots X_n$ should be broken down as given below :

$$A \to X_1C_1$$
$$C_1 \to X_2C_2$$
$$C_2 \to X_3C_3$$
$$\vdots$$
$$C_{n-2} \to X_{n-1}X_n$$

Each with two variables on the right.

(ii) Greibach Normal Form (GNF)

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

$A \to a\alpha$,

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.

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 $A \to A\alpha$ 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

$A \to A\alpha$ ... [Left recursive]

$A \to \beta$ ... [For termination of recursion]

The language generated by above production is :

$A \to A\alpha$ [From production $A \to A\alpha$]

$\to A\alpha\alpha$ [From production $A \to A\alpha$]

$\to A\alpha\alpha\alpha$ [From production $A \to A\alpha$]

:

:

$\to A\alpha^n$ [From production $A \to A\alpha$]

$\to \beta\alpha^n$ [From production $A \to \beta$]

Right recursive grammar for $\beta\alpha^n$:

A right recursive grammar for $\beta\alpha^n$ can be written as :

$A \to \beta B \mid \beta$ [where B generates a string α^n, production $A \to \beta$ is for termination of recursion]

$B \to \alpha B \mid \alpha$

Thus a left recursive grammar

$$A \to A\alpha \mid \beta$$

can be written using a right recursive grammar as:

$A \to \beta B \mid \beta$ $B \to \alpha B \mid \alpha$	right recursive grammar

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. $A \to Aa \mid b$	$\{b, ba, baa, \ldots ba^n\}$	$A \to b \mid bB$ $B \to aB \mid a$
2. $A \to Aa \mid b \mid c$	$\left\{\begin{matrix} b, ba, baa, \ldots ba^n, \\ c, ca, caa, \ldots ca^n \end{matrix}\right\}$	$A \to b \mid c \mid bB \mid cB$ $B \to aB \mid a$
3. $A_1 \to A_1A_2A_3 \mid A_2A_3$	$\left\{\begin{matrix} A_2A_3, A_2A_3A_2A_3, \ldots \\ \ldots, (A_2A_3)^n \end{matrix}\right\}$	$A_1 \to A_2A_3 \mid A_2A_3B_1$ $B_1 \to A_2A_3B_1 \mid A_2A_3$
4. $A_1 \to A_1A_2A_3 \mid A_4A_1 \mid A_5A_3$	$\left\{\begin{matrix} A_4A_1, A_4A_1A_2A_3 \ldots \\ A_4A_1(A_2A_3)^n, \\ A_5A_3, A_5A_3A_2A_3, \ldots \\ A_5A_3(A_2A_3)^n \end{matrix}\right\}$	$A_1 \to A_4A_1 \mid A_5A_3 \mid A_4A_1B_1 \mid A_5A_3B_1$ $B_1 \to A_2A_3B_1 \mid A_2A_3$
5. $S \to S10 \mid 0$	$\{0, 010, 01010, \ldots, 0(10)^n\}$	$S \to 0B \mid 0$ $B \to 10B \mid 10$

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 $A \to X_1X_2 \ldots X_i \ldots X_n$, other than X_1, every other symbol should be a variable. X_1 could be a terminal.

Example :

Consider a production $A \to V_1V_2aV_3bV_4$

Terminal symbols a and b can be removed by rewriting the production

$A \to V_1V_2aV_3bV_4$ as

$A \to V_1V_2C_aV_3C_bV_4$ and adding two productions.

$C_a \to a$

$C_b \to b$

Thus, at the end of step 2 all productions must be of the forms.

(a) $A \to \alpha$ (b) $A \to a$ (c) $A \to a\alpha$

Where 'a' is a terminal and α is a string of non-terminals.

3. Rename variables as $A_1, A_2, A_3 \ldots A_n$ to create A-productions.

   Example : Consider a grammar given below

   $S \to aXSY \mid YSX \mid b$

   The variables S, X and Y can be renamed as A_1, A_2 and A_3 respectively. Then the productions become

   $A_1 \to aA_2A_1A_3 \mid A_3A_1A_2 \mid b$ [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 from $B_i \rightarrow a\alpha$, where a is a terminal and α is a string of non-terminals.

Q. 2(a) Design a FA which checks the divisibility by 4 for a decimal number. **(Chap. 2, 6 Marks)**

Ans. : [**Hint : Refer Example 2.2.24**]

The required DFA is given below.

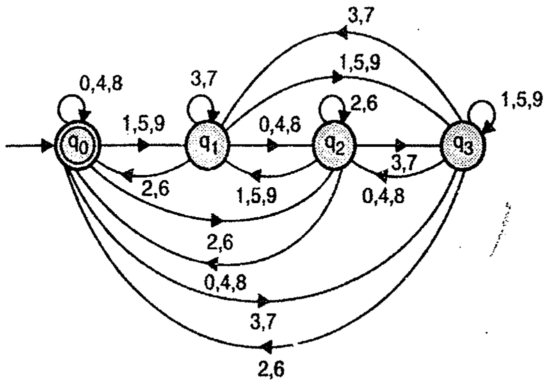


Fig. 1-Q. 2(a)

The state q_0 is for the running remainder as 0.

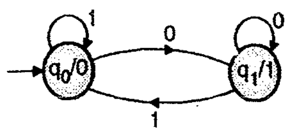
The state q_1 is for the running remainder as 1.

The state q_2 is for the running remainder as 2.

The state q_3 is for the running remainder as 3.

Q. 2(b) Construct a Moore and Mealy machine to generate 1's compliment of a given binary number. *(Ans. : Refer Example 2.7.6(a))* **(Chap. 2, 6 Marks)**

Ans. :



(a)

Fig. 1-Q. 2(b)

	0	1	Output
$\rightarrow q_0$	q_1	q_0	0
q_1	q_1	q_0	1

(b)

Fig. 1-Q. 2(b)

- 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. Solution for mealy machine.

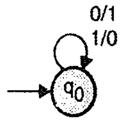


Fig. 1-Q. 2(b)

Q. 2(c) Write CFGs for given CFLs :

(i) Language containing the strings with equal number of a's and b'S. *(Ans. : Refer Example 4.2.21)*

(ii) Languages containing the strings containing a's and b's with at least 2 a's. *(Ans. : Refer Example 4.2.21)*

Ans. : [**Hint : (ii) Refer Example as Ex. 4.2.21(A)**]

(i) Language containing the strings with equal number of a's and b'

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,

$$B \rightarrow bS$$

and

$$B \rightarrow aBB$$

Thus the set of productions for the above language are given by :

$$P = \{ \quad S \rightarrow aB \mid bA$$
$$A \rightarrow aS \mid bAA \mid a$$
$$B \rightarrow bS \mid aBB \mid b$$
$$\}$$

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

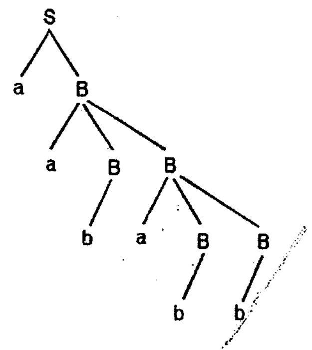


Fig.1-Q. 2(c) : Parse tree for aababb

(II) Languages containing the strings containing a's and b's with at least 2 a's

Step 1 : RE for the given language

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

Step 2 : CFG from RE

$S \rightarrow XaXaX$ where X generates $(a+b)^*$

$X \rightarrow aX \mid bX \mid \epsilon$

Q. 3(a) Define Turing Machine. Comment on language acceptance by Turing Machine.

(Ans. : Refer sections 6.1 and 7.1)

(Chap. 6, 4 Marks)

Ans. :

Introduction to Turing Machine

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 :

FA ≤ DPDA ≤ NPDA ≤ Post machine = Turing machine

Turing machine is capable of performing computations on inputs and producing a new result.

An abstract model of a turing machine is shown in Fig. 1-Q. 3(a).

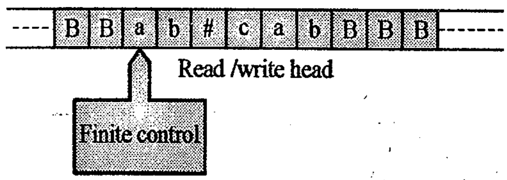


Fig. 1-Q. 3(a) : An example of a Turing Machine

- 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).

Recursively Enumerable and Recursive Language

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.

Q. 3(b) Write short notes on : **(Chap. 6, 6 Marks)**

(i) Universal Turing Machine

(Ans. : Refer section 6.5)

(ii) Multi-tape Turing Machine

(Ans. : Refer section 6.4.3)

(iii) Limitation of Turing Machine

Ans. : [Hint : Refer section 6.5]

(i) Universal Turing Machine

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 complier.
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

$\{(q_i,a_i,q_j,a_j,m_f) : q_i,q_j, \in Q \; ; a_i,a_j, \in \Gamma \; ;$

$m_f \in$ {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.

(ii) Multi-tape Turing Machine

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. 1-Q. 3(b).

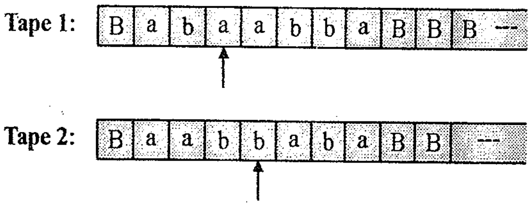


Fig. 1-Q. 3(b) : A two-tape turing machine

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.

(iii) Limitation of Turing Machine

1. There is a limit to the computational power of Turing Machine.
2. There are a number of problems, like halting problem of TM, which cannot be solved by TM.
3. TM cannot model concurrency in an algorithm well.

Q. 3(c) Construct a Turing Machine to accept the language of even number of 1's and even number 0's over $\Sigma = \{0, 1\}$ **(Chap. 6, 8 Marks)**

Ans. : [Hint : Please insert after Example 6.3.12]

Step 1 : Construct DFA for the given language.

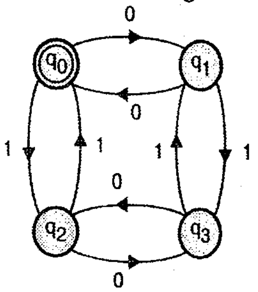


Fig. 1-Q. 3(c)

Step 2 : DFA to TM

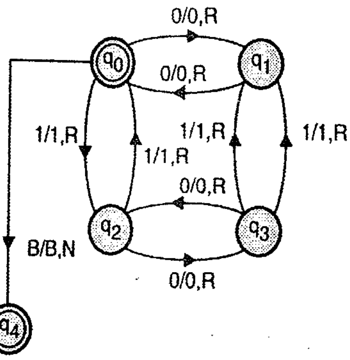


Fig.2-Q. 3(c)

Q. 4(a) Explain the representation of TM.
(Ans. : Refer section 6.2) **(Chap. 6, 4 Marks)**

Ans. :

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 $\rightarrow$ Move to left cell

R $\rightarrow$ Move to right cell

N $\rightarrow$ Remain in the current cell (No movement)

Q. 4(b) Design a Turing Machine to add two unary numbers. **(Chap. 6, 6 Marks)**

Ans. : [Hint : Refer Example as Ex. 6.3.2(A)]

This can be done easily by changing the first '0' to B and replacing the separator # with '0'.

Example

$3 + 4 = 7$

B	0	0	0	#	0	0	0	0	B	⇒	B	B	0	0	0	0	0	0	0	B

The TM is given below

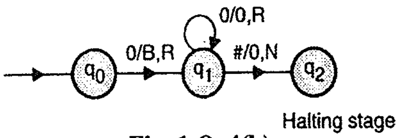


Fig. 1-Q. 4(b)

Q. 4(c) Construct TM for

L = {all strings with equal no. of a's and b's}.

(Ans. : Refer Example 6.2.8)

(Chap. 6, 8 Marks)

Ans. :

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.

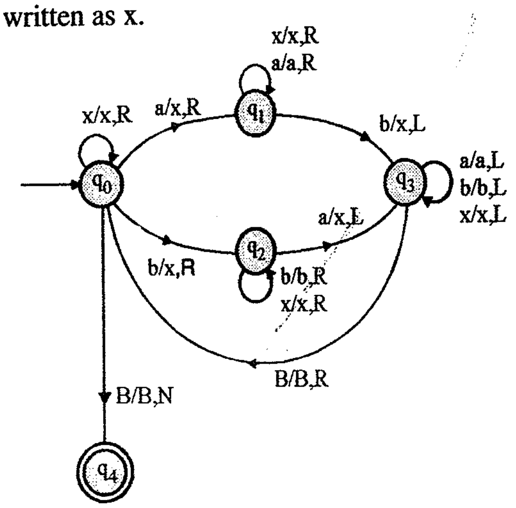


Fig. 1-Q. 4(c) : State transition diagram

	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	q_4 ← Halting state

Fig. 2-Q. 4(c) : 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 being given below.

$\vdash$B**a**bbaB $\vdash$Bx**b**baB $\vdash$B**x**xbaB $\vdash$**B**xxbaB
q_0 q_1 q_3 q_3

$\vdash$B**x**xbaB $\vdash$Bx**x**baB $\vdash$Bxx**b**aB $\vdash$Bxxx**a**B
q_0 q_0 q_0 q_2

$\vdash$Bxx**x**xB $\vdash$Bx**x**xxB $\vdash$B**x**xxxB $\vdash$**B**xxxxB
q_3 q_3 q_3 q_3

$\vdash$B**x**xxxB $\vdash$Bx**x**xxB $\vdash$Bxx**x**xB $\vdash$Bxxx**x**B
q_0 q_0 q_0 q_0

$\vdash$Bxxxx**B** $\vdash$Bxxxx**B**
q_0 q_4

Q. 5(a) Differentiate between FA and PDA. **(Chap. *, 6 Marks)**

Ans. :

Sr. No.	FA	PDA
1.	Less powerful than PDA.	More powerful than FA.
2.	Accepts regular language.	Accepts context free language.
3.	It does not have memory.	It has memory in the form of stack.
4.	Transition on the basis of current state and the next input.	Transition on the basis of current state top of the stack symbol and the next input.

Q. 5(b) Construct NPDA that accepts the language generated by S = S+S |S*S| 4.

(Ans. : Refer Example 5.6.5)

(Chap. 5, 6 Marks)

Ans. :

The equivalent PDA, M is given by :

$M = (\{q\}, \{+, *, 4, 2\}, \{+, *, 4, 2, S\}, \delta, q, S, \phi)$

where δ is given by :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, S + S), (q, S * S), (q, 4)\}$ | for every production in G

$$\left.\begin{array}{l} \delta(q, +, +) = \{(q, \varepsilon)\} \\ \delta(q, *, *) = \{(q, \varepsilon)\} \\ \delta(q, 2, 2) = \{(q, \varepsilon)\} \\ \delta(q, 4, 4) = \{(q, \varepsilon)\} \end{array}\right| \text{for every terminal in T.}$$

Q. 5(c) Illustrate the working of shift reduce parser for id+id*id.

Consider the following grammar :

E → E + E | T

T → T * F | F

F → {E} | id

(Ans. : Refer Example 5.9.2)

(Chap. 5, 6 Marks)

Ans. : Consider the grammar

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

and the input string id + id * id. Show the working of the shift reduce parser.

Solution :

Sr. No.	Stack	Input	Action
1.	empty	id + id *id	Shift
2.	id	+ id * id	Reduce by F → id
3.	F	+ id * id	Reduce by T → F
4.	T	+ id * id	Reduce by E → T
5.	E	+ id * id	Shift
6.	E +	id * id	Shift
7.	E + id	* id	Reduce by F → id
8.	E + F	* id	Reduce by T → F
9.	E + T	* id	Shift
10.	E + T*	id	Shift
11.	E + T * id	–	Reduce by F → id
12.	E + T * F	–	Reduce by T → T * F
13.	E + T	–	Reduce by
14.	E	–	Accept

Q. 6(a) What are the two different ways to define PDA acceptability ? *(Ans. : Refer section 5.4)*

(Chap. 5, 4 Marks)

Ans. :

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.

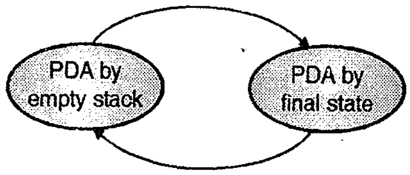


Fig. 1-Q. 6(a) : Equivalence of two PDAs

Acceptance by Final State

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 :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \alpha) \right\}$$

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.

Acceptance by Empty Stack

Let the PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$ then the language accepted through an empty stack is given by :

$$L(M) = \left\{ w \mid (q_0, w, z_0) \overset{*}{\underset{M}{\Rightarrow}} (q_1, \varepsilon, \varepsilon) \right\}$$

Where q_1 is any state belonging to Q and the stack becomes empty on application of input string w.

Q. 6(b) Construct PDA that accepts language generated by following CFG :

$s \rightarrow ss \mid (s) \mid ()$ **(Chap.5, 4 Marks)**

Ans. :

The equivalent PDA M is given by

$M = \{ \{ q_1 \}, \{(,)\}, \{(,), S\}, \delta, q, S, \phi \}$

Where δ is given by

$\delta(q, s) = \{ (q, ss), (q, (s)), (q, ()) \}$

$\delta(q, (, () = (q, \in)$

$\delta(q,),)) = (q, \in)$

Q. 6(c) Explain closure property of CFL with suitable example. *(Ans. : Refer section 4.9.1)* **(Chap. 4, 6 Marks)**

Ans. :

Closure Properties

- 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.

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.

$$\therefore \quad L(G) = L_1 \cup L_2$$

2. CFL is Closed under Concatenation

Theorem

If L_1 and L_2 are context-free languages, then L_1L_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 G1 and G2 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.

$$S \rightarrow S_1S_2$$

The start symbol S will generate a string w of the form :

$$w = w_1w_2, \text{ where } w_1 \in L_1 \text{ and } w_2 \in L_2.$$

The grammar G can be written as :

$$G = (V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup \{ S \rightarrow S_1S_2 \}, S)$$

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.

$$S \rightarrow SS_1$$
$$S \rightarrow \varepsilon$$

The production $S \rightarrow SS_1 \mid \varepsilon$ will generate a string w^* where $w \in L_1$.

The grammar G can be written as :

$$G = (V_1, T_1, P_1 \cup \{ S \rightarrow SS_1 \mid \varepsilon \}, S)$$

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 :

$$S \rightarrow AB$$
$$A \rightarrow aAb \mid \varepsilon$$
$$B \rightarrow cB \mid \varepsilon$$

The language L_2 is a CFL with set of productions given below :

$$S \rightarrow AB$$
$$A \rightarrow aA \mid \varepsilon$$
$$B \rightarrow bBc \mid \varepsilon$$

- 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.

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 $\left(L_1' \cup L_2' \right)'$, 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. 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 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 :

$\delta((q_1, q_2), u, \beta) = ((p_1, p_2), y)$[transition for M]

if and only if

$\delta_1 (q_1, u, \beta) = (p_1, y)$ [transition for M_1]

and $(q_2, u) \underset{M_2}{\overset{*}{\Rightarrow}} (p_2, \varepsilon)$

- 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 times, it requires $\mid u \mid$ 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).

$$L(M) = L(M_1) \cap L(M_2)$$

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 $A \to \alpha$ is a production in P then

$A \to \alpha^R$ is a production in P^R.

Q. 7(a) What do you mean by NP-problems ? Justify that Travelling Salesman problem is NP problem. *(Ans. : Refer section 7.5.1 and Example 7.7.2)* **(Chap. 7, 8 Marks)**

Ans. :

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.
2. 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 $P \subset NP$.

Travelling salesman problem

Given a set of cities $C = \{C_1, C_2, ..., C_n\}$ with $n > 1$ and a function d which assigns to each pair of cites (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.

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.

Q. 7(b) Define Undecidability. Let $HALT_{TM}$ = { <M, w> where M is a TM and M halts on input w} Prove that $HALT_{TM}$ is undecidable.
(Ans. : Refer section 7.4.1)
(Chap. 7, 8 Marks)

Ans. :

Halting Problem of a Turing Machine

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 Chapter 6.

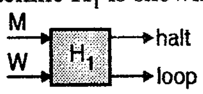
- Unsolvability of halting problem of a Turing machine can be proved through the method of contradiction.

Step 1 : Let us assume that the halting problem of a Turing machine is solvable.

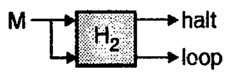
There exists a machine H_1(say). H_1 takes two inputs :

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.

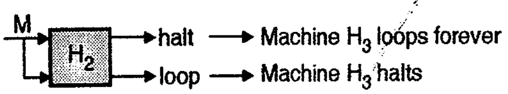


Step 2 : Let us revise the machine H_1 as H_2 to take M as both inputs and H_2 should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.



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.



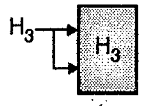
H_3 will do the opposite of the output of H_2.

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.

Q. 8(a) Define and explain Recursive and Recursively enumerable languages.
(Ans. : Refer section 7.1) **(Chap. 7, 8 Marks)**

Ans. :

Recursively Enumerable and Recursive Language

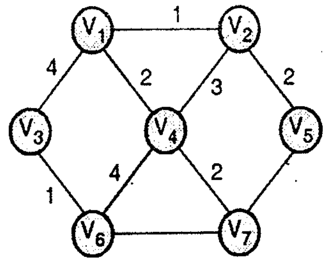
- There is a difference between recursively enumerable (Turing Acceptable) and recursive (Turing Decidable) language.
- Following statements are equivalent :
  1. The language L is **Turing acceptable.**
  2. The language L is **recursively enumerable.**
- Following statements are equivalent
  1. The 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.

Q. 8(b) What is a Kruskals's Algorithm ? How can we solve this problem using Turing Machine ?
(Ans. : Refer sections 7.6.4 and 7.6.5) **(Chap. 7, 8 Marks)**

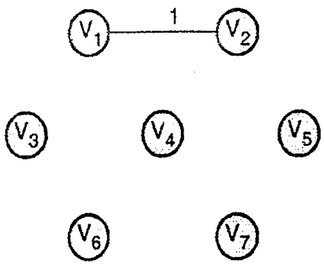
Ans. :

Kruskal's Algorithm

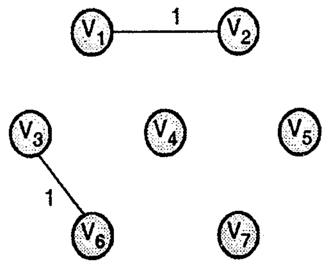
It is another method for finding the minimum cost spanning tree of the given graph. In Kruskal's algorithm, edges are added to the spanning tree in increasing order of cost. If any selected edge e forms a cycle in the spanning tree, it is discarded. Fig. 1-Q. 8(b) shows the sequence in which the edges are added to the spanning tree.



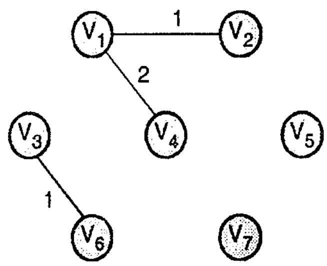
(a) Given graph G



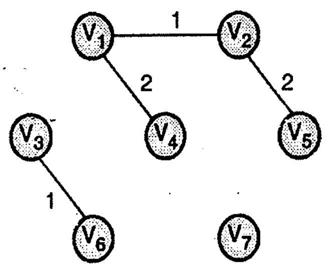
(b) Spanning tree, edge (V_1, V_2) is added



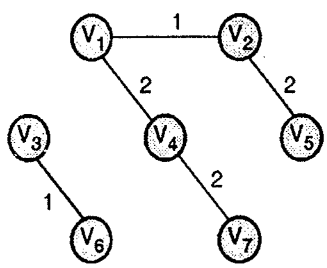
(c) Spanning tree, edge (V_3, V_6) is added



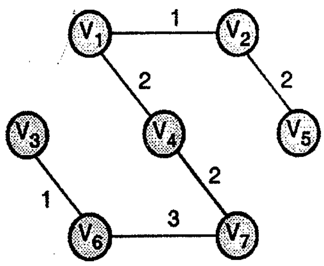
(d) Spanning tree, edge (V_1, V_4) is added



(e) Spanning tree, edge (V_2, V_5) is added



(f) Spanning tree, edge (V_4, V_7) is added



(g) Spanning tree, edge (V_6, V_7) is added

Fig. 1-Q. 8(b) : 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 $e_1, e_2, \ldots e_k$.
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,

   $$V_T = V \text{ and } E_T = \{\ \}$$

3. for every edge e_i in $(e_1, e_2, \ldots, e_k)$

   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.

Kruskal's Algorithm using a Turing Machine

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 it's 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.

❑❑❑