Syllabus

Mumbai University

Revised syllabus (Rev-2016) from Academic Year 2017-18

Automata Theory

Course Code	Course Name	Theory	Practical	Tutorial	Theory	Oral and Practicals	Tutorial	Total
ITC405	Automata Theory	03	—	01	03	—	01	04

Course Code	Course Name	Examination Scheme				Term Work	Oral and Practical	Oral	Total
		Internal Assessment			End Sem. Exam				
		Test 1	Test 2	Avg. of two Tests					
ITC405	Automata Theory	20	20	20	80	—	—	—	100

$ 3 hours shown as theory to be taken class wise and 1 hour to be taken tutorial as batch wise

Course Objectives : Students will try

1. To learn fundamentals of Regular and Context Free Grammars and Languages
2. To understand the relation between Regular Language and Finite Automata and machines.
3. To learn how to design Automata's and machines as Acceptors, Verifiers and Translators.
4. To understand the relation between Contexts free Languages, PDA and TM.
5. To learn how to design PDA as acceptor and TM as Calculators.
6. To learn how to co-relate Automata's with Programs and Functions.

Course Outcomes : The students will be able to :

1. Understand, design, construct, analyze and interpret Regular languages, Expression and Grammars.
2. Design different types of Finite Automata and Machines as Acceptor, Verifier and Translator.
3. Understand, design, analyze and interpret Context Free languages, Expression and Grammars.
4. Design different types of Push down Automata as Simple Parser.
5. Design different types of Turing Machines as Acceptor, Verifier, Translator and Basic computing machine.
6. Compare, understand and analyze different languages, grammars, Automata and Machines and appreciate their power and convert Automata to Programs and Functions

Prerequisite : Basic Mathematical Fundamentals: Sets, Logic, Relations, Functions.

Detailed syllabus

Sr. No.	Module	Detailed Content	Hours	Mapping
I	Introduction and Regular Languages	Languages : Alphabets and Strings. Regular Languages : Regular Expressions, Regular Languages, Regular Grammars, RL and LL grammars, Closure properties. **(Refer Chapters 1, 3 and 5)**	06	CO1

Scanned by CamScanner

Sr. No.	Module	Detailed Content	Hours	Mapping
II	Finite Automata and machines	Finite Automata : FA as language acceptor or verifier, NFA (with and without ε) , DFA, RE to NFA, NFA to DFA, Reduced DFA , NFA-DFA equivalence, FA to RE. Finite State Machines : m/c with output Moore and Mealy machines. M/c as translators. Melay and Moore m/c conversion. **(Refer Chapters 2 and 3)**	09	CO2
III	Context Free Grammars	Context Free Languages : CFG, Leftmost and Rightmost derivations, Ambiguity, Simplification and Normalization (CNF) and Chomskey Hierarchy (Types 0 to 3) **(Refer Chapter 4)**	08	CO3
IV	Push Down Automata	Push Down Automata: Deterministic (single stack) PDA, Equivalence between PDA and CFG. **(Refer Chapter 6)**	05	CO4
V	Turing Machine	Turing Machine : Deterministic TM, Multi-track and Multi-tape TMs, concept of UTM and idea of system program. Issue and concept of Halting Problem **(Refer Chapter 7)**	07	CO5
VI	Applications of Automata	1. Power and Limitations of Regular and Context Free Grammars and Machines 2. Designing Functions : FA : Acceptor and Verifier. FSM : Translator PDA : Simple Parser for WF parenthesis, palindromes etc. TM : Basic bit wise calculator (+ /- /AND/OR) and Translator **(Refer Chapter 8)**	04	CO2 CO4 CO5 CO6

❑❑❑

Scanned by CamScanner

Chapter 1 : Introduction 1-1 to 1-3

✓ **Syllabus Topic :** Alphabets and Strings 1-1
1.1 Languages 1-1
1.1.1 Kleene Closure 1-2
1.1.2 Recursive Definition of a Language 1-2
1.2 Natural and Formal Language 1-3

Chapter 2 : Introduction to Finite Automata and Machines 2-1 to 2-57

✓ **Syllabus Topic :** DFA 2-1
2.1 Deterministic Finite Automata (DFA) 2-1
2.1.1 Definition of a DFA 2-1
✓ **Syllabus Topic :** FA as Language Acceptor or Verifier 2-1
2.1.2 Representation of a DFA 2-1
2.1.3 Designing a DFA 2-2
2.1.4 Examples on DFA 2-4
2.1.4(a) Examples on Counting of Symbols 2-4
2.1.4(b) Examples on Substring 2-7
2.1.4(c) Examples of Divisibility 2-13
2.1.5 Language of DFA 2-15
2.1.6 Equivalence of DFAs 2-16
2.1.7 Closure Property of Language Accepted by a DFA 2-17
2.1.7(a) Union, Intersection, Difference 2-18
2.1.7(b) Complementation 2-19
2.1.8 Minimization of DFA 2-20
2.1.8(a) Algorithm for Minimization DFA's 2-20
✓ **Syllabus Topic :** NFA 2-23
2.2 Non-deterministic Finite Automata 2-23
2.2.1 Definition of NFA 2-23
2.2.2 Processing of a String by NFA 2-24
✓ **Syllabus Topic :** NFA to DFA 2-27
2.2.3 NFA to DFA Conversion 2-27
✓ **Syllabus Topic :** NFA with ε 2-38
2.2.4(a) Equivalence of ε-NFA and NFA 2-38
2.2.4(b) The Formal Notation for an ε-NFA 2-38
2.2.4(c) ε-Closures 2-39
2.2.4(d) ε-NFA to DFA 2-40
2.2.5 Difference between NFA and DFA 2-44
2.3 Finite Automata as Output Devices 2-44
2.3.1 A Sample Mealy Machine 2-44
2.3.2 Formal Definition of a Mealy Machine 2-45
2.3.3 A Sample Moore Machine 2-45
2.3.4 Formal Definition of a Moore Machine 2-45
✓ **Syllabus Topic :** Mealy and Moore Machine Conversion 2-50
2.3.5 Conversion of a Mealy Machine into a Moore Machine 2-50
2.3.6 Conversion of a Moore Machine into a Mealy Machine 2-54
2.4 Minimization of a Mealy Machine 2-56
2.5 Exam Pack (University Questions) 2-57

Chapter 3 : Regular Expressions and Languages 3-1 to 3-28

✓ **Syllabus Topic :** Regular Expression, Regular Languages 3-1
3.1 Introduction 3-1
3.2 Finite Automata Representing a Regular Expression 3-3
3.2.1 Composite Finite State Automata 3-3
✓ **Syllabus Topic :** RE to NFA 3-5
3.3 Determination of Regular Expression 3-5
3.3.1 Language Generated by a Regular Expression 3-5
3.3.2 Basic Properties of Regular Expressions 3-5
✓ **Syllabus Topic :** FA to RE 3-13
3.4 DFA to Regular Expression 3-13
3.4.1 State/Loop Elimination Process 3-13
3.4.1(a) A Generic One State Machine 3-14
3.4.1(b) A Generic Two State Machine 3-14
3.4.2 Arden's Theorem 3-22
3.4.2(a) Application of Arden's Theorem 3-23
3.5 FA Limitations **(Dec. 2016)** 3-25
✓ **Syllabus Topic :** Closure Properties 3-25
3.6 Closure Properties of Regular Language **(May 2014)** 3-25
3.6.1 Regular Language is Closed under Union 3-25
3.6.2 Regular Language is Closed under Concatenation 3-26
3.6.3 Regular Language is Closed under Kleene Star 3-26
3.6.4 Regular Language is Closed under Complementation 3-26
3.6.5 Regular Language is Closed under Intersection 3-26
3.6.6 Regular Languages are Closed under Difference 3-26
3.6.7 Regular Languages are Closed under Reversal 3-26
3.7 Application of RE 3-28
3.7.1 R.E. in Unix 3-28
3.7.2 Lexical Analysis 3-28
3.8 Exam Pack (University Questions) 3-28

Chapter 4 : Context Free Grammars and Languages 4-1 to 4-36

4.1 An Example to Explain Grammar 4-1
✓ **Syllabus Topic :** CFG 4-2
4.2 Context Free Grammar **(Dec. 2016, May 2017)** 4-2
4.2.1 Notations 4-2
4.2.2 The Language of a Grammar 4-3

Scanned by CamScanner

✓ **Syllabus Topic :** Leftmost and Rightmost Derivations 4-3
4.2.3 Writing Grammar for a Language 4-7
4.2.3(a) Union Rule for Grammar 4-10
4.2.3(b) Concatenation Rule for Grammar 4-10
✓ **Syllabus Topic :** Ambiguity 4-17
4.3 Ambiguous Grammar 4-17
✓ **Syllabus Topic :** Simplification 4-23
4.4 Simplification of CFG 4-23
4.4.1 Elimination of Useless Symbols 4-23
4.4.2 Elimination of ε-productions 4-26
4.4.3 Elimination of Unit Productions 4-27
✓ **Syllabus Topic :** Normalization 4-30
4.5 Normal Forms for CFG **(Dec. 2016)** 4-30
✓ **Syllabus Topic :** CNF 4-30
4.5.1 Chomsky Normal Form (CNF) 4-30
4.5.1(a) Algorithm for CFG to CNF Conversion **(May 2014)** 4-30
✓ **Syllabus Topic :** Chomsky Hierarchy (Types 0 to 3) 4-33
4.6 Chomsky Classification for Grammar **(May 2014, Dec. 2014, May 2017)** 4-33
4.6.1 Type 3 or Regular Grammar 4-34
4.6.2 Type 2 or Context Free Grammar 4-34
4.6.3 Type 1 or Context Sensitive Grammar 4-34
4.6.4 Type 0 or Unrestricted Grammar 4-34
4.7 Properties of Context-Free Languages 4-34
4.7.1 Closure Properties 4-34
4.7.1(a) CFL is Closed under Union 4-34
4.7.1(b) CFL is Closed under Concatenation 4-35
4.7.1(c) CFL is Closed under Kleene Star 4-35
4.7.1(d) CFL is not Closed under Intersection 4-35
4.7.1(e) CFL is not Closed under Complementation 4-35
4.7.1(f) Intersection of CFL and RL 4-35
4.7.1(g) CFL is Closed under Reversal 4-36
4.7.2 Algorithmic Properties (Decision Properties) 4-36
4.8 Exam Pack (University Questions) 4-36

Chapter 5 : Regular Languages **5-1 to 5-12**

✓ **Syllabus Topic :** Regular Grammars, RL and LL Grammar 5-1
5.1 Regular Grammar 5-1
5.2 DFA to Right-Linear Regular Grammar 5-1
5.2.1 Right Linear Grammar to DFA 5-2
5.3 DFA to Left-Linear Grammar 5-4
5.3.1 Left Linear Grammar to DFA 5-5
5.4 Right Linear Grammar to Left Linear Grammar 5-6
5.5 Left Linear Grammar to Right Linear Grammar 5-7

Chapter 6 : Push Down Automata **6-1 to 6-25**

✓ **Syllabus Topic :** Deterministic (Single Stack) PDA 6-1
6.1 Introduction to Push Down Automata (PDA) 6-1
6.2 The Formal Definition of PDA **(May 2014, May 2015)** 6-2
6.3 Instantaneous Description of a PDA 6-3
6.4 The Language of a PDA 6-4
6.4.1 Acceptance by Final State 6-4
6.4.2 Acceptance by Empty Stack 6-4
6.5 Non-deterministic PDA (NPDA) 6-11
6.6 Push Down Automata and Context Free Language 6-13
6.6.1 Construction of PDA from CFG 6-13
✓ **Syllabus Topic :** Equivalence between PDA and CFG 6-15
6.6.2 Construction of CFG from PDA 6-16
6.7 Deterministic Push Down Automata (DPDA) 6-22
6.7.1 Regular Language and DPDA 6-22
6.8 Application of PDA 6-23
6.9 Parsing 6-23
6.9.1 Top-Down Parsing 6-23
6.9.2 Bottom-Up Parsing 6-24
6.10 Exam Pack (University Questions) 6-25

Chapter 7 : Turing Machine **7-1 to 7-31**

✓ **Syllabus Topic :** Deterministic TM 7-1
7.1 Introduction to Turing Machine **(Dec. 2014)** 7-1
7.2 The Formal Definition of Turing Machine 7-2
7.2.1 A String Accepted by TM 7-3
7.2.2 Instantaneous Descriptions for Turing Machines 7-4
7.3 Turing Machines as Computer of Functions 7-13
7.4 Extension of Turing Machine **(May 2014, May 2015)** 7-24
7.4.1 Two-way Infinite Turing Machine 7-24
7.4.2 A Turing Machine with Multiple Heads 7-24
✓ **Syllabus Topic :** Multi-track and Multi-tape TMs 7-24
7.4.3 Multi-Tape Turing Machine 7-24
7.4.4 Non-deterministic Turing Machine 7-25
✓ **Syllabus Topic :** Concept of UTM and Idea of System Program 7-26
7.5 Universal Turing Machine 7-26
7.5.1 Church-Turing Hypothesis 7-27
7.5.2 Power of Various Machines 7-27
7.6 Un-decidability 7-27
✓ **Syllabus Topic :** Issue and Concept of Halting Problem 7-28
7.6.1 Halting Problem of a Turing Machine **(Dec. 2014, May 2015, Dec. 2016)** 7-28

Scanned by CamScanner

7.6.2 Un-decidability of Post Correspondence Problem **(May 2014, Dec. 2014, Dec. 2016)** 7-29
7.7 Exam Pack (University Questions) 7-30

Chapter 8 : Applications of Automata 8-1 to 8-3

✓ **Syllabus Topic :** Power and Limitation of Regular and Context Free Grammars and Machines 8-1
8.1 Power and Limitation of Regular and Context Free Grammars and Machines 8-1
✓ **Syllabus Topic :** Designing Functions 8-1
8.2 Designing Function 8-1
✓ **Syllabus Topic :** FA as Acceptor and Verifier 8-1
8.2.1 FA as Acceptor and Verifier 8-1
✓ **Syllabus Topic :** FSM : Translator 8-2
8.2.2 FM as Translator 8-2
✓ **Syllabus Topic :** Simple Parser for WF Parenthesis, Palindromes etc. 8-2
8.2.3 Simple Parser using PDA 8-2
✓ **Syllabus Topic :** TM : Basic Bitwise Calculator (+ /–/AND/OR) and Translator 8-2
8.2.4 TM for Bitwise Operation 8-2

❑❑❑

Scanned by CamScanner

CHAPTER 1

Introduction

Syllabus Topics

Languages : Alphabets and Strings.

Syllabus Topic : Alphabets and Strings

1.1 Languages

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,

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

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

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

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

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

For example,

if ω = xyz

then ω^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,

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

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

For example,

if ω = abb

then $\omega^0 = \varepsilon$

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

Scanned by CamScanner

- 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 :

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

Some examples of language are :

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

1.1.1 Kleene Closure

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

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

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 1.1.1

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^*.

1.1.2 Recursive Definition of a Language

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

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

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

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

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 1.1.2

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.

Scanned by CamScanner

Deriving $5x^2 + 7x$

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

1.2 Natural and Formal Language

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

A formal language is based on rules which are explicitly stated.

1. List of symbols is explicitly stated.
2. Rules for forming a string (word) are explicitly stated.
3. Rules for forming a sentence are explicitly stated.

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

❑❑❑

Scanned by CamScanner

CHAPTER 2

Introduction to Finite Automata and Machines

Syllabus Topics

Finite Automata : FA as language acceptor or verifier, NFA (with and without ε), DFA, NFA to DFA, Reduced DFA , NFA-DFA equivalence.

Finite State Machines : m/c with output Moore and Mealy machines. M/c as translators. Mealy and Moore m/c conversion.

Syllabus Topic : DFA

2.1 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.

2.1.1 Definition of a DFA

A deterministic finite automata is a quintuple.

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

Q is a set of states.

Σ is a set of alphabet.

$q_0 \in Q$ is the initial state,

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

Syllabus Topic : FA as Language Acceptor or Verifier

2.1.2 Representation of a DFA

Let the machine M be a deterministic finite automata.

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

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

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

q_0 is the starting state

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

and δ is the transition function as given below :

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

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

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

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

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

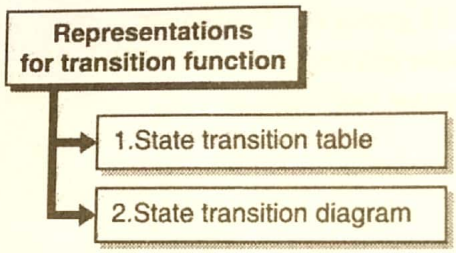


Fig. C2.1 : Representation of transition funtion

1. State transition table

The Fig. 2.1.1 shows the state transition table of the transition function discussed in this section.

	0	1 ←	
→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. 2.1.1 : State transition table

- In state transition table, there is a row for every state $q_i \in Q$.
- In state transition table, there is a column for every alphabet $A_i \in \Sigma$.

Scanned by CamScanner

- Starting state is marked as '→'.
- Final state is marked as '*'.

2. State transition diagram

The Fig. 2.1.2 shows the state transition diagram of the transition function discussed in this section.

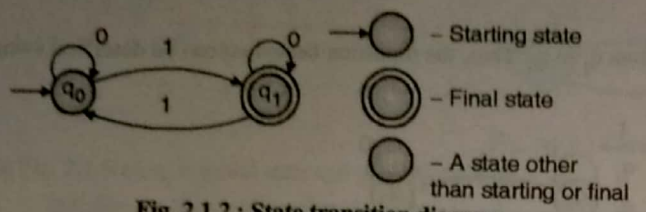


Fig. 2.1.2 : State transition diagram

Language of the above DFA

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

Thus we can conclude that the DFA accepts a string if and only if the number of 1's is odd. The language L accepted by M, the given DFA, is the set of all strings having odd number of 1's.

Or,

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

Simulating the functioning of the above DFA for a given input

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

$(q_0, 0010110) \overset{\delta}{\Rightarrow} (q_0, 010110)$ [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.

2.1.3 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 :

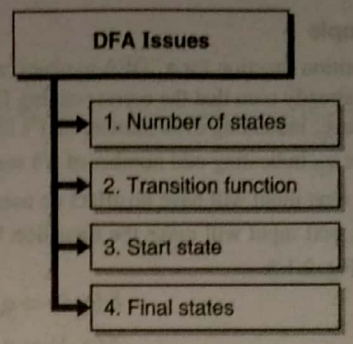


Fig. C2.2 : Issues to be considered while designing DFA

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 is even or odd. It does not have to count number of 1's. Let us look at a binary number given below :

Binary Number →	0	1	1	0	1	1	0	0	1	1
Number of 1's →	even	odd	even	even	odd	even	even	even	odd	even

- Machine will have two states :
  (i) One corresponding to even number of 1's, so far.
  (ii) One corresponding to odd number of 1's, so far.
- These two states are necessary and sufficient to conclude whether a binary number contains even number of 1's.

2. DFA to check whether a string over alphabets (a, b) contains a substring abb :

DFA has to search for substring abb in input stream. Input string is not stored in a memory variable. Input is made one symbol at a time. As the substring is of length 3, it may be necessary to remember the preceding two symbols.

- Preceding symbol is a. (a is the first symbol in abb).
- Preceding two symbols are ab. (ab constitutes first two symbols of abb).
- DFA has already seen the substring abb in the input string. Once the substring abb is found in input string, this status will not change irrespective of what follows thereafter in the input string.
- Preceding symbols are neither a nor ab, and abb has not come earlier.

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

2. Transition function

Transition function gives the next-state, depending on :

1. Current state 2. Current input

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

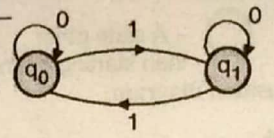
Scanned by CamScanner

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.
   - 0 as next input will have no effect on number of 1's.
   - 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. 2.1.3.

$\delta(q_0,0) \Rightarrow q_0$
$\delta(q_0,1) \Rightarrow q_1$
$\delta(q_1,0) \Rightarrow q_1$
$\delta(q_1,1) \Rightarrow q_0$

	0	1
q_0	q_0	q_1
q_1	q_1	q_0



(a) Tabular form (b) Transition table (c) Transition diagram

Fig. 2.1.3 : Transition behaviour

2. Give transition function for a "DFA to check whether a string over alphabets (a, b) contains a substring abb".
   We have already seen that the corresponding DFA will have four states.
   1. State q_1, preceding symbol is a.
   2. State q_2, preceding two symbols are ab.
   3. State q_3, substring abb has already been seen in input string.
   4. State q_0, situations q_1, q_2, or q_3 are not there.

   State of the DFA after every input symbol for a sample input data is shown in Fig. 2.1.4.

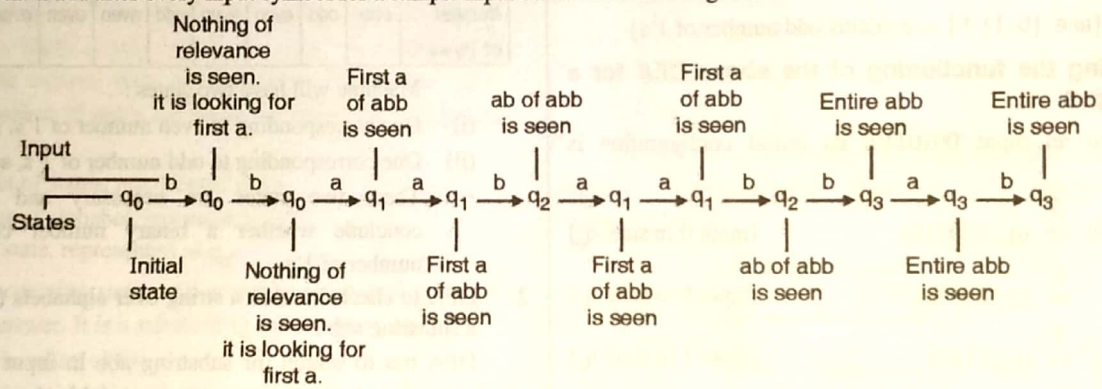


Fig. 2.1.4 : States of DFA

- 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.

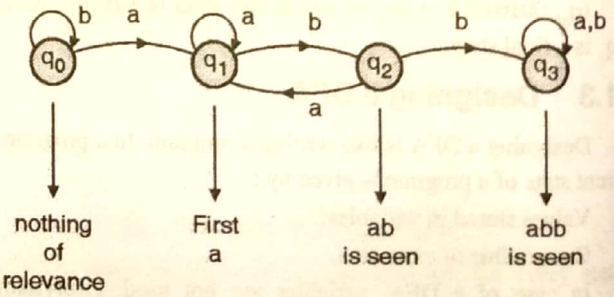
Transition behaviour is described using Fig. 2.1.5.

$\delta(q_0,b) \Rightarrow q_0$
$\delta(q_0,a) \Rightarrow q_1$
$\delta(q_1,a) \Rightarrow q_1$
$\delta(q_1,b) \Rightarrow q_2$
$\delta(q_2,a) \Rightarrow q_1$
$\delta(q_2,b) \Rightarrow q_3$
$\delta(q_3,a) \Rightarrow q_3$
$\delta(q_3,b) \Rightarrow q_3$

(a) 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. 2.1.5 : Transition behaviour

Scanned by CamScanner

3. Starting state and final states

In Fig. 2.1.3(c), q_0 is both the initial state and final state as :

1. Zero number of 1's is even number of 1's.
2. Machine will have "yes" answer for even number of 1's. q_0 is final state. Machine in Fig. 2.1.3(c) is redrawn in Fig. 2.1.6 with initial and final states marked.

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

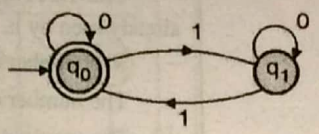


Fig. 2.1.6 : Final DFA for Fig. 2.1.3

In Fig. 2.1.5(c), q_0 is initial state and q_3 is the final state.

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

Machine in Fig. 2.1.5 is redrawn in Fig. 2.1.7 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

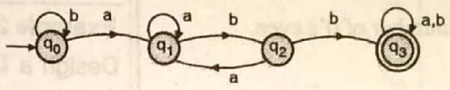


Fig. 2.1.7 : Final DFA for Fig. 2.1.5(c)

2.1.4 Examples on DFA

2.1.4(a) Examples on Counting of Symbols

Example 2.1.1

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

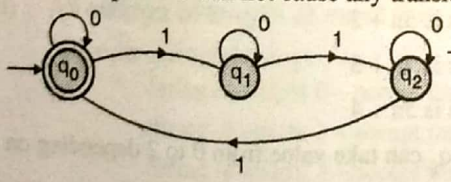
An input of 1 will cause a transition from :

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.



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

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

Fig. Ex. 2.1.1 : Final DFA for Example 2.1.1(a)

– 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.

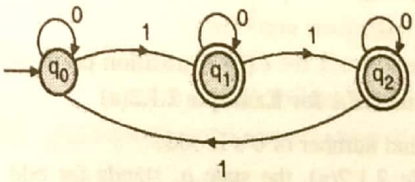
Thus, $L = [\omega \in \{0, 1\}^* \mid \text{no. of 1's in } \omega \text{ is multiple of 3}]$

Complement of L is given by :

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

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

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′.



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

(c) Transition diagram **(d) Transition table**

Fig. Ex. 2.1.1 : Final DFA for Example 2.1.1(b)

Example 2.1.2

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 :

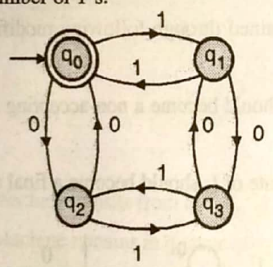
(a) Number of 1's is even and number of 0's is even.

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

Scanned by CamScanner

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.



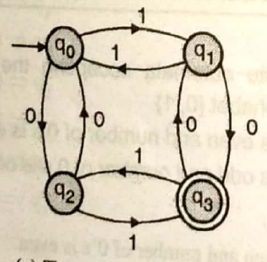
	0	1
$\rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

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

Fig. Ex. 2.1.2 : Final DFA for Example 2.1.2(a)

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

In solution of Example 2.1.2(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

(c) Transition diagram **(d) Transition table**

Fig. Ex. 2.1.2 : Final DFA for Example 2.1.2(b)

Example 2.1.3

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

Solution :

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

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

The required DFA is given in Fig. Ex. 2.1.3.

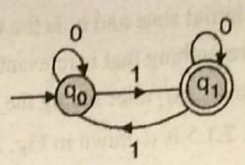


Fig. Ex. 2.1.3

Example 2.1.4

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

Solution :

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

Case 1 – 5n

Case 2 – 5n + 1

Case 3 – 5n + 2

Case 4 – 5n + 3

Case 5 – 5n + 4

Number of 0's should be divisible by 5.

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

Case 1 – 3m

Case 2 – 3m + 1

Case 3 – 3m + 2

Number of 1's is divisible by 3.

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

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

q_{0j} – number of 0's is 5n

q_{1j} – number of 0's is 5n + 1

q_{2j} – number of 0's is 5n + 2

q_{3j} – number of 0's is 5n + 3

q_{4j} – number of 0's is 5n + 4

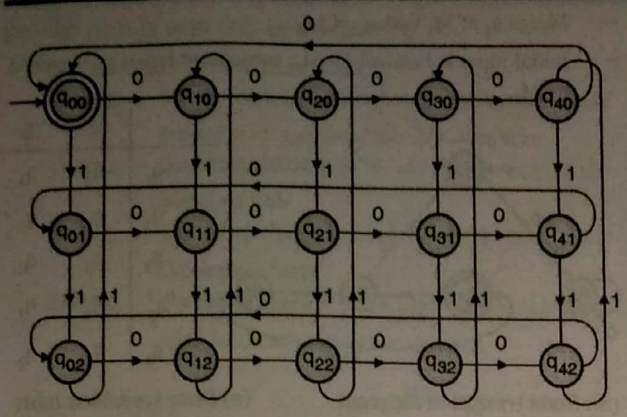
Similarly, in state q_{ij} can take value from 0 to 2 depending on number of 1's seen so far.

q_{i0} – number of 1's is 3m

q_{i1} – number of 1's is 3m + 1

q_{i2} – number of 1's is 3m + 2

Scanned by CamScanner



(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 table

Fig. Ex. 2.1.4 : Final DFA for Example 2.1.4

Example 2.1.5

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.

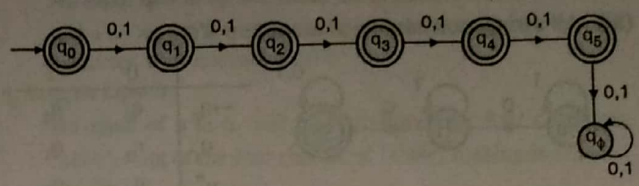
String of length 2 – accept through q_2.

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.



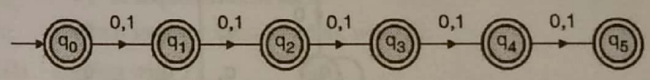
(a) State transition diagram

	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. 2.1.5 : Final DFA for Example 2.1.5(i) with explicit failure state

It is not necessary to mention the failure/dead state explicitly. If a transition is not mentioned, it is taken as a failure transition. Thus, the solution can be given with out explicit failure state.



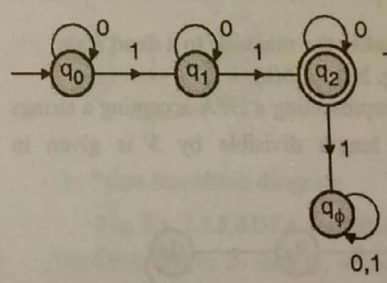
(c) State transition diagram

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

(d) State transition table

Fig. Ex. 2.1.5 : Final DFA for Example 2.1.5(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

(e) State transition diagram (f) State transition table

Fig. Ex. 2.1.5 : Final DFA for Example 2.1.5(ii)

– First 1 takes the machine from q_0 to q_1.

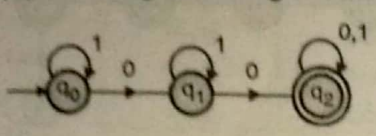
– Second 1 takes the machine from q_1 to q_2.

Scanned by CamScanner

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

(g) State transition diagram **(h) State transition table**

Fig. Ex. 2.1.5 : Final DFA for Example 2.1.5(iii)

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

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

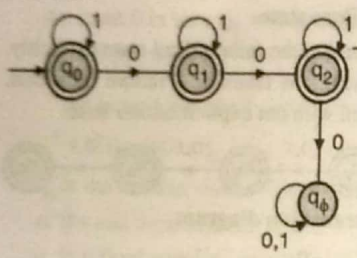
These are the valid strings :

A string not containing 0 – accept through q_0.

A string containing one 0 – accept through q_1.

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

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



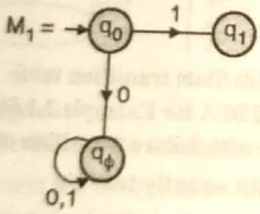
	0	1
$\rightarrow q_0^*$	q_1	q_0
q_1^*	q_2	q_1
q_2^*	q_ϕ	q_2
q_ϕ	q_ϕ	q_ϕ

(i) State transition diagram **(j) State transition table**

Fig. Ex. 2.1.5 : Final DFA for Example 2.1.5(iv)

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

Step 1 : A machine M_1, representing a DFA accepting a string starting with 1 is given in Fig. Ex. 2.1.5(k) :



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

Fig. Ex. 2.1.5(k)

Step 2 : A machine M_2, representing a DFA accepting a strings starting having length divisible by 3 is given in Fig. Ex. 2.1.5(*l*) :

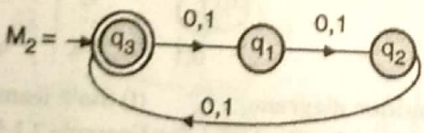
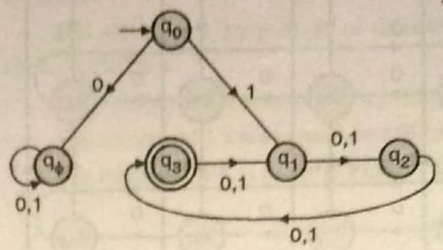


Fig. Ex. 2.1.5 (*l*)

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

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



	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

(m) State transition diagram **(n) State transition table**

Fig. Ex. 2.1.5 : Final DFA for Example 2.1.5(v)

2.1.4(b) Examples on Substring

Example 2.1.6

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

(a) All strings starting with abb.

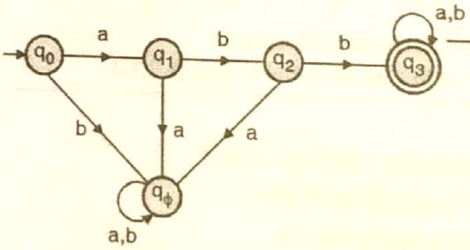
(b) All strings with abb as a substring i.e., abb anywhere in the string.

(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.

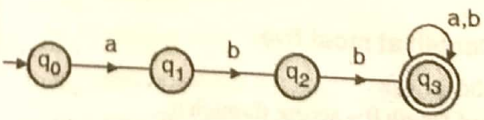


	a	b
$\rightarrow q_0$	q_1	q_ϕ
q_1	q_ϕ	q_2
q_2	q_ϕ	q_3
q_3^*	q_3	q_3
q_ϕ	q_ϕ	q_ϕ

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

Fig. Ex. 2.1.6 : Final DFA for Example 2.1.6(a)

A DFA without explicit failure state is given in Fig. Ex. 2.1.6(a).



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

(c) State transition diagram **(d) State transition table**

Fig. Ex. 2.1.6 : Final DFA for Example 2.1.6(a), without a failure / dead state

Scanned by CamScanner

(b) All strings with abb as a substring

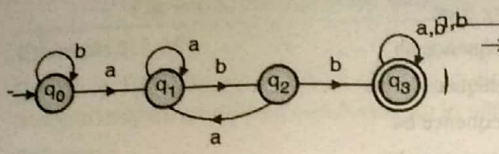
The machine will have fours states :

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

(e) State transition diagram **(f) State transition table**

Fig. Ex. 2.1.6 : Final DFA for Example 2.1.6(b)

q_0 to q_0 on input 'b' :

First character in 'abb' is a.

q_0 to q_1 on input 'a' :

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

q_1 to q_1 on input 'a' :

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

q_1 to q_2 on input 'b' :

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

q_2 to q_1 on input 'a' :

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

q_2 to q_3 on input b :

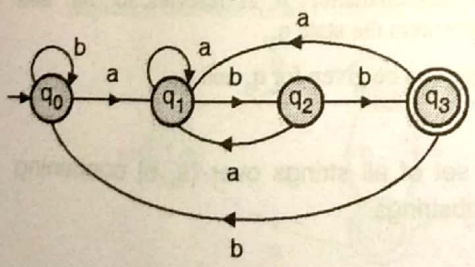
q_3 is for preceding three characters as 'abb'.

q_3 to q_3 on input a or b :

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

(c) All strings ending in abb

This can be seen as an extention of solution which is given in Example 2.1.6(b) as the substring 'abb' should be at the end of the string. Transitions from q_3 should be modified to handle the condition that the string has to end in 'abb'.



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

(g) State transition diagram **(h) State transition table**

Fig. Ex. 2.1.6 : Final DFA for Example 2.1.6(c)

q_3 to q_1 on input a :

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

q_3 to q_0 on input b :

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

Example 2.1.7

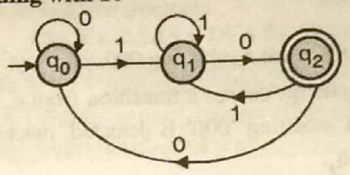
Design DFA for a language of string 0 and 1 that :

I) Ending with 10 II) Ending with 11

III) Ending with 1

Solution :

I) Ending with 10



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

(a) State transition diagram

Fig. Ex. 2.1.7 : DFA for string ending with 10

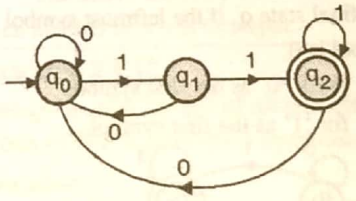
Meaning of various states

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

q_1 : '1' of sequence 10 is seen.

q_2 : Final state for strings ending with 10.

II) Ending with 11



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

(b) State transition diagram

Fig. Ex. 2.1.7 : DFA for string ending with 11

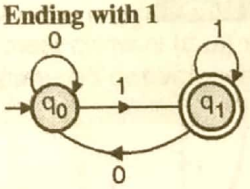
Meaning of various states

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

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

q_2 : Final state of strings ending with '11'

III) Ending with 1



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

(c) State transition diagram

Fig. Ex. 2.1.7 : DFA for string ending with 1

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

Example 2.1.8

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

Scanned by CamScanner

Solution :

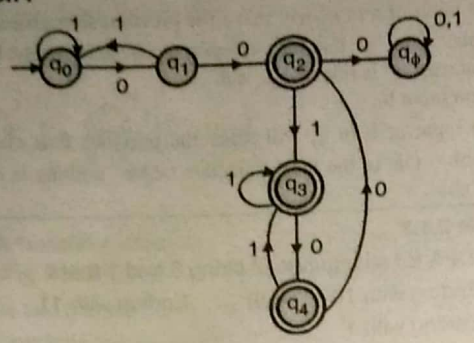


Fig. Ex. 2.1.8 : Final DFA for Example 2.1.8

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

Example 2.1.9

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

Solution :

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

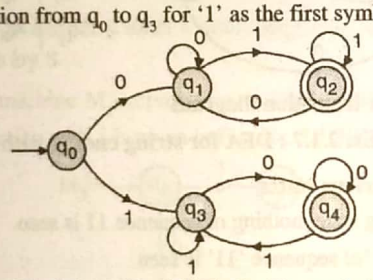
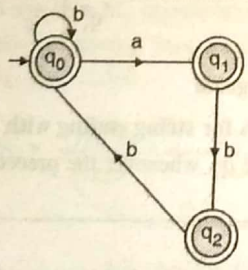


Fig. Ex. 2.1.9 : Final DFA for Example 2.1.9

Example 2.1.10 MU - May 2014, 5 Marks

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. 2.1.10 : Final DFA (without explicit failure state) for Example 2.1.10

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

Example 2.1.11

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

Solution :

Meaning of different states :

$q_0 \rightarrow$ starting state

$q_1 \rightarrow$ a of sequence ab

$q_2 \rightarrow$ ab of sequence ab

$q_3 \rightarrow$ b of sequence ba

$q_4 \rightarrow$ ba of sequence ba

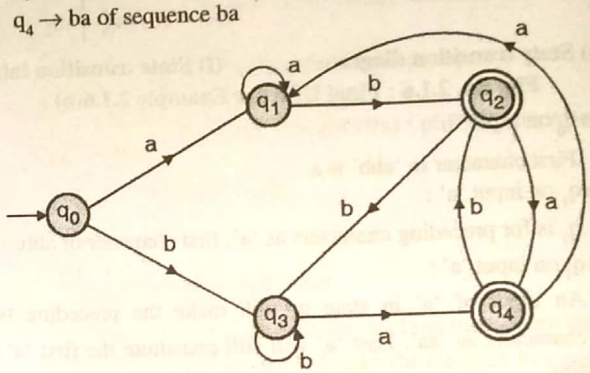


Fig. Ex. 2.1.11 : DFA for Example 2.1.11

Transitions

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

- Input b in q_0 takes the machine to q_3 as the first character 'b' of 'ba' is the preceding character.
- Input 'a' in q_1 makes the preceding two characters as 'aa'. Out of 'aa', only the last character 'a' is relevant to 'ab' and hence the machine requires in q_1.
- Input 'b' in q_1 makes the preceding two characters as 'ab'. Machine enters the state q_2 which stands for previous two characters as ab.
- Input 'a' in q_2 makes the preceding two characters as 'ba'. Machine enters the state q_4 which stands for previous two characters as ba.
- Input b in q_2 makes the preceding two characters as 'bb'. Out of 'bb', only the last character 'b' is relevant to 'ba' and hence the machine enters the state q_3.
- Similar explanation can be given for q_3 and q_4.

Example 2.1.12

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

Solution :

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

Scanned by CamScanner

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

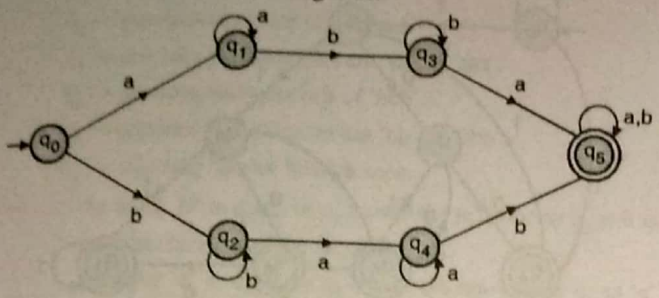


Fig. Ex. 2.1.12 : DFA for Example 2.1.12

Example 2.1.13

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

Solution :

Meaning of different states :

$q_0 \rightarrow$ starting state

$q_1 \rightarrow$ a of sequence aa is seen.

$q_2 \rightarrow$ aa of sequence aa is seen.

$q_3 \rightarrow$ b of sequence bb is seen.

$q_4 \rightarrow$ bb of sequence bb is seen.

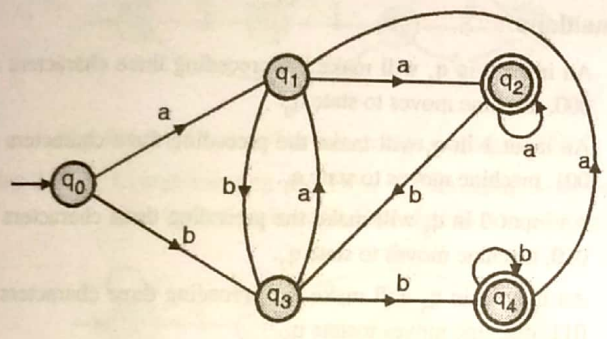


Fig. Ex. 2.1.13

Example 2.1.14

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

Solution :

- Machine enters the failure state q_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 the input sequence contains a and b alternatively.

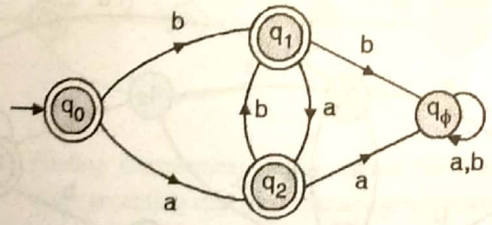
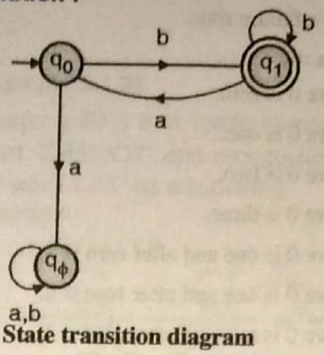


Fig. Ex. 2.1.14 : DFA for Example 2.1.14

Example 2.1.15

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

Solution :



(a) State transition diagram

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

(b) State transition table

Fig. Ex. 2.1.15 : DFA for Example 2.1.15

Example 2.1.16 MU - May 2014, 5 Marks

Design a DFA to accept strings over the alphabet $\Sigma = \{a, b\}$ containing even number of a's.

Solution : The transition diagram of the given DFA is given below

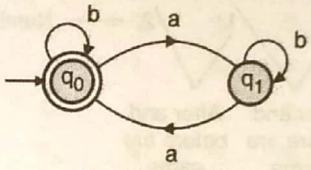


Fig. Ex. 2.1.16

The above machine will enter the accepting state q_0 on seeing even number of a's in the input.

Example 2.1.17

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 :

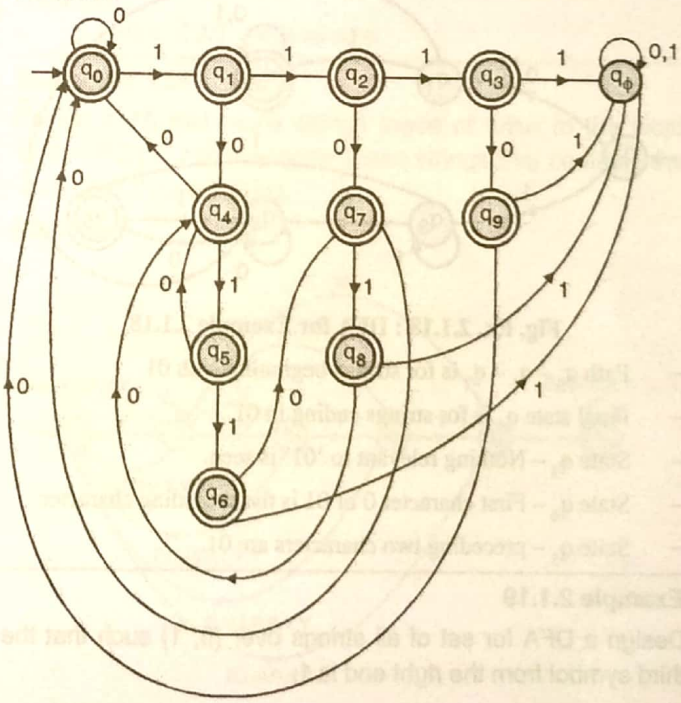


Fig. Ex. 2.1.17 : DFA for Example 2.1.17

Scanned by CamScanner

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

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

If the sum total of number of 1's before 0 and number of 1's after 0 is 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.

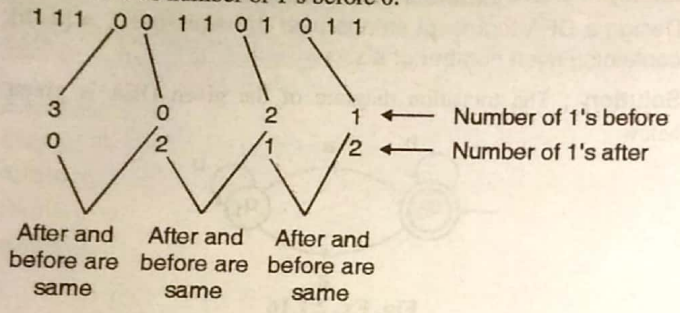


Fig. Ex. 2.1.17(a)

Example 2.1.18

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

Solution :

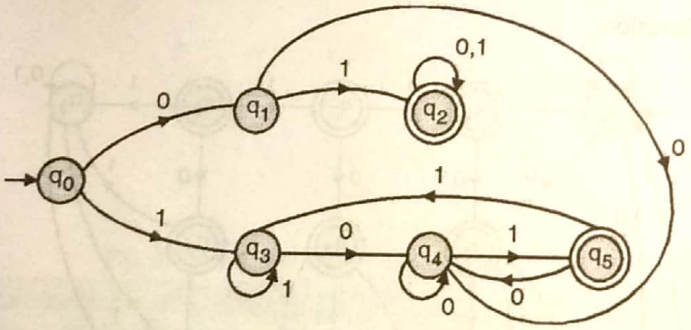


Fig. Ex. 2.1.18 : DFA for Example 2.1.18

- 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 2.1.19

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

Solution :

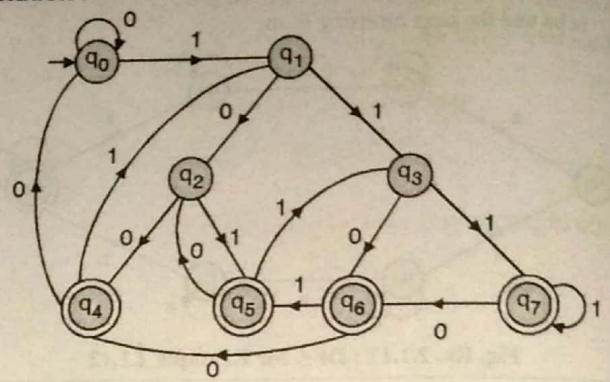


Fig. Ex. 2.1.19 : DFA for Example 2.1.19

Following 4 combinations for the preceding three characters are accepting.

100, 101, 110, 111.

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

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

Transitions

- An input 0 in q_4 will make the preceding three characters as 000, machine moves to state q_0.
- An input 1 in q_4 will make the preceding three characters as 001, machine moves to state q_1.
- An input 0 in q_5 will make the preceding three characters as 010, machine moves to state q_2.
- An input 1 in q_5 will make the preceding three characters as 011, machine moves tostate 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 2.1.20

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

Solution :

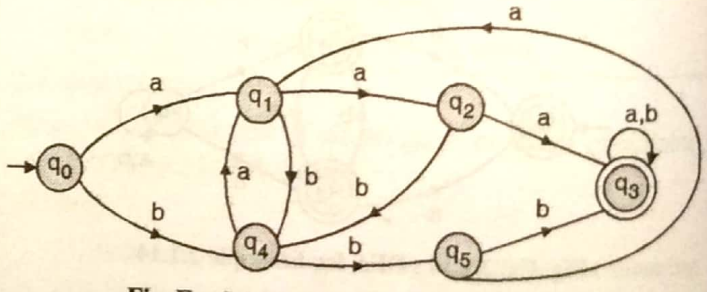


Fig. Ex. 2.1.20 : DFA for Example 2.1.20

Scanned by CamScanner

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 2.1.21

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.

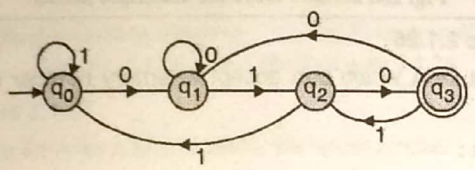


Fig. Ex. 2.1.21(a) : DFA for strings ending in 010

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

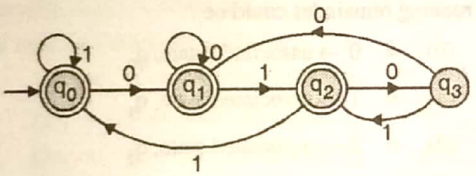


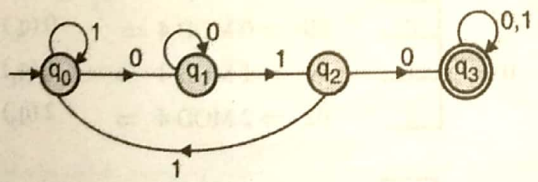
Fig. Ex. 2.1.21(b) : DFA for strings not ending in 010

Example 2.1.22 MU - May 2014, May 2015, 5 / 10 Marks

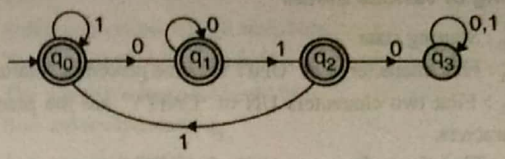
Design a DFA to accept strings over the binary alphabet that do not contain the substring 010

Solution : First we will design a DFA for accepting strings with 010 as a substring and then we will take its complement to obtain the required DFA.

Step 1 : DFA for strings containing the substring 010.



Step 2 : Finding complement of the above DFA by making every accepting state as a non-accepting state and every non-accepting state as an accepting state.



Example 2.1.23

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 :

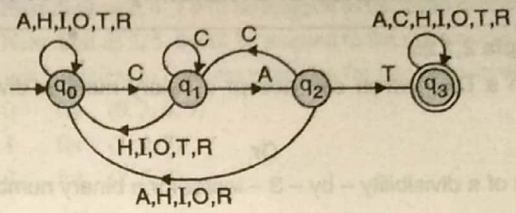


Fig. Ex. 2.1.23(a) :State transition diagram

	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 table

Fig. Ex. 2.1.23 : DFA for Example 2.1.23

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 2.1.24

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

Solution :

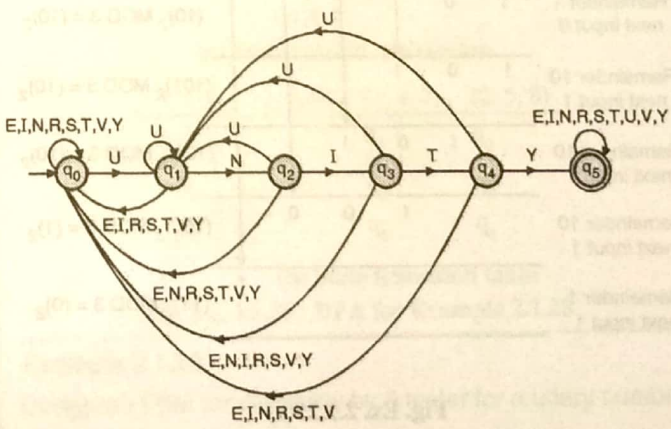


Fig. Ex. 2.1.24

Scanned by CamScanner

Meaning of various states

q_0 : Starting state

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

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

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

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

q_5 : Entire 'UNITY' has been seen.

2.1.4(c) Examples of Divisibility

Example 2.1.25

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 :

  $0 \rightarrow$ associated state, q_0

  $1 \rightarrow$ associated state, q_1

  $2 \rightarrow$ associated state, q_2
- Starting with the most significant bit, input is taken one bit at a time. Running remainder is calculated after every input.

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

Number to be divided : 101101

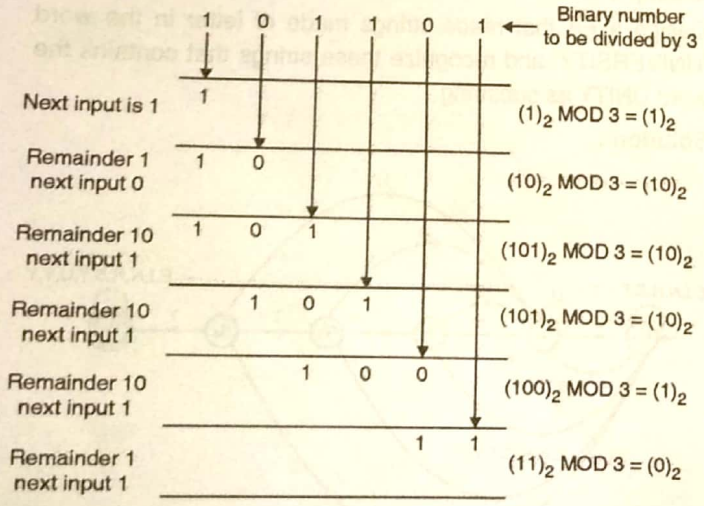
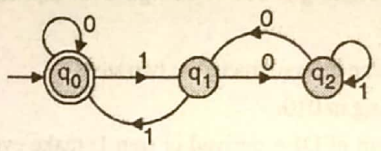


Fig. Ex. 2.1.25(a)

The calculation of next remainder is shown below,

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

(b) State transition diagram **(c) State transition table**

Fig. Ex. 2.1.25 : DFA for Example 2.1.25

Example 2.1.26

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

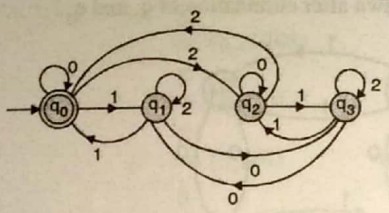
Transition behaviour will depend on the current remainder (current state) and the next input. These two will be used to get the next remainder (next state).

Transition behaviour is explained below :

Current remainder	Next input	Calculation of remainder	Remainder
(q_0) 0	0	00 ⇒ 0 MOD 4 ⇒	0 (q_0)
	1	01 ⇒ 1 MOD 4 ⇒	1 (q_1)
	2	02 ⇒ 2 MOD 4 ⇒	2 (q_2)
(q_1) 1	0	10 ⇒ 3 MOD 4 ⇒	3 (q_3)
	1	11 ⇒ 4 MOD 4 ⇒	0 (q_0)
	2	12 ⇒ 5 MOD 4 ⇒	1 (q_1)

Scanned by CamScanner

Current remainder	Next input	Calculation of remainder	Remainder
	0	20 ⇒ 6 MOD 4 ⇒	2 (q_2)
(q_2) 2	1	21 ⇒ 7 MOD 4 ⇒	3 (q_3)
	2	22 ⇒ 8 MOD 4 ⇒	0 (q_0)
	0	100 ⇒ 9 MOD 4 ⇒	1 (q_1)
(q_2) 10	1	101 ⇒ 10 MOD 4 ⇒	2 (q_2)
	2	102 ⇒ 11 MOD 4 ⇒	3 (q_3)
Ternary		Ternary Decimal	Decimal



	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

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

Fig. Ex. 2.1.26 : DFA for Example 2.1.26

Example 2.1.27

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

Solution :

A ternary system has three alphabets

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

Base of a ternary number is 3.

The running remainder could be :

$(0)_3$ = 0 → associated state, q_0

$(1)_3$ = 1 → associated state, q_1

$(2)_3$ = 2 → associated state, q_2

$(10)_3$ = 3 → associated state, q_3

$(11)_3$ = 4 → associated state, q_4

↑ ↑

Ternary Decimal

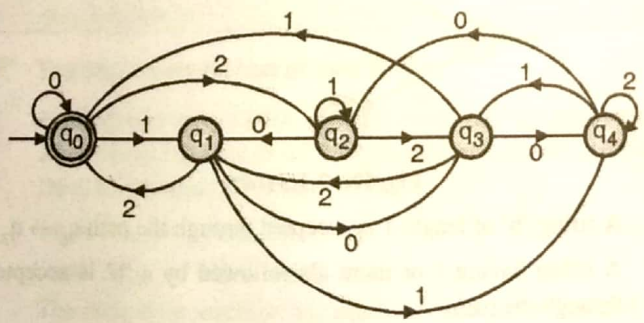


Fig. Ex. 2.1.27

Example 2.1.28

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 → associated state, q_0

1 → associated state, q_1

2 → 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 :

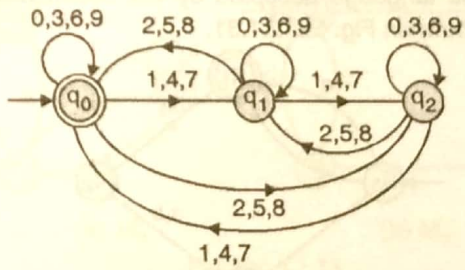
0 for (0, 3, 6, 9)

1 for (1, 4, 7)

2 for (2, 5, 8)

Transition behaviour is explained below :

Current remainder	Next input	Calculation of remainder	Remainder
	0	00 ⇒ 0 MOD 3 ⇒	0 (q_0)
0	1	01 ⇒ 1 MOD 3 ⇒	1 (q_1)
	2	02 ⇒ 2 MOD 3 ⇒	2 (q_2)
	0	10 ⇒ 10 MOD 3 ⇒	1 (q_1)
1	1	11 ⇒ 11 MOD 3 ⇒	2 (q_2)
	2	12 ⇒ 12 MOD 3 ⇒	0 (q_0)
	0	20 ⇒ 20 MOD 3 ⇒	2 (q_2)
2	1	21 ⇒ 21 MOD 3 ⇒	0 (q_0)
	2	22 ⇒ 22 MOD 3 ⇒	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. 2.1.28 : DFA for Example 2.1.28

Example 2.1.29

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

Scanned by CamScanner

Solution :

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

For unary number to be divisible by 3, the number of 0's should be multiple of 3. The FSM is given in Fig. Ex. 2.1.29.

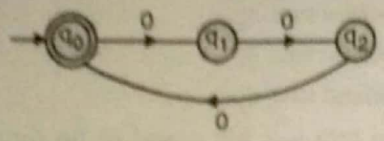


Fig. Ex. 2.1.29

Example 2.1.30

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

Solution :

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

The required DFA is given in Fig. Ex. 2.1.30.

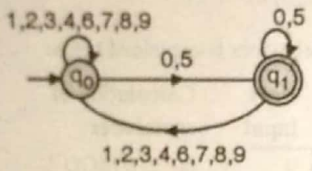


Fig. Ex. 2.1.30

2.1.5 Language of DFA

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

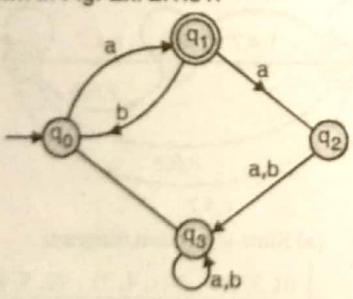
$L(M) = \{\omega \mid \delta^*(q_0, \omega) \text{ is in } F\}$

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

Example 2.1.31

Describe the language accepted by the deterministic finite automata shown in Fig. Ex. 2.1.31.

(i)



(ii)

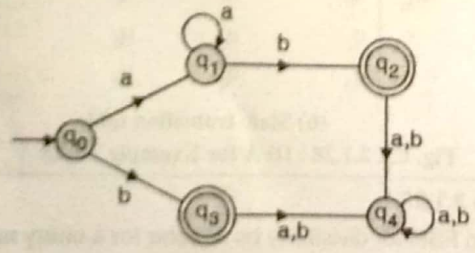


Fig. Ex. 2.1.31

Solution :

(i)

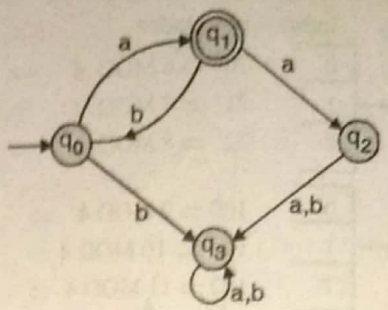


Fig. Ex. 2.1.31(a)

- 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.

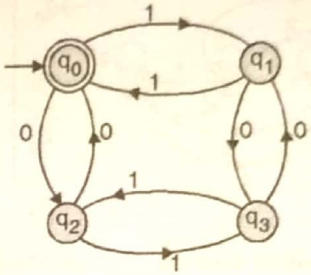


Fig. Ex. 2.1.31(b)

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

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

(ii)

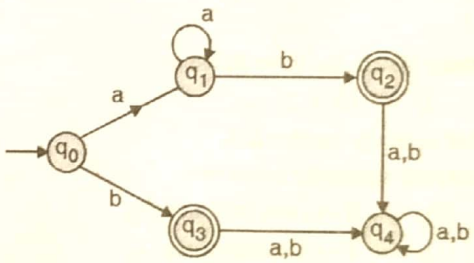


Fig. Ex. 2.1.31(c)

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

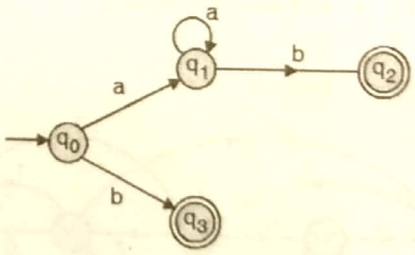


Fig. Ex. 2.1.31(d)

- 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.
- $q_0 \rightarrow q_1 \rightarrow q_1 \cdots q_1 \rightarrow q_2$

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

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

Scanned by CamScanner

Example 2.1.32

Consider the following transition diagram in Fig. Ex. 2.1.32. Test whether the string 110101 is accepted by the finite automata represented by above transition diagram. Show the entire sequence of states traversed.

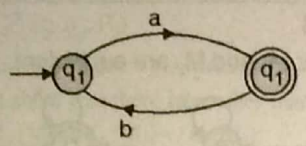


Fig. Ex. 2.1.32

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.

2.1.6 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 δ_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 ∈ Σ
    {
    add a transition < δ1 (s_i, a_i), δ2 (q_i, a_i) to M3
    }
    }
}
```

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

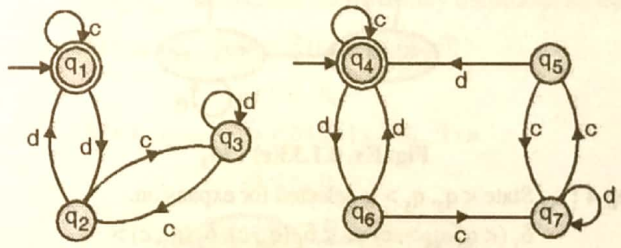
or

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

}

Example 2.1.33

Show whether the automata M_1 and M_2 in given Fig. Ex. 2.1.33 (a) and (b) are equivalent or not.



(a) M_1 **(b) M_2**

Fig. Ex. 2.1.33

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.

$$\delta_3 (< q_1, q_4 >, c) \Rightarrow < \delta_1 (q_1, c), \delta_2 (q_4, c) >$$
$$\Rightarrow < q_1, q_4 >$$
$$\delta_3 (< q_1, q_4 >, d) \Rightarrow < \delta_1 (q_1, d), \delta_2 (q_4, d) >$$
$$\Rightarrow < q_2, q_6 >$$

Scanned by CamScanner

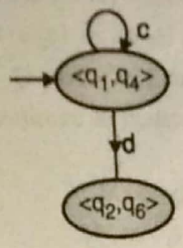


Fig. Ex. 2.1.33(c)

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

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

$$\Rightarrow < q_3, q_7 >$$

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

$$\Rightarrow < q_1, q_4 >$$

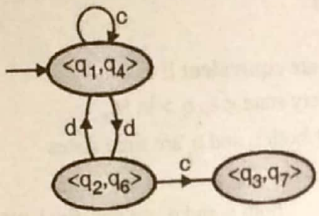


Fig. Ex. 2.1.33(d)

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

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

$$\Rightarrow < q_2, q_5 >$$

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

$$\Rightarrow < q_3, q_7 >$$

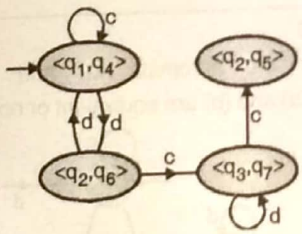


Fig. Ex. 2.1.33(e) : M_3

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

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

$$\Rightarrow < q_3, q_7 >$$

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

$$\Rightarrow < q_1, q_4 >$$

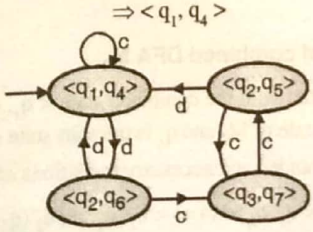


Fig. Ex. 2.1.33(f) : M_3

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.

Example 2.1.34

Find out whether M_1 and M_2 are equivalent.

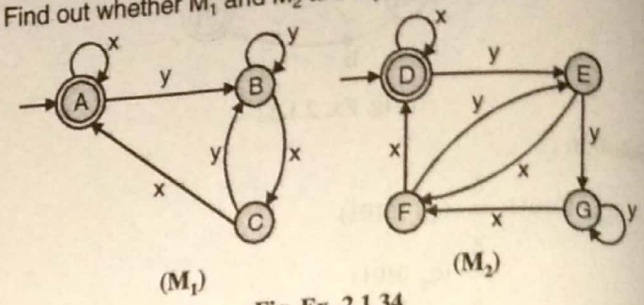


Fig. Ex. 2.1.34

Solution :

Two DFAs are equivalent if for any string $\omega \in \Sigma^*$, either the machines will be in a final state or both the machines will be in a non-final state. This can be established by constructing a combined DFA for M_1 and M_2.

Construction of combined DFA

In the Fig. Ex. 2.1.34(a), following subsets are generated.

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

Hence, machines M_1 and M_2 are equivalent.

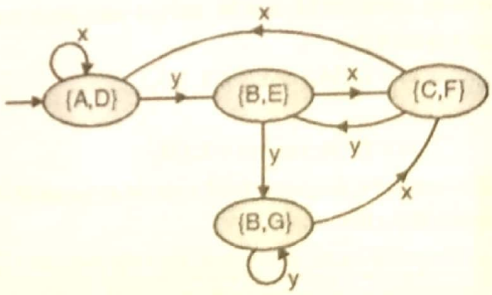


Fig. Ex. 2.1.34(a)

2.1.7 Closure Property of Language Accepted by a DFA

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

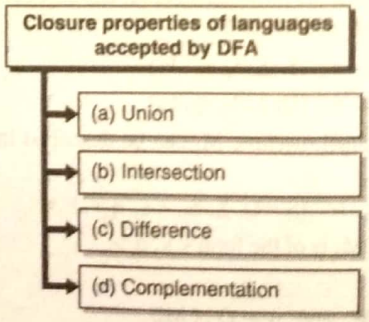


Fig. C2.3 : Closure properties of language accepted by DFA

Scanned by CamScanner

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

→ 2.1.7(a) 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(< s_i, q_i > has not been expanded)
{
    for each alphabet a_i ∈ Σ
    {
    add a transition < δ_1 (s_i, a_i), δ_2(q_i, a_i) > to M_c.
    }
}
}
```

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

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

Set of final states F_3 for M_3

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

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

$$= \{< s_i, q_i > \in \text{states of } M_c \mid s_i \in F_1 \wedge q_i \in F_2\}$$

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

☞ Deriving M_5 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 $= \{< s_i, q_i > \in \text{states of } M_c \mid s_i \in F_1 \cap q_i \notin F_2\}$

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

Example 2.1.35

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.

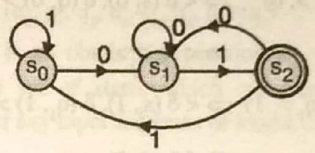
M_2 : For accepting a set of strings over alphabet {0, 1} containing even number of 1's.

Design a DFA for the following :

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

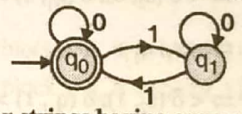
Solution :

$M_1 =$



(a) DFA for strings ending in 01

$M_2 =$



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

Fig. Ex. 2.1.35

Construction of combined DFA M_c

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.

$$\delta(< s_0, q_0 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_0, 0) >$$
$$\Rightarrow < s_1, q_0 >$$
$$\delta(< s_0, q_0 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_0, 1) >$$
$$\Rightarrow < s_0, q_1 >$$

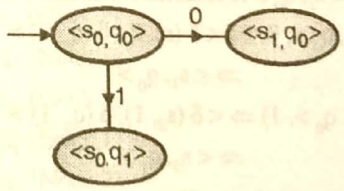


Fig. Ex. 2.1.35(c)

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

$$\delta(< s_0, q_1 >, 0) \Rightarrow < \delta(s_0, 0), \delta(q_1, 0) >$$
$$\Rightarrow < s_1, q_1 >$$
$$\delta(< s_0, q_1 >, 1) \Rightarrow < \delta(s_0, 1), \delta(q_1, 1) >$$
$$\Rightarrow < s_0, q_0 >$$

Scanned by CamScanner

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

$\Rightarrow <s_1, q_0>$

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

$\Rightarrow <s_2, q_1>$

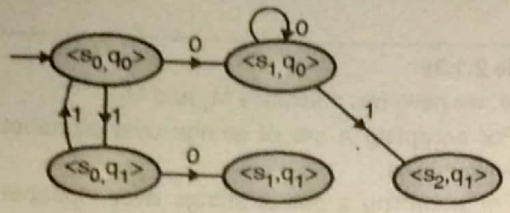


Fig. Ex. 2.1.35(d)

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

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

$\Rightarrow <s_1, q_1>$

$\delta(<s_1, q_1>, 1) \Rightarrow <\delta(s_1, 1), \delta(q_1, 1)>$

$\Rightarrow <s_2, q_0>$

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

$\Rightarrow <s_1, q_1>$

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

$\Rightarrow <s_0, q_0>$

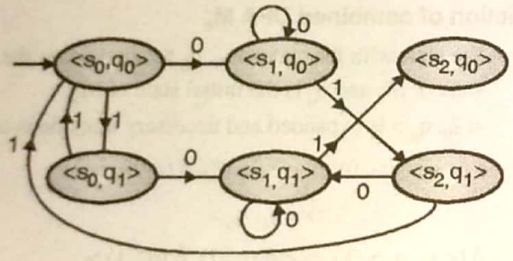


Fig. Ex. 2.1.35(e)

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

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

$\Rightarrow <s_1, q_0>$

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

$\Rightarrow <s_0, q_1>$

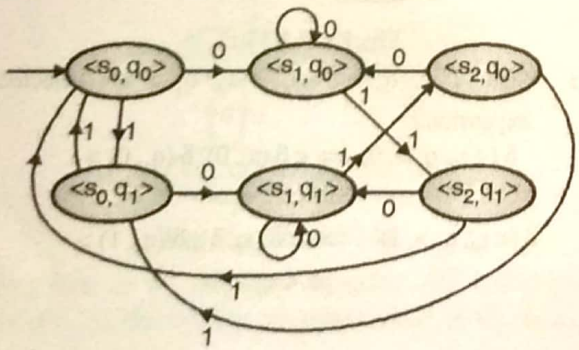


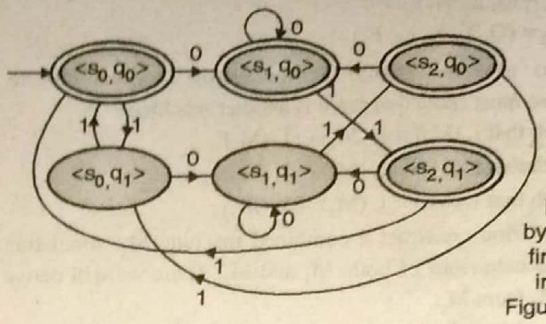
Fig. Ex. 2.1.35(f)

Finding a 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 **or** $q_i \in$ final states of M_2.



Derived by selecting final states in DFA of Figure 2.2.33(f)

Fig. Ex. 2.1.35(g) : DFA for L (M_1) ∪ 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** $q_i \in$ final states of M_2.

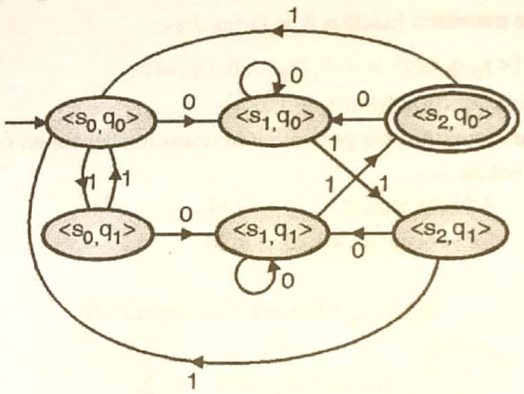


Fig. Ex. 2.1.35(h) : DFA for L (M_1) ∩ L (M_2)

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.

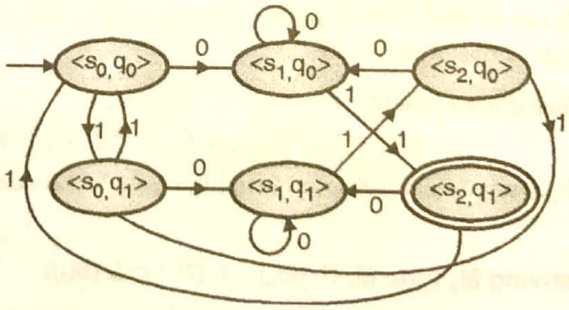


Fig. Ex. 2.1.35(i) : DFA for L (M_1) – L (M_2)

➜ 2.1.7(b) 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,

Scanned by CamScanner

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

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 :

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

2.1.8 Minimization of DFA

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

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

☞ Equivalent states

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

(1) If q_i is an accepting state and q_j is non-accepting, then q_i and q_j are distinguishable.

(2) If q_i is non-accepting and q_j is accepting, then q_i and q_j are distinguishable.

(3) If there is string such that.

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

or

$\delta^*(q_i, x) \notin F$ 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.

2.1.8(a) Algorithm for Minimization DFA's

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

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

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

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

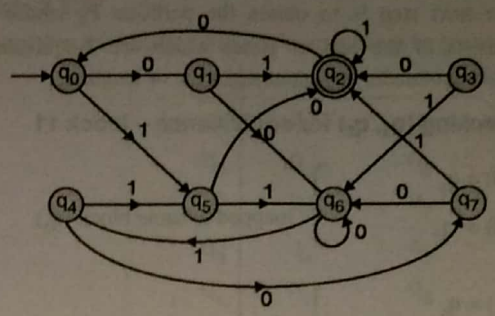


Fig. 2.1.8 : DFA considered for minimization

States of the DFA in Fig. 2.1.8 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,

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

Scanned by CamScanner

- 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. 2.1.8 is given in Fig. 2.1.9. The process of drawing of minimum-state DFA is as below :

1. Start state is $<q_0, q_4>$, since q_0 was start state in Fig. 2.1.8.
2. Only accepting state is $<q_2>$, since q_2 is the only accepting state of Fig. 2.1.8.
3. Since, there is a transition from q_0 to q_1 on input 0 in Fig. 2.1.8, we add a transition from $<q_0, q_4>$ to $<q_1, q_7>$ on input 0 in Fig. 2.1.9.
4. Since, there is a transition from q_0 to q_5 on input 1 in Fig. 2.1.8, we add a transition from $<q_0, q_4>$ to $<q_3, q_5>$ on input 1 in Fig. 2.1.9.
5. Other transitions can be added in a similar fashion.

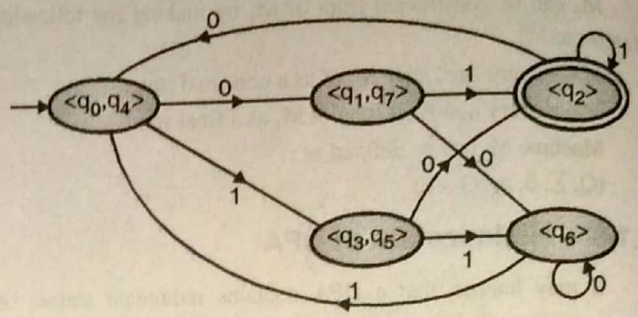


Fig. 2.1.9 : Minimum state DFA equivalent to Fig. 2.1.8

Example 2.1.36

Construct the minimum state automata equivalent to given DFA.

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

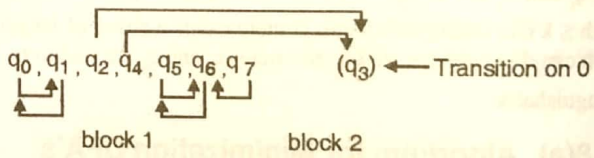
Solution :

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

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

block 1 block 2

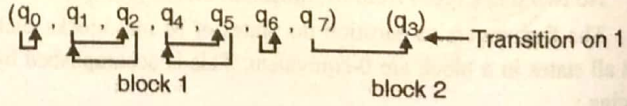
Step 2 : Finding 1-equivalence partitioning of states by considering transition on '0' and transition on '1'.



On input 0, block 1 is successor of q_0, q_1, q_5, q_6, q_7.

On input 0, block 2 is successor of q_2, q_4.

$\therefore q_2, q_4$ are distinguishable from $q_0, q_1\ q_5, q_6, q_7$



On input 1, block 2 is successor of q_7.

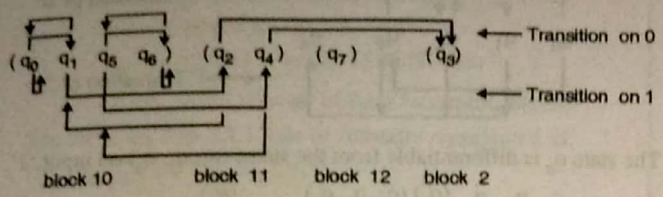
On input 1, block 1 is successor of $q_0, q_1\ q_2, q_4, q_5, q_6$.

q_7 is distinguishable from $q_0, q_1, q_2, q_4, q_5, q_6$.

$P_1 = (q_0, q_1, q_5, q_6)\ (q_2, q_4)\ (q_7)\ (q_3)$

Scanned by CamScanner

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



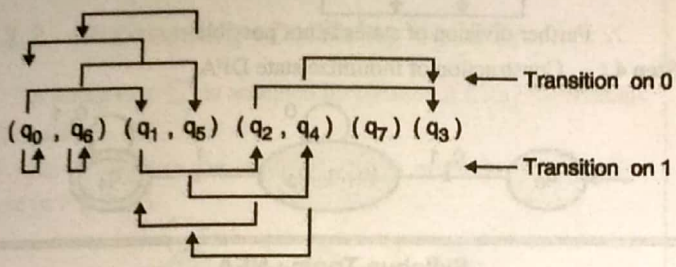
On input 1, block 11 is successor of q_1, q_5.

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.



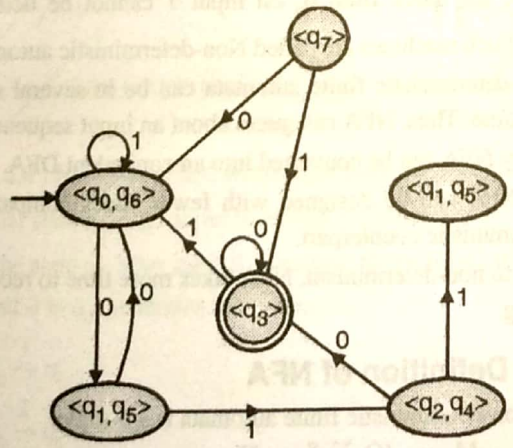
Blocks can not be divided further.

$\therefore P_3 = P_2 = (q_0, q_6)\ (q_1, q_5)\ (q_2, q_4)\ (q_7)\ (q_3)$ 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. 2.1.36

Example 2.1.37

Construct the minimum-state DFA equivalent to given DFA.

	0	1
$\rightarrow q_0$	q_1	q_4
q_1	q_2	q_5
q_2^*	q_3	q_7
q_3	q_4	q_7
q_4	q_5	q_8
q_5^*	q_6	q_1
q_6	q_7	q_1
q_7	q_8	q_2
q_8^*	q_0	q_4

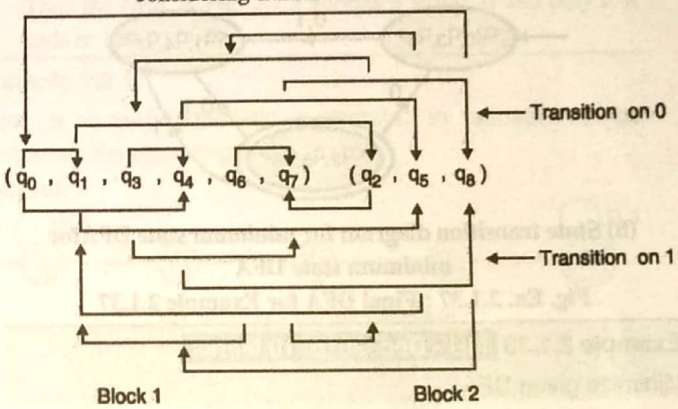
Solution :

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

$$P_0 = (q_0, q_1, q_3, q_4, q_6, q_7) \qquad (q_2, q_5, q_8)$$

block 1 block 2

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



On input 0, block 1 is successor of q_0, q_3, q_6.

On input 0, block 2 is successor of q_1, q_4, q_7.

On input 1, block 1 is successor of q_0, q_3, q_6.

On input 1, block 2 is successor of q_1, q_4, q_7.

} block 1

States q_0, q_3, q_6 are distinguishable from q_1, q_4, q_7.

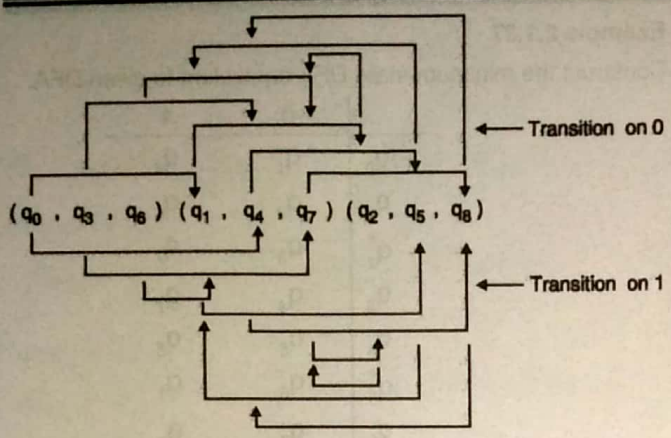
Block 2 is 1-equivalent

$$P_1 = (q_0, q_3, q_6) \qquad (q_1, q_4, q_7) \qquad (q_2, q_5, q_8)$$

block 11 block 12 block 2

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

Scanned by CamScanner



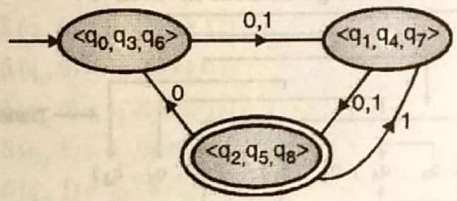
Blocks can not be divided further.

$\therefore P_2 = P_1 = (q_0, q_3, q_6)\ (q_1, q_4, q_7)\ (q_2, q_5, q_8)$ 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



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

Fig. Ex. 2.1.37 : Final DFA for Example 2.1.37

Example 2.1.38 MU - Dec. 2016. 10 Marks

Minimize given DFA

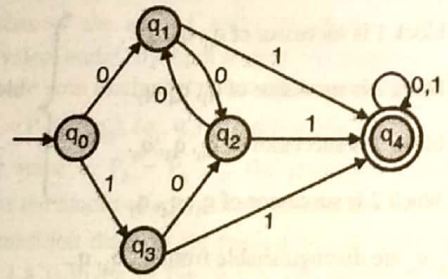


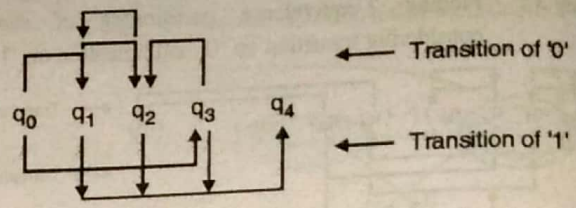
Fig. Ex. 2.1.38

Soln. :

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

$$P_0 = \underset{\text{block 1}}{(q_0, q_1, q_2, q_3)} \quad \underset{\text{block 2}}{(q_4)}$$

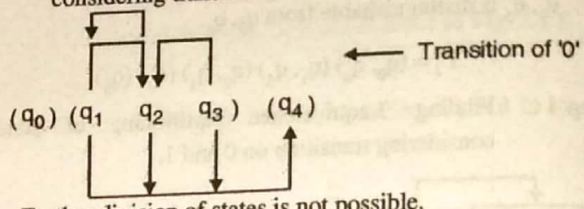
Step 2 : Finding 1-equivalence partitioning by considering transitions on '0' and '1'.



The state q_0 is differentiable from the states (q_1, q_2, q_3) on input '1'.

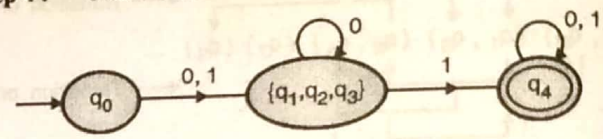
$$\therefore\ p_1 = (q_0)\ (q_1, q_2, q_3) \quad (q_4)$$

Step 3 : Finding 2-equivalence partitioning of states by considering transitions.



$\therefore$ Further division of states is not possible.

Step 4 : Construction of minimize state DFA.



Syllabus Topic : NFA

2.2 Non-deterministic Finite Automata

Non-deterministic finite automata can reside in multiple states at the same time. The concept of non-deterministic finite automata is being explained with the help of transition diagram (Fig. 2.2.1).

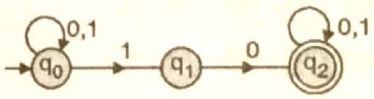


Fig. 2.2.1 : Transition diagram of a nondeterministic finite automata

If the automata is in the state q_0 and the next input symbol is 1, then the next state will be either :

(1) q_0 (2) q_1.

Thus, the move from q_0 on input 1 cannot be determined uniquely. Such machines are called Non-deterministic automata.

- Non-deterministic finite automata can be in several states at any time. Thus, NFA can guess about an input sequence.
- Every NFA can be converted into an equivalent DFA.
- An NFA can be designed with fewer states compare to its deterministic counterpart.
- Due to non-determinism, NFA takes more time to recognize a string.

2.2.1 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

Scanned by CamScanner

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. 2.2.1, can be formally represented as, $(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$

where, the transition function δ is given below in Fig. 2.2.2.

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

Fig. 2.2.2 : A transition table for NFA in Fig. 2.2.1

2.2.2 Processing of a String by NFA

A string $\omega \in \Sigma^*$ is accepted by NFA M if $\delta^*(q_0, \omega)$ contains a final state.

Let us see how the string 011010 is processed by the NFA given in Fig. 2.2.3.

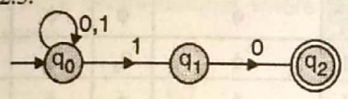


Fig. 2.2.3 : A sample NFA

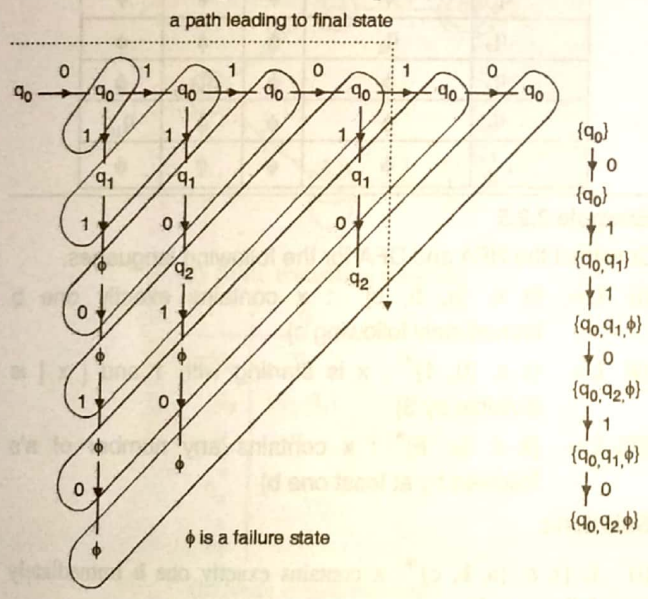


Fig. 2.2.4 : States reached while processing of 011010

- From the state q_0, after input 0, resulting state is q_0.
- From the state q_0, after input 1, resulting states are q_0, q_1.

An input 0 in q_0, generates two paths :

1. $q_0 \xrightarrow{1} q_0$
2. $q_0 \xrightarrow{1} q_1$

On input 011010, the automata can take any of the four paths:

1. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} \phi \xrightarrow{0} \phi \xrightarrow{1} \phi \xrightarrow{0} \phi$

[Once in ϕ state (dead state) machine will remain in ϕ state]

2. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} \phi \xrightarrow{0} \phi$
3. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$
4. $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0$

Out of the four paths, the path number 3 takes the machine from start state to the final state on input 011010. q_2 is a final state.

A string is accepted by non-deterministic finite automata, if there exists a path that takes the machine to a final state.

Thus, 011010 will be accepted by the NFA of Fig. 2.2.3.

- 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 :

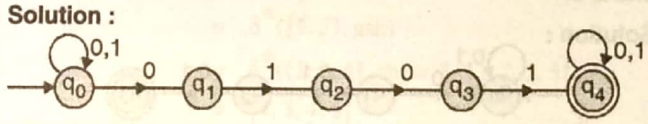
$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$$

- 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 in Fig. 2.2.3 accepts a string, if and only if it ends in 10.

Example 2.2.1

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. 2.2.1

- 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.

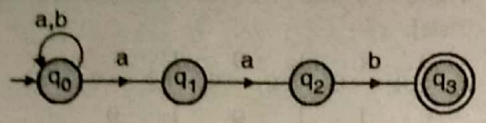
Example 2.2.2

Construct a NFA that accepts any positive number of occurrences of various strings from the following language L given by,

$L = \{x \in \{a,b\} \mid x \text{ ends with aab}\}$

Scanned by CamScanner

Solution :



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

Fig. Ex. 2.2.2

- The language accepted by the NFA shown in Fig. Ex. 2.2.2 consists of string ending in aab. A string of length n, ending in aab can be accepted by the NFA if :
  1. The machine remains in q_0 for first n-3 inputs.
  2. On last three inputs aab, the machine makes the transition as :

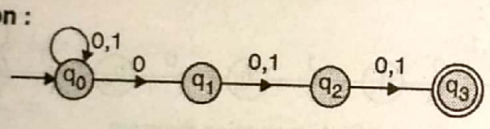
$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$$

- To reach the final state, and to remain there, it is necessary that the last two characters of the string should be 10.
- Thus the NFA accepts a string, if and only if it ends in 10.

Example 2.2.3

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. 2.2.3 : NFA for Example 2.2.3

- 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 2.2.4

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 :

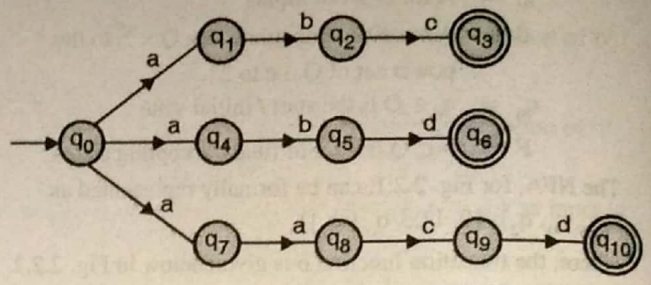


Fig. Ex. 2.2.4 : State diagram of NFA

- 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.

Table Ex. 2.2.4 : State table for NFA

	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

Example 2.2.5

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 ∈ {0, 1}* : x is starting with 1 and | x | is divisible by 3}

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

Solution :

(i) L {x ∈ {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.

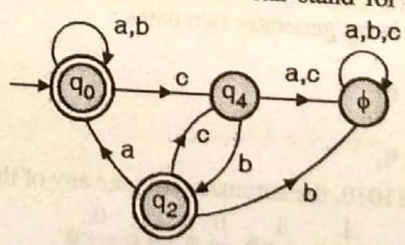


Fig. Ex. 2.2.5(a) : A DFA standing for both DFA and NFA

Scanned by CamScanner

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

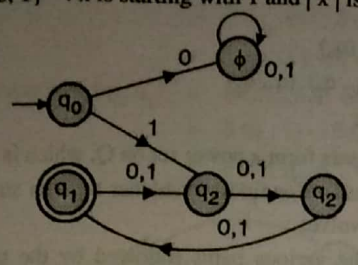


Fig. Ex. 2.2.5(b) : A DFA standing for both DFA and NFA

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

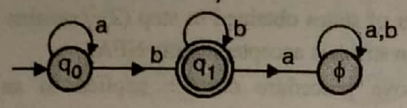


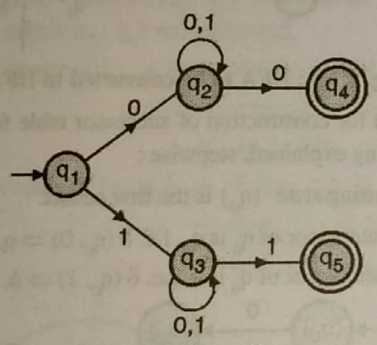
Fig. Ex. 2.2.5(c) : A DFA standing for both DFA and NFA

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

Example 2.2.6

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

Solution :



(a) State transition diagram

	0	1
$\rightarrow q_1$	q_2	q_3
q_2	$\{q_2, q_4\}$	q_2
q_3	q_3	$\{q_3, q_4\}$
q_4^*	–	–
q_5^*	–	–

(b) State transition table

Fig. Ex. 2.2.6

- 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 2.2.7

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

(a) Draw a transition diagram

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

(c) Calculate $\delta^*(1, abaab)$

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

Solution :

(a) Transition diagram is given in Fig. Ex. 2.2.7.

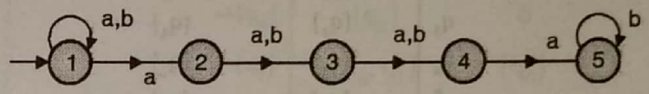


Fig. Ex. 2.2.7 : State transition diagram

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

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

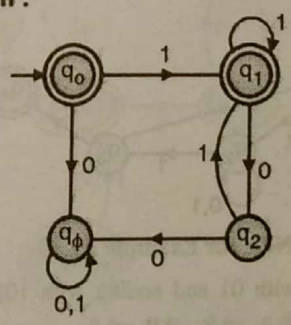
$= \delta^*(\{1, 2, 4\}, ab) = \delta(\{1, 2, 3, 5\}, b)$

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

Example 2.2.8

Construct a FA for accepting L over {0,1} such that each 0 is immediately preceded and immediately followed by 1.

Solution :



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

Fig. Ex. 2.2.8

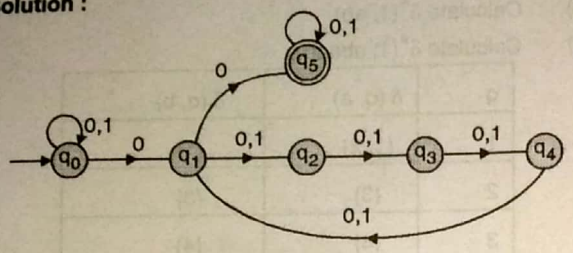
- If the starting symbol is 0, string is rejected by entering the failure state q_ϕ.
- A loop through $q_1 \rightarrow q_2 \rightarrow q_1$ ensures that every 0 is preceded and succeeded by 1.
- An input 0 in state q_2 will mean two consecutive 0's. The string is rejected by entering the failure state q_ϕ.

Scanned by CamScanner

Example 2.2.9

Construct a NFA that accept the set of strings in (0 + 1)* such that some two 0's are separated by string whose length is 4i, for some i >=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. 2.2.9 : NFA for Example 2.2.9

- The loop $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_1$ can absorb a string of length 4i.
- On receiving the 0 of the portion of the string containing 'two 0's separated by string of length 4i, the machine makes a move to q_1 from q_0. Then, it loops i times in $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_1$. Then it gets a 0 and makes a move from q_1 to q_5.

Example 2.2.10

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

Solution :

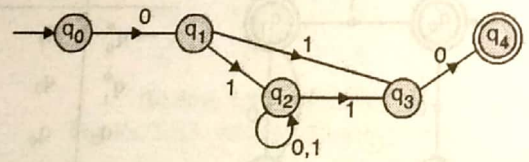


Fig. Ex. 2.2.10 : NFA for Example 2.2.10

- 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.

Syllabus Topic : NFA to DFA

2.2.3 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 :

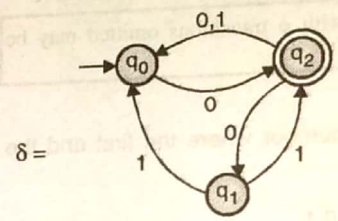
1. ϕ
2. (q_0), (q_1), (q_2)
3. (q_0, q_1), (q_0, q_2), (q_1, q_2)
4. (q_0, q_1, q_2)

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^*	$\{q_0, q_1\}$	$\{q_0\}$

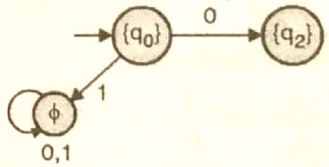
Fig. 2.2.5 : NFA to be converted to DFA

Algorithm for construction of successor table for the NFA of Fig. 2.2.5 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. $\delta(q_0, 1) \Rightarrow \phi$.



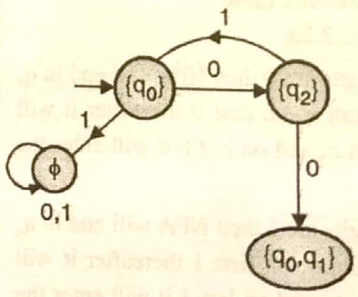
	0	1
$\rightarrow \{q_0\}$	$\{q_2\}$	ϕ

Fig. 2.2.6

Step 2 : A new subset $\{q_2\}$ is generated. Successor of the subset $\{q_2\}$ are generated.

0 – successor of $\{q_2\}$ are $\{q_0, q_1\}$.

1 – successor of $\{q_2\}$ are $\{q_0\}$.



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

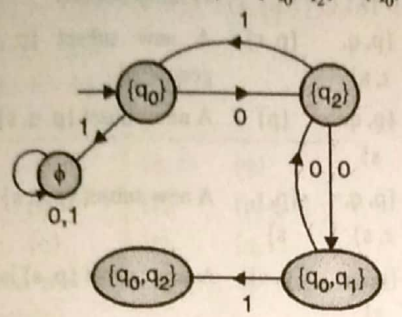
Fig. 2.2.7

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

Scanned by CamScanner

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



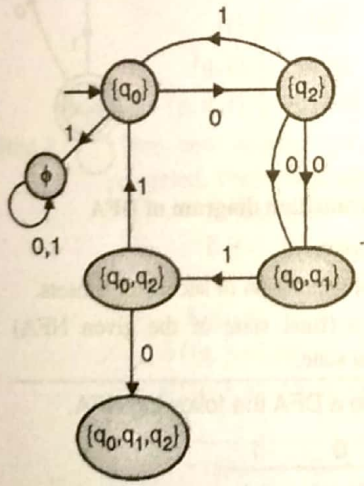
	0	1
→$\{q_0\}$	$\{q_2\}$	ϕ
$\{q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_2\}$	$\{q_0, q_2\}$

Fig. 2.2.8

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



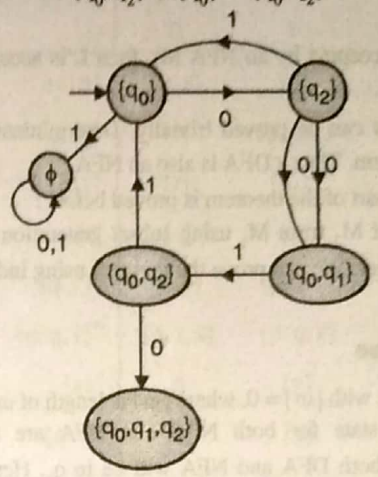
	0	1
→$\{q_0\}$	$\{q_2\}$	ϕ
$\{q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$

Fig. 2.2.9

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

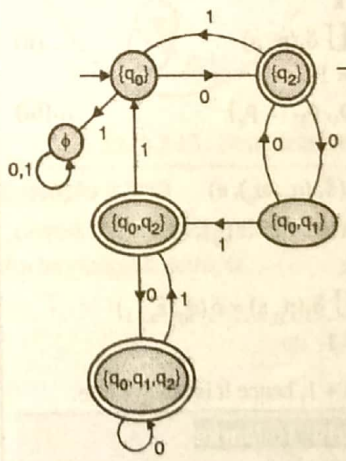


	0	1
→$\{q_0\}$	$\{q_2\}$	ϕ
$\{q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_0\}$

Fig. 2.2.10

Since, a new subset is not generated the process of subset generation stops.

Step 6 : q_2 is the final state in NFA of Fig. 2.2.5. Every subset containing q_2 should be takes as a final state.



	0	1
→$\{q_0\}$	$\{q_2\}$	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. 2.2.11 : Final DFA after marking of accepting states

Theorem 2.1 : A language L is accepted by some NFA if and only if it is accepted by some DFA.

OR

For every NFA, there exists an equivalent DFA.

Scanned by CamScanner

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 $\{p_1, p_2, \cdots p_k\}$

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

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

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

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

$$\begin{aligned} \delta_2(q_0, \omega_{n+1}) &= \delta_2(\delta_2(q_0, \omega_n), a) \\ &= \delta_2(\{p_1, p_2, \cdots p_k\}, a) \\ &= \bigcup_{i=1}^{k} \delta_1(p_i, a) = \delta_1(q_0, s_{n+1}) \end{aligned}$$

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

Example 2.2.11 **MU - Dec. 2014, 10 Marks**

Convert to a DFA the following NFA :

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

Solution :

Table Ex. 2.2.11 : State transition table of DFA

	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, 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}	

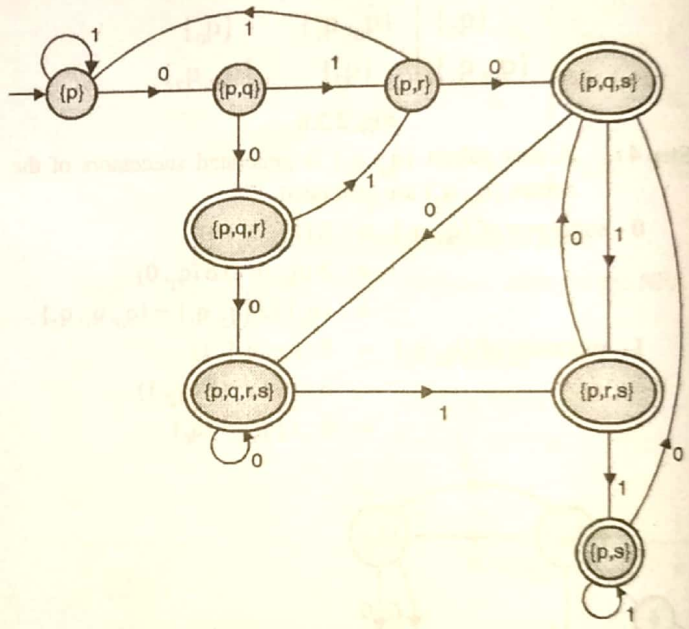


Fig. Ex. 2.2.11 : State transition diagram of DFA

- 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 2.2.12 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} = δ ({p}, 0) = {q, s}

1-Successor of {p} = δ ({p}, 1) = {q}

Scanned by CamScanner

	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 shown in Fig. Ex. 2.2.12.

Table Ex. 2.2.12 : State transition table of DFA

	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}

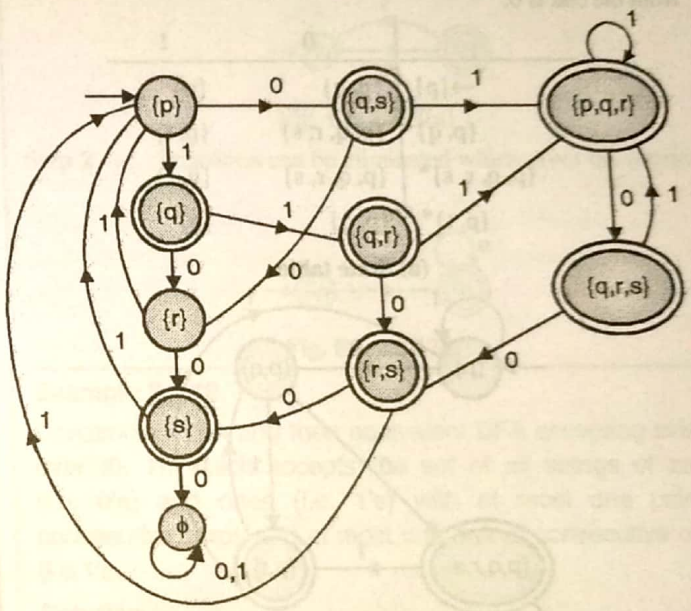


Fig. Ex. 2.2.12 : State transition diagram of the DFA

Example 2.2.13

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} = δ({p}, 0) = {p, q}

1-Successor of {p} = δ({p}, 1) = {p}

Step 2 : The new subsets {p, q} is generated. Successors of {p, q} are calculated.

Scanned by CamScanner

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

Step 3 : Two new subsets $\{p, q, r, s\}$ and $\{p, t\}$ are generated. Their successors are calculated.

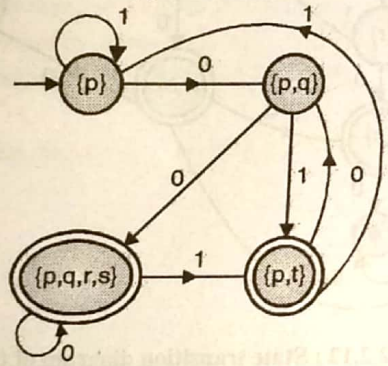
$$\delta(\{p, q, r, s\}, 0) = \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) \cup \delta(s, 0) = \{p, q\} \cup \{r, s\} \cup \{p, r\} \cup \phi = \{p, q, r, s\}$$
$$\delta(\{p, q, r, s\}, 1) = \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1) \cup \delta(s, 1) = \{q\} \cup \{t\} \cup \{t\} \cup \phi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

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. 2.2.13 : Final DFA for Example 2.2.13

Example 2.2.14 MU- Dec. 2016, 5 Marks

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 :

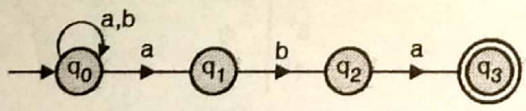


Fig. Ex. 2.2.14(a) : Non-deterministic finite automata for Example 2.2.14

Non-determinism should be utilized to full extent while designing an NFA. A string of length n, ending in aba can be recognized by the NFA given in Fig. Ex. 2.2.14(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.

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

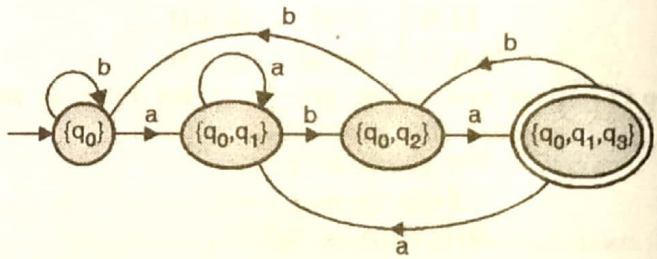
Step 3 : A new subset $\{q_0, q_2\}$ is generated. Successors of $\{q_0, q_2\}$ are calculated.

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

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

	a	b
→$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$*	$\{q_0, q_1\}$	$\{q_0, q_2\}$

(c) State table of the DFA

Fig. Ex. 2.2.14

Example 2.2.15

Construct a NFA and its equivalent DFA for accepting a language defined over $\Sigma = \{0,1\}$ such that each string has two consecutive zeroes followed by 1.

Scanned by CamScanner

Solution :

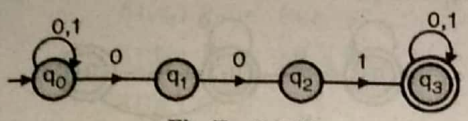


Fig. Ex. 2.2.15

A string of length n, having two consecutive zeroes followed by 1 can be recognized by the NFA. Symbols before 001 can be absorbed by the state q_0 by making a guess. On guessing 001, the machine can make a transition from q_0 to q_3.

NFA to DFA conversion

	0	1	
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	A new subset $\{q_0, q_1\}$ is generated
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$	A new subset $\{q_0, q_1, q_2\}$ is generated
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_3\}$	A new subset $\{q_0, q_3\}$ is generated
$\{q_0, q_3\}^*$	$\{q_0, q_1, q_3\}$	$\{q_0, q_3\}$	A new subset $\{q_0, q_1, q_3\}$ is generated
$\{q_0, q_1, q_3\}^*$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_3\}$	A new subset $\{q_0, q_1, q_2, q_3\}$ is generated
$\{q_0, q_1, q_2, q_3\}^*$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_3\}$	

Transition behaviour of $\{q_0, q_3\}$, $\{q_0, q_1, q_3\}$ and $\{q_0, q_1, q_2, q_3\}$ is identical. Therefore, these three states can be merged into a single state with resultant DFA as shown in Fig. Ex. 2.2.15(a) :

		0	1
$\rightarrow\{q_0\}$	as $\rightarrow$A	B	A
$\{q_0, q_1\}$	as B	C	A
$\{q_0, q_1, q_2\}$	as C	C	D
$\{q_0, q_3\}$	as D*	D	D

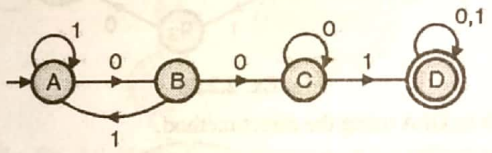
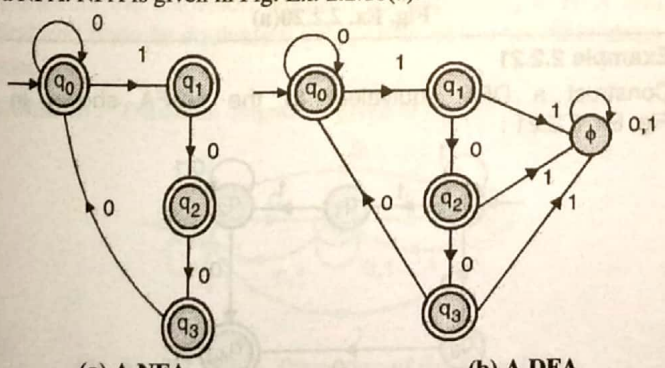


Fig. Ex. 2.2.15(a)

Example 2.2.16

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 a NFA. NFA is given in Fig. Ex. 2.2.16(a).



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

Fig. Ex. 2.2.16

From Fig. Ex. 2.2.16(a), it is clear that between any two 1's there will be at least three 0's. Thus every block of 4 consecutive symbols will contain at least three 0's. Every state is a final state as the failure cases have not been shown.

Above NFA can be converted into a DFA by introducing the dead state. DFA is given in Fig. Ex. 2.2.16(b).

Example 2.2.17 **MU - May 2015, 10 Marks**

Convert the following NFA to a reduced DFA

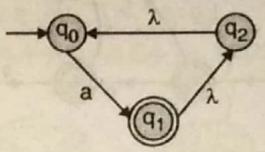


Fig. Ex. 2.2.17

Solution :

Step 1 : The state q_2 can be eliminated

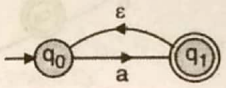


Fig. Ex. 2.217(a)

Step 2 : $\in$ - Transition can be eliminated which gives the required DFA.

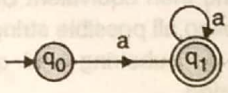


Fig. Ex. 2.2.17(b)

Example 2.2.18

Construct a NFA and then equivalent DFA accepting strings over {0, 1}, which accepts the set of all strings of zeros (i.e. 0's) and ones (i.e. 1's) with at most one pair of consecutive zeros and at most one pair of consecutive ones (i.e.1's).

Solution :

This problem is fully deterministic in nature. Technically, a DFA with ϕ transitions not mentioned can be called as NFA.

NFA is given in Fig. Ex. 2.2.18(a).

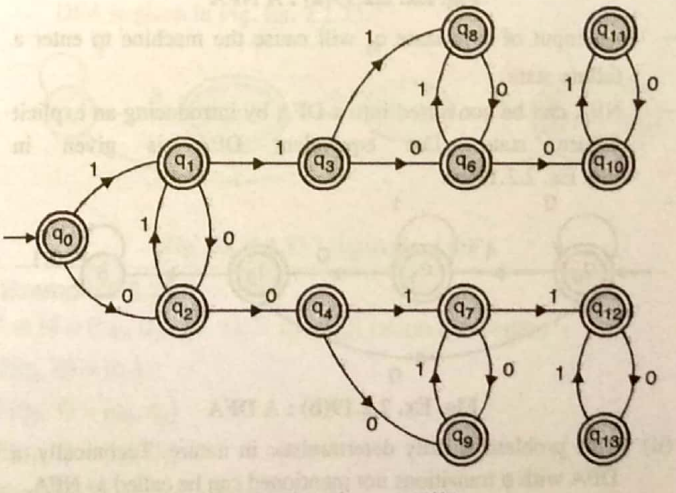


Fig. Ex. 2.2.18(a) : A NFA

Scanned by CamScanner

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} → A pair of 00 and a pair of 11 have been seen

NFA in Fig. Ex. 2.2.18(a) can be converted into a DFA by introducing an explicit dead state.

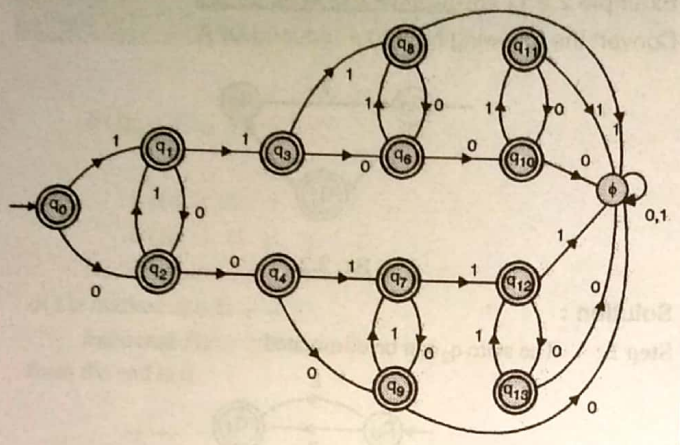


Fig. Ex. 2.2.18(b) : A DFA

Example 2.2.19

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 and ones which do not contain 011 as substring.

Solution :

(i) This problem is fully deterministic in nature as technically, a DFA with ϕ transitions not mentioned can be called as NFA.

– State q_1 is for previous character as 1.

– State q_2 is for previous two characters as 10.

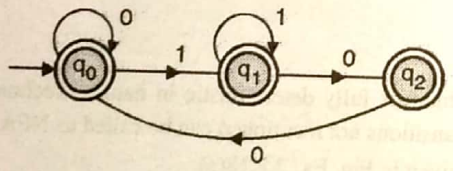


Fig. Ex. 2.2.19(a) : A NFA

– An input of 1 in state q_2 will cause the machine to enter a failure state.

– NFA can be converted into a DFA by introducing an explicit failure state. The equivalent DFA is given in Fig. Ex. 2.2.19(b).

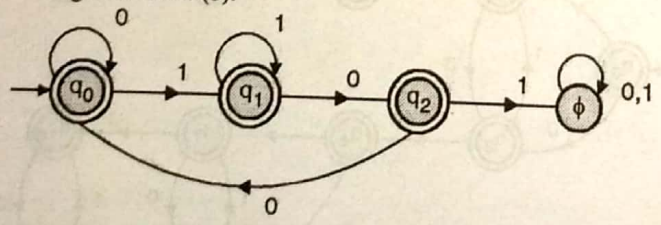


Fig. Ex. 2.2.19(b) : A DFA

(ii) This problem is fully deterministic in nature. Technically, a DFA with ϕ transitions not mentioned can be called as NFA.

– State q_1 is for previous character as 0.

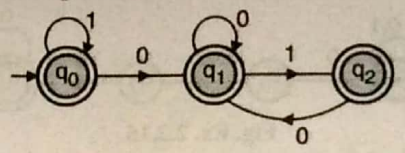


Fig. Ex. 2.2.19(c) : A NFA

– 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.

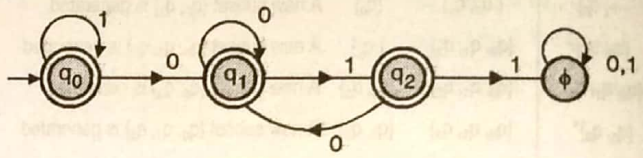


Fig. Ex. 2.2.19(d) : An equivalent DFA

Example 2.2.20 MU - May 2014, 10 Marks

Design a NFA for accepting input strings that contain either the keyword 000 or the keyword 010 and convert it into an equivalent DFA.

Solution :

1. NFA for accepting strings either 000 or 010 as substring.

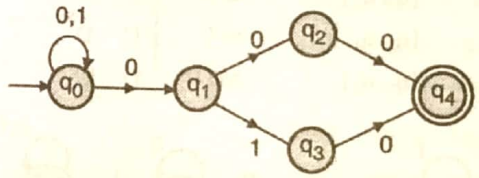


Fig. Ex. 2.2.20

2. NFA to DFA using the direct method.

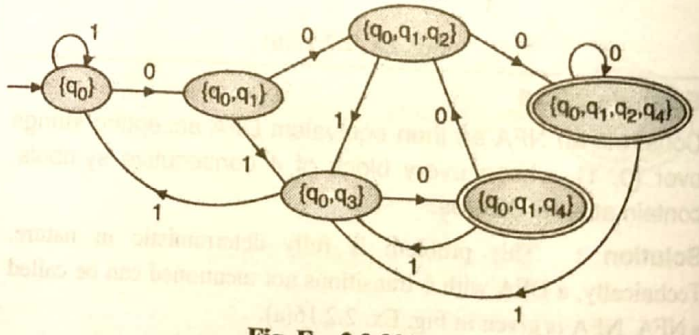


Fig. Ex. 2.2.20(a)

Example 2.2.21

Construct a DFA equivalent to the NDFA shown in Fig. Ex. 2.2.21 :

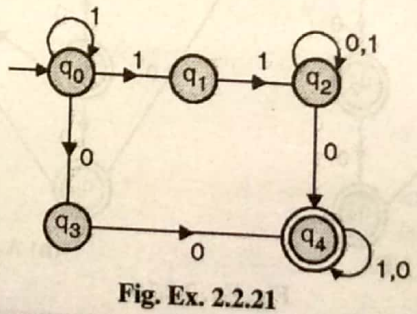


Fig. Ex. 2.2.21

Scanned by CamScanner

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

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

DFA is given in Fig. Ex. 2.2.21(a).

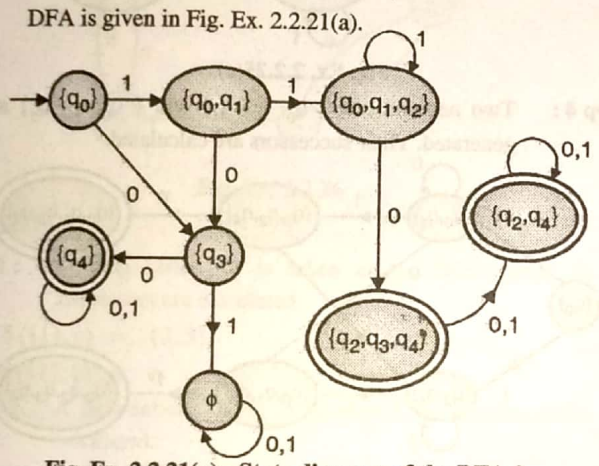


Fig. Ex. 2.2.21(a) : State diagram of the DFA for Example 2.2.21

Example 2.2.22

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. 2.2.22(a).

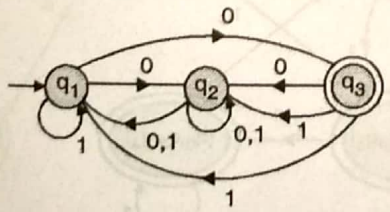


Fig. Ex. 2.2.22(a) : Transition of the given NFA

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. 2.2.22(b).

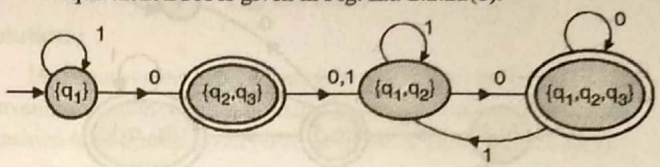


Fig. Ex. 2.2.22(b) : Equivalent DFA

Example 2.2.23

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. 2.2.23.

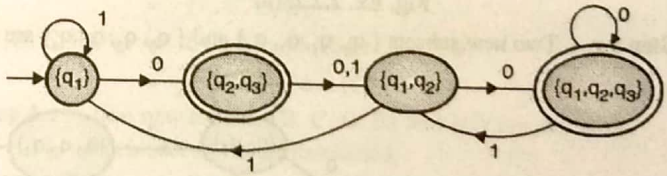


Fig. Ex. 2.2.23 : Equivalent DFA

Example 2.2.24

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.

Scanned by CamScanner

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. 2.2.24.

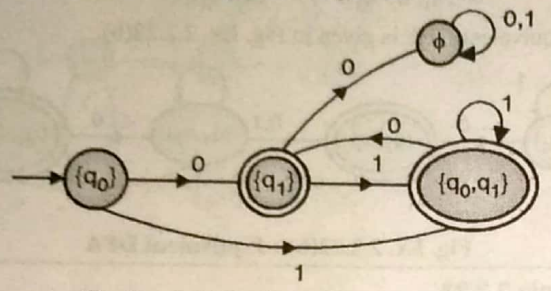


Fig. Ex. 2.2.24 : An equivalent DFA

Example 2.2.25

Convert NFA shown in Fig. Ex. 2.2.25 to its corresponding DFA.

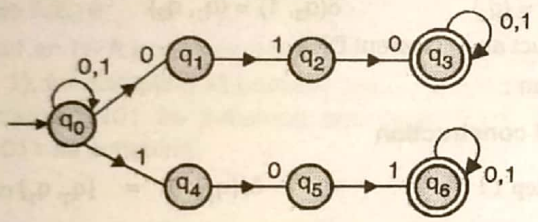


Fig. Ex. 2.2.25

Solution :

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

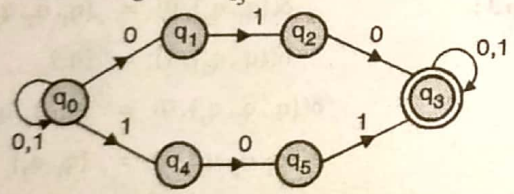


Fig. Ex. 2.2.25(a)

Step 2 :

We start with the subset $\{q_0\}$

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

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

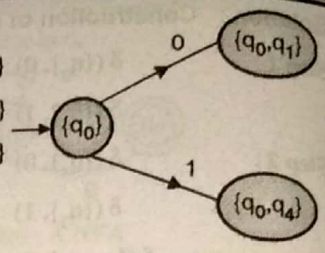


Fig. Ex. 2.2.25(b)

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

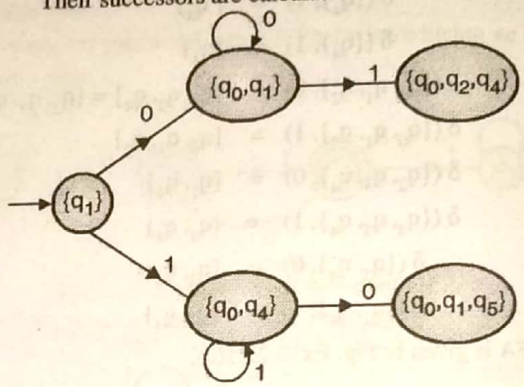


Fig. Ex. 2.2.25(c)

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

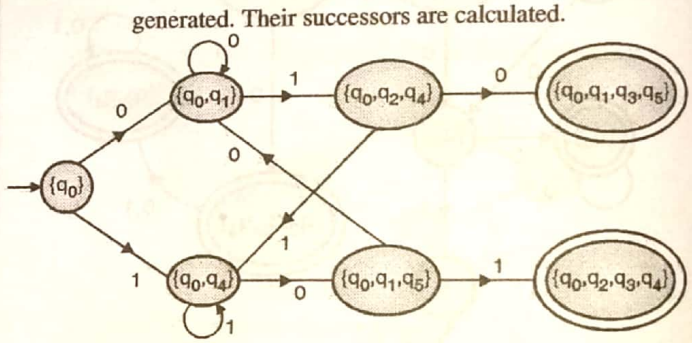


Fig. Ex. 2.2.25(d)

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.

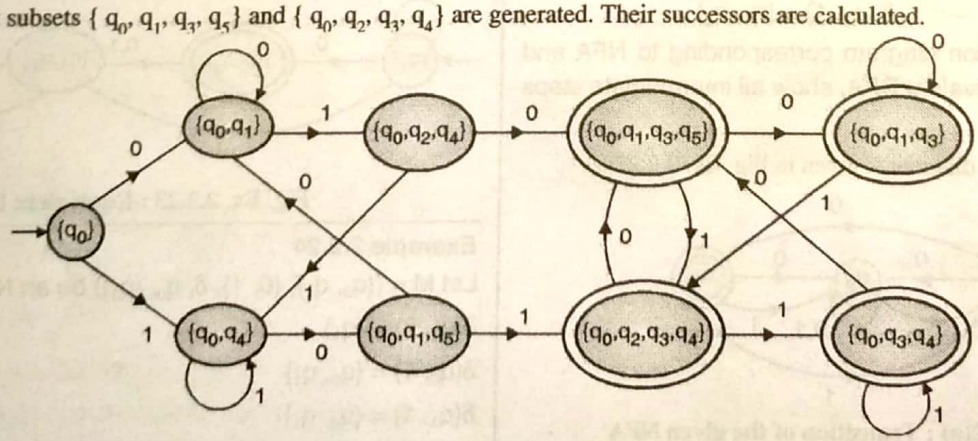


Fig. Ex. 2.2.25(e)

Scanned by CamScanner

Step 6 : Following states can be combined into a single state. These states are $\{q_0, q_1, q_3, q_5\}$, $\{q_0, q_1, q_3\}$, $\{q_0, q_2, q_3, q_4\}$, $\{q_0, q_3, q_4\}$

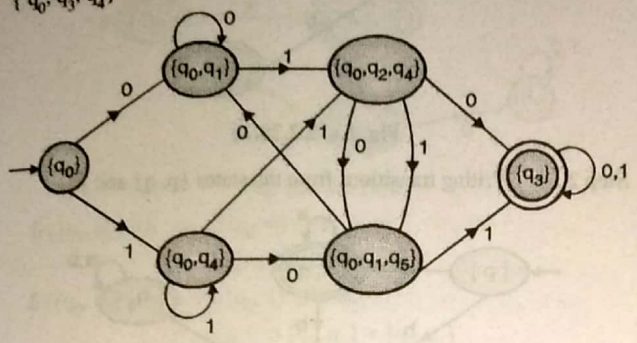


Fig. Ex. 2.2.25(f) : Final DFA

Example 2.2.26

Convert NFA shown in Fig. Ex. 2.2.26 to its corresponding DFA.

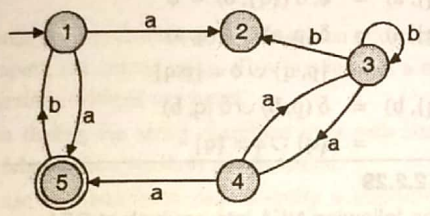


Fig. Ex. 2.2.26

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. 2.2.26(a).

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

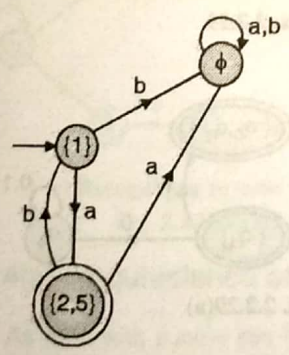


Fig. Ex. 2.2.26(a) : DFA for Example 2.2.26

Example 2.2.27

Convert the NFA shown in Fig. Ex. 2.2.27 to DFA.

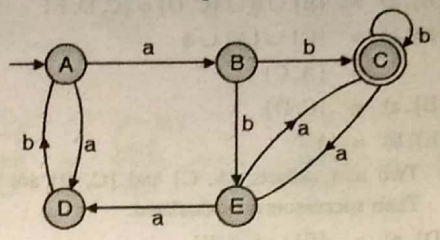


Fig. Ex. 2.2.27

Solution :

In some cases where number of states is large, it is convenient to derive subsets using a state transition table. State transition table for the given NFA is shown in Table Ex. 2.2.27(a).

Table Ex. 2.2.27 : State transition table for NFA of Example 2.2.27

	a	b
→A	{B, D}	ϕ
B	ϕ	{C, E}
C*	{E}	{C}
D	ϕ	{A}
E	{C, D}	ϕ

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

$\delta(\{A\}, a) = \{B, D\}$

$\delta(\{A\}, b) = \phi$

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

$\delta(\{B, D\}, a) = \delta(B, a) \cup \delta(D, a)$

$= \phi \cup \phi = \phi$

$\delta(\{B, D\}, b) = \delta(B, b) \cup \delta(D, b)$

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

Step 3 : A new subset {A, C, E} is generated. Its successors are calculated.

$\delta(\{A, C, E\}, a) = \delta(A, a) \cup \delta(C, a) \cup \delta(E, a)$

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

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

$\delta(\{A, C, E\}, b) = \delta(A, b) \cup \delta(C, b) \cup \delta(E, b)$

$= \phi \cup \{C\} \cup \phi = \{C\}$

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

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

$= \phi \cup \{E\} \cup \phi \cup \{C, D\} = \{C, D, E\}$

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

$= \{C, E\} \cup \{C\} \cup \{A\} \cup \phi$

$= \{A, C, E\}$

$\delta(\{C\}, a) = \{E\}$

$\delta(\{C\}, b) = \{C\}$

Scanned by CamScanner

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. 2.2.27(a).

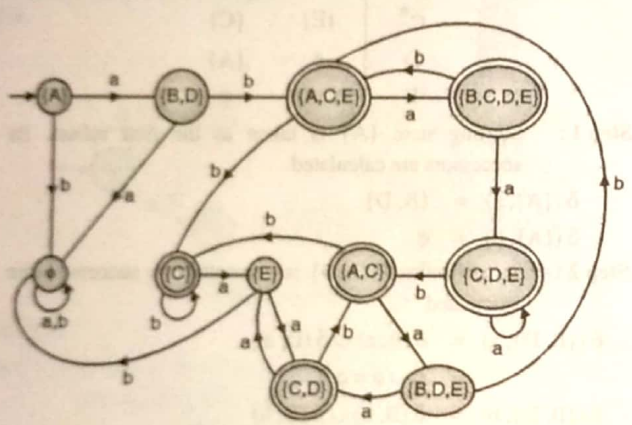


Fig. Ex. 2.2.27(a) : Final DFA for Example 2.2.27

Example 2.2.28

Convert the following NFA to its equivalent DFA.

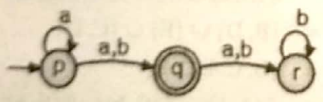


Fig. Ex. 2.2.28

Solution :

The state r is is a dead state and it can be removed.

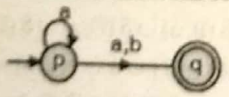


Fig. Ex. 2.2.28(a)

NFA to DFA

Step 1 : Writing transition from the state p.

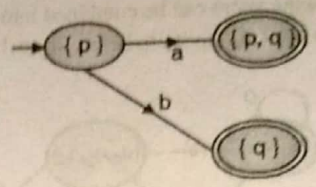


Fig. Ex. 2.2.28(b)

Step 2 : Writing transitions from the states {p, q} and {q}.

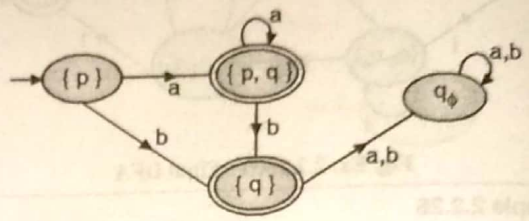


Fig. Ex. 2.2.28(c)

$\delta(\{q\}, a) = \phi, \delta(\{q\}, b) = \phi$

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

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

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

$= \{q\} \cup \phi = \{q\}$

Example 2.2.29

Convert the following NFA into equivalent DFA

$M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ where

$\delta(q_0, 0) = \{q_0, q_1\}, \delta(q_0, 1) = \{q_1\}, \delta(q_1, 0) = \phi$

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

Solution :

The state transition table for the above NFA is as given below :

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

Step 1 : Transitions from the starting state q_0.

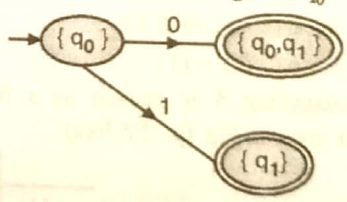


Fig. Ex. 2.2.29

Step 2 : Transitions from q_1.

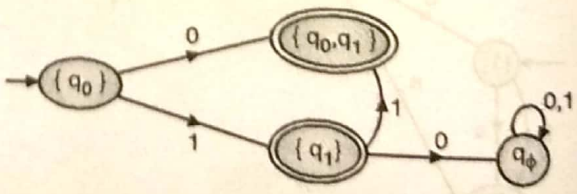


Fig. Ex. 2.2.29(a)

Scanned by CamScanner

Step 3 : Transitions from $\{q_0, q_1\}$

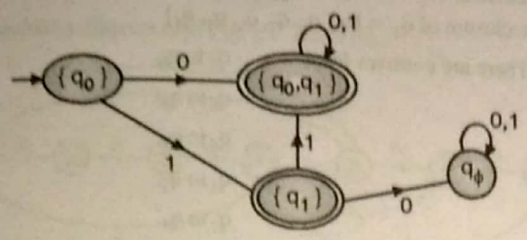


Fig. Ex. 2.2.29(b)

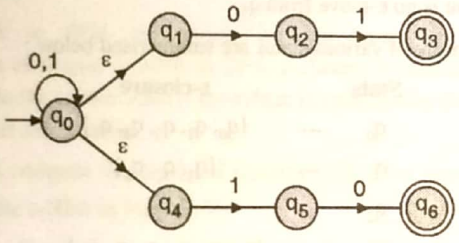
$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$$
$$= \{q_0, q_1\} \cup \phi = \{q_0, q_1\}$$
$$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$$
$$= \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}$$

Syllabus Topic : NFA with ε

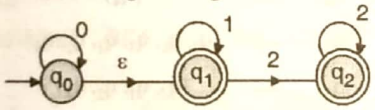
2.2.4 NFA with ε-Transitions

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

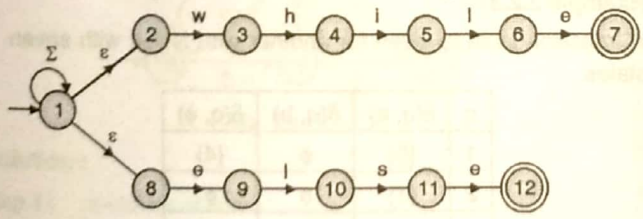
- When finding the string described by a path containing arc with label ε, the ε symbols are discarded.
- The use of ε-transition may simplify a transition graph by reducing number of labelled arcs.



(a) Strings ending in 01 or 10



(b) Starting with zero or more 0's followed by zero or more 1's and ending in 01



(c) Recognizing reverse words 'while' or 'else'

Fig. 2.2.12 : Few sample ε-NFAs

2.2.4(a) Equivalence of ε-NFA and NFA

An NFA with ε-move can be converted into an equivalent NFA without ε-move. We can always find the equivalence between systems with ε-move and without ε-move. This can be understood with the help of an example.

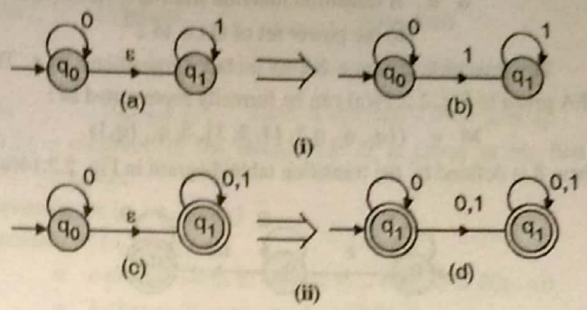


Fig. 2.2.13 : NFA with ε-move converted to NFA without ε-move

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

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

(2) If q_j is a final state then make q_i as a final state.

Example 2.2.30

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

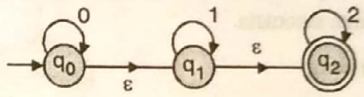


Fig. Ex. 2.2.30 : NFA with ε-move

Solution :

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

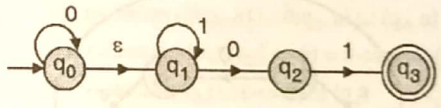


Fig. Ex. 2.2.30(a) : ε-move between q_1 and q_2 is removed

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

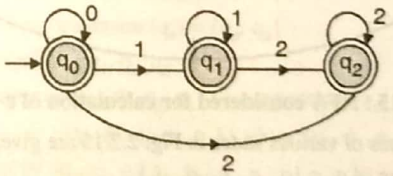


Fig. Ex. 2.2.30(b) : Final NFA without ε-move after removing ε-move between q_0 and q_1

2.2.4(b) 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

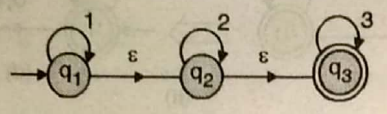
Scanned by CamScanner

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. 2.2.14(a) can be formally represented as :

$$M = (\{q_1, q_2, q_3\}, \{1, 2, 3\}, \delta, q_1, \{q_3\})$$

where δ is defined by the transition table/diagram in Fig. 2.2.14(b).



(a)

	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	$\{q_3\}$

(b)

Fig. 2.2.14 : An NFA used for formal notation

2.2.4(c) ε-Closures

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.

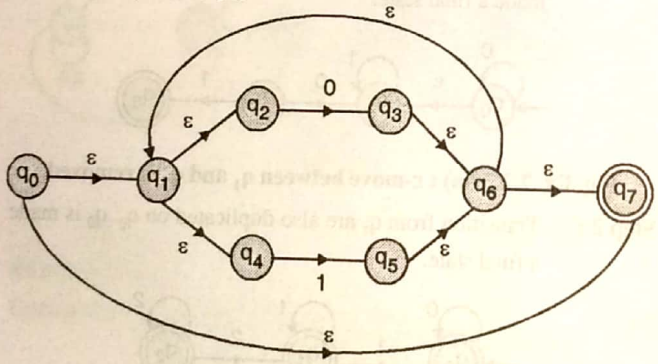


Fig. 2.2.15 : NFA considered for calculation of ε-closures

ε-closures of various states in Fig. 2.2.15 are given below :

ε-closure of $q_0 = \{q_0, q_1, q_2, q_4, q_7\}$

There are ε-moves from q_0 to q_1,
q_0 to q_7,
q_1 to q_2,
q_1 to q_4.

ε-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 2.2.31

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 $\delta^*(1, ba)$

Scanned by CamScanner

Solution:

(a) Transition diagram is in Fig. Ex. 2.2.31(a).

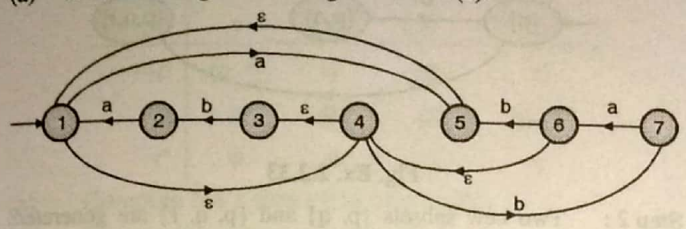


Fig. Ex. 2.2.31(a) : State transition diagram

(b) Calculate δ*(1,ba)

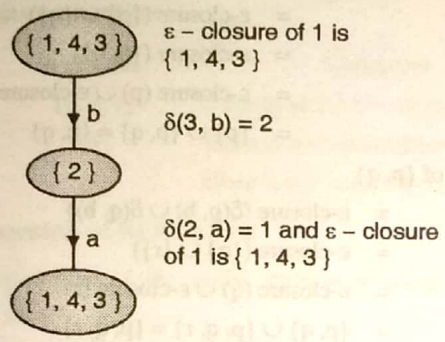


Fig. Ex. 2.2.31(b)

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

2.2.4(d) 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. 2.2.15.

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

Example 2.2.32

Find an equivalent DFA for the ε-NFA given in Fig. Ex. 2.2.32.

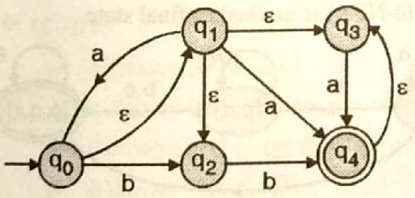


Fig. Ex. 2.2.32

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.

ε-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 $\{q_0, q_1, q_2, q_3\}$

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

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

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

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

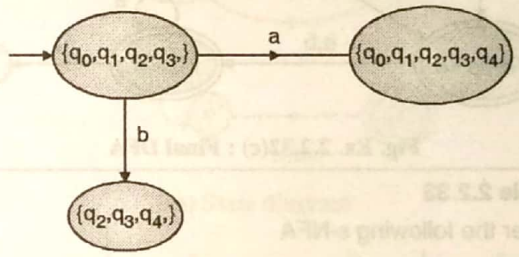


Fig. Ex. 2.2.32(a)

Step 3 : Two new subsets $\{q_2, q_3, q_4\}$ and $\{q_0, q_1, q_2, q_3, q_4\}$ have been generated. Their successors are calculated.

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 $\{q_4\}$

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

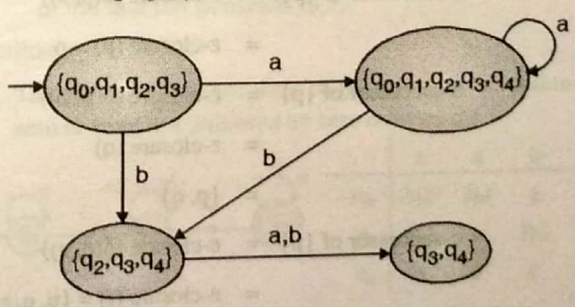


Fig. Ex. 2.2.32(b)

Scanned by CamScanner

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

a-successor of $\{q_3, q_4\}$

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

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

$= \{q_3, q_4\}$

b-successor of $\{q_3, q_4\}$

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

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

No further subsets are generated. Every subset containing q_4 (final state in Fig. Ex. 2.2.32) is marked as final state.

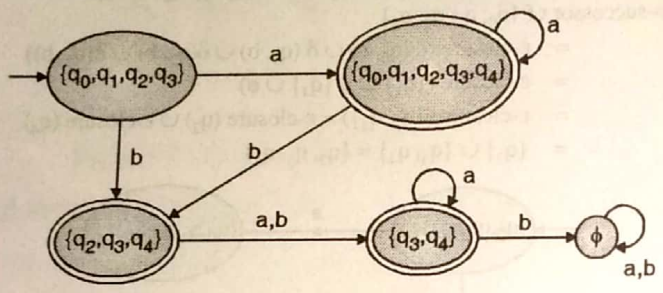


Fig. Ex. 2.2.32(c) : Final DFA

Example 2.2.33

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

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

b-successor of {p} $= \varepsilon\text{-closure}(\delta(p, b))$

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

$= \{p, q\}$

c-successor of {p} $= \varepsilon\text{-closure}(\delta(p, c))$

$= \varepsilon\text{-closure}(r) = \{p, q, r\}$

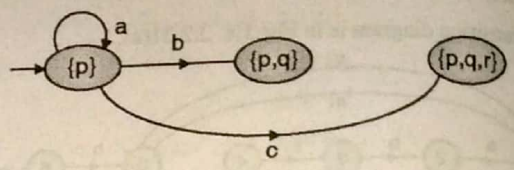


Fig. Ex. 2.2.33

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

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

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

$= \varepsilon\text{-closure}(\{p, q\})$

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

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

b-successor of {p, q}

$= \varepsilon\text{-closure}(\delta(p, b) \cup \delta(q, b))$

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

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

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

c-successor {p, q}

$= \varepsilon\text{-closure}(\delta(p, c) \cup (q, c))$

$= \varepsilon\text{-closure}(\{r\} \cup \phi) = \{p, q, r\}$

a-successor of {p, q, r}

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

$= \{p, q\} \cup \varepsilon\text{-closure}(r)$

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

b-successor of {p, q, r}

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

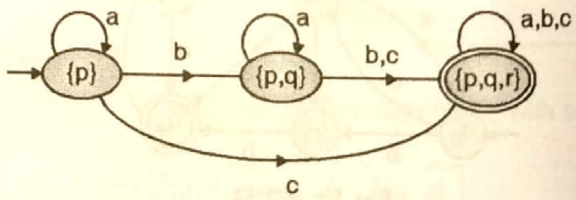
$= \{p, q, r\}$

c-successor of {p, q, r}

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

$= \{p, q, r\} \cup \varepsilon\text{-closure}(p) = \{p, q, r\}$

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



(a) State transition diagram of final DFA

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

(b) State transition table of final DFA

Fig. Ex. 2.2.33

Scanned by CamScanner

Example 2.2.34

Consider the following ε-NFA.

	ε	a	b	c
→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 of a DFA.

Solution :

(a) ε-closure of states

State	ε-closure	Comment
p	{p, q, r}	There is an ε-move from p to {q, r}
q	{q}	There is no ε-move from q.
r	{r}	There is no ε-move from r.

(b) Conversion of NFA to DFA

Step 1 : ε-closure of the initial state p is taken as the first subset.

ε-closure (p) = {p, q, r}

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

a-successor of {p, q, r}

= ε-closure (δ(p, q, r}, a))

= ε-closure (δ(p, a) ∪ δ(q, a) ∪ δ(r, a))

= ε-closure (ϕ ∪ {p} ∪ ϕ)

= ε-closure ({p}) = {p, q, r}

b-successor of {p, q, r}

= ε-closure [δ({p, q, r,}, b)]

= ε-closure (δ(p, b) ∪ δ(q, b) ∪ δ(r, b))

= ε-closure ({q} ∪ {r} ∪ ϕ)

= ε-closure ({q, r})

= ε-closure (q) ∪ ε-closure (r) = {q} ∪ {r}

= {q, r}

c-successors of {p, q, r}

= ε-closure (δ({p, q, r}, c))

= ε-closure (δ(p, c) ∪ (q, c) ∪ δ(r, c))

= ε-closure ({r} ∪ {p, q} ∪ ϕ)

= ε-closure ({p, q, r})

= ε-closure (p) ∪ ε-closure (q) ∪ ε-closure (r)

= {p, q, r} ∪ {q} ∪ {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 (δ{q, r}, a))

= ε-closure (δ(q, a) ∪ δ(r, a))

= ε-closure ({p} ∪ ϕ)

= ε-closure (p)

= {p, q, r}

b-successors of {q, r}

= ε-closure (δ({q, r}, b))

= ε-closure (δ(q, b) ∪ δ(r, b))

= ε-closure ({r} ∪ ϕ)

= ε-closure ({r}) = {r}

c-successor of {q, r}

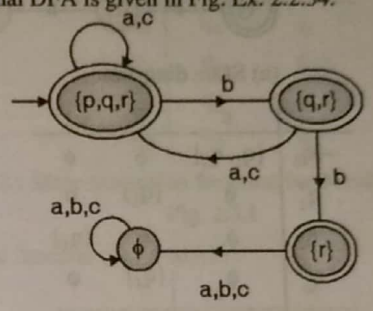
= ε-closure (δ({q, r}, c))

= ε-closure (δ(q, c) ∪ δ(r, c))

= ε-closure ({p, q} ∪ ϕ)

= {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 in Fig. Ex. 2.2.34.



(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. 2.2.34 : Final DFA for Example 2.2.34

Example 2.2.35

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

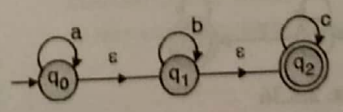
(a) The set of strings consisting of zero or more a's followed by zero or more b's, followed by zero or more c's.

(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
→q_0	{q_1}	{q_0}	ϕ	ϕ
q_1	{q_2}	ϕ	{q_1}	ϕ
q_2^*	ϕ	ϕ	ϕ	{q_2}

Fig. Ex. 2.2.35 : ε-NFA for Example 2.2.35(a)

Scanned by CamScanner

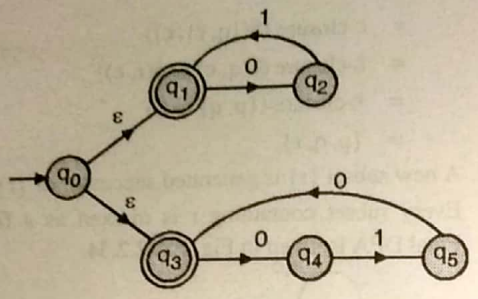
Meaning of various states :

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

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

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

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



(a) State diagram

	ε	0	1
$\rightarrow q_0$	$\{q_1, q_3\}$	ϕ	ϕ
q_1^*	ϕ	$\{q_2\}$	ϕ
q_2	ϕ	ϕ	$\{q_1\}$
q_3^*	ϕ	$\{q_4\}$	ϕ
q_4	ϕ	ϕ	$\{q_5\}$
q_5	ϕ	$\{q_3\}$	ϕ

(b) State table

Fig. Ex. 2.2.35 : ε-NFA for Example 2.2.35(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.

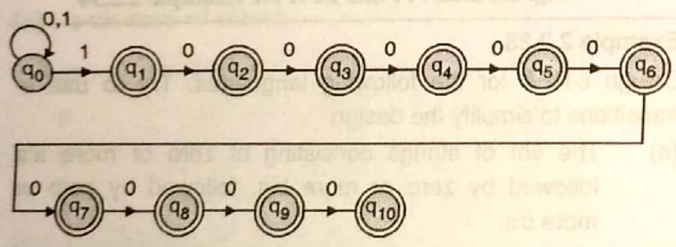


Fig. Ex. 2.2.35(c) : NFA for Example 2.2.35(c)

- 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 10th position from the end contain 1.

Example 2.2.36

Convert the following NFA-ε to its equivalent DFA.

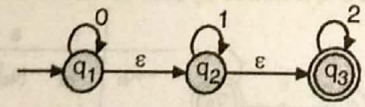


Fig. Ex. 2.2.36

Solution :

Step 1 : Finding ε-closure

State	ε-closure
q_1	$\{q_1, q_2, q_3\}$
q_2	$\{q_2, q_3\}$
q_3	$\{q_3\}$

Step 2 : Initial state is taken as the ε-closure of q_1. Transitions are written from it.

$\delta(\{q_1, q_2, q_3\}, 0) =$ ε-closure $(\delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0))$

$=$ ε-closure $(q_1 \cup \phi \cup \phi) = \{q_1, q_2, q_3\}$

$\delta(\{q_1, q_2, q_3\}, 1) =$ ε-closure $(\delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1))$

$=$ ε-closure $(\phi \cup q_2 \cup \phi) = \{q_2, q_3\}$

$\delta(\{q_1, q_2, q_3\}, 2) =$ ε-closure $(\delta(q_1, 2) \cup \delta(q_2, 2) \cup \delta(q_3, 2))$

$=$ ε-closure $(\phi \cup \phi \cup q_3) = \{q_3\}$

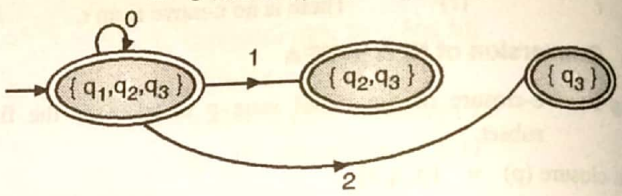


Fig. Ex. 2.2.36(a)

Step 3 : Transitions from $\{q_2, q_3\}$ and $\{q_3\}$ are calculated.

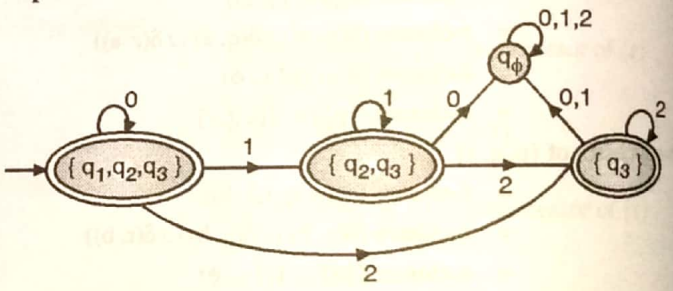


Fig. Ex. 2.2.36(b)

Example 2.2.37

Design an FSM to accept those strings having 101 or 110 as substring.

Solution :

The required FSM is given below.

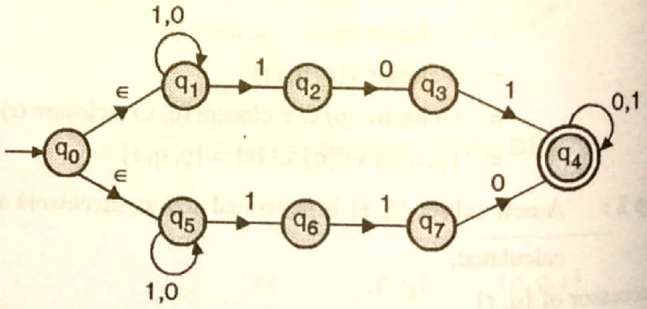


Fig. Ex. 2.2.37

- The path from $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$ is for the substring 101.
- The path from $q_0 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_4$ is for the substring 110.

Scanned by CamScanner

2.2.5 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.

2.3 Finite Automata as Output Devices

Finite automata that we have discussed so far have a limited capability of either accepting a string or rejecting a string. Acceptance of a string was based on the reachability of a machine from starting state to final state. Finite automata can also be used as an output device.

- Such machines do no have final state/states.
- Machine generates an output on every input. The value of the output is a function of current state and the current input.

Such machines are characterised by two behaviours :

1. **State transition function (δ) :** State transition function (δ) is also known as STF.
2. **Output function (λ) :** 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.

   $\lambda : Q \rightarrow O$

2.3.1 A Sample Mealy Machine

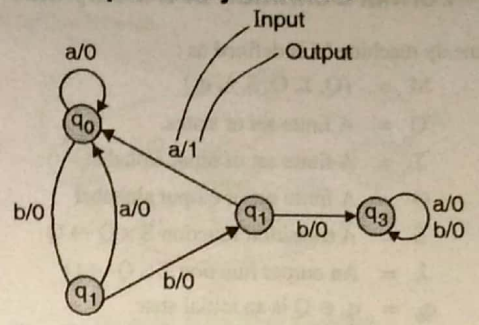


Fig. 2.3.1 : State diagram of a Mealy machine

State transition function (δ) (or STF) :

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

Fig. 2.3.2 : State transition function for Mealy machine of Fig. 2.3.1

Output function (λ) (or MAF) :

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

Fig. 2.3.3 : Output function for mealy machine of Fig. 2.3.1

State table for both δ and λ (both STF and MAF) :

	A	b
$\rightarrow q_0$	$q_0/0$	$q_1/0$
q_1	$q_0/0$	$q_2/0$
q_2	$q_0/1$	$q_3/0$
q_3	$q_3/0$	$q_3/0$

Output

Next state

Fig. 2.3.4 : State table depicting both transition and output behavior of mealy machine of Fig. 2.3.1

- An arc from state q_i in a mealy machine is associated with :
  1. Input alphabet $\in \Sigma$
  2. An output alphabet $\in$ O.
- An arc marked as 'a/0' in Fig. 2.3.1 implies that :
  1. a is in input
  2. 0 is an output.
- State transition behavior and output behavior of a mealy machine can be shown separately as in Fig. 2.3.2 and 2.3.3; or they can be combined together as in Fig. 2.3.4.

Scanned by CamScanner

2.3.2 Formal Definition of a Mealy Machine

A mealy machine M is defined as :

$$M = (Q, \Sigma, O, \delta, \lambda, q_0)$$

Where,
- Q = A finite set of states.
- Σ = A finite set of input alphabet
- O = A finite set of output alphabet
- δ = A transition function $\Sigma \times Q \rightarrow Q$
- λ = An output function $\Sigma \times Q \rightarrow O$
- q_0 = $q_0 \in Q$ is an initial state.

2.3.3 A Sample Moore Machine

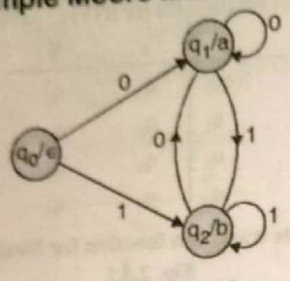


Fig. 2.3.5 : State diagram of a Moore machine

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

Fig. 2.3.6 : State transition function for Moore machine of Fig. 2.3.5

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

Fig. 2.3.7 : Output function for Moore machine of Fig. 2.3.5

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

Fig. 2.3.8 : State table depicting both transition and output behavior of Moore machine of Fig. 2.3.5

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

2.3.4 Formal Definition of a Moore Machine

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 2.3.1

Give Mealy and Moore machine for the following :

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

Solution :

(a) Mealy machine :

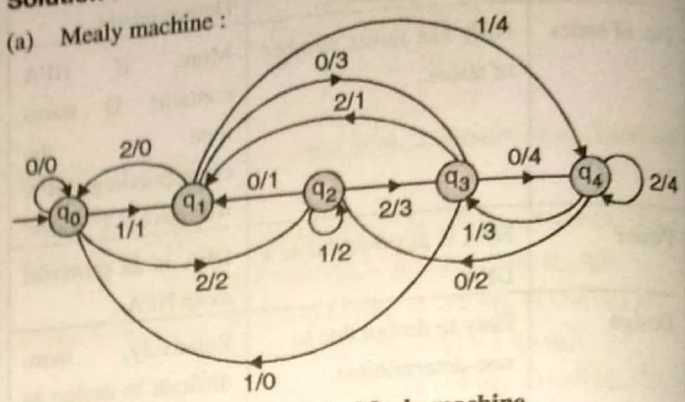


Fig. Ex. 2.3.1(a) : Mealy machine

Meaning of various states is :

q_0 – Running remainder is 0

q_1 – Running remainder is 1

q_2 – Running remainder is 2.

q_3 – Running remainder is $3 = (10)_3$

q_4 – Running remainder is $4 = (11)_3$

In decimal system

In ternary system

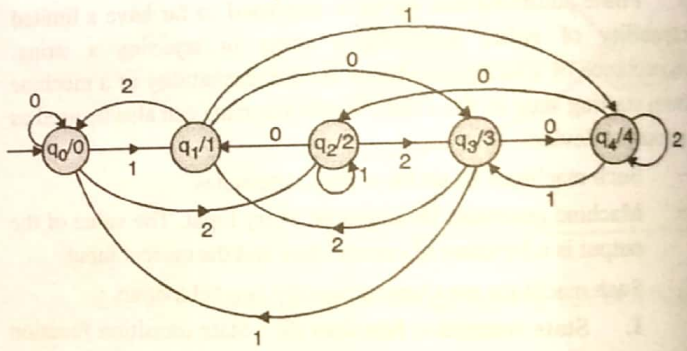


Fig. Ex. 2.3.1(b) : Moore machine

Example 2.3.2 MU - May 2015, 5 Marks

Draw the transition matrix and transition graph for a binary adder.

OR

Design Mealy machine 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 :

$$\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}$$

Scanned by CamScanner

– Sum of carry will be calculated in a usual way.

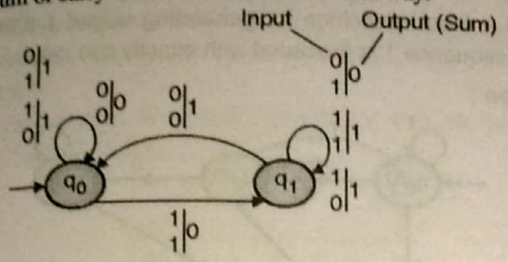


Fig. Ex. 2.3.2(a) : Transition graph

	$\binom{0}{0}$	$\binom{0}{1}$	$\binom{1}{0}$	$\binom{1}{1}$	
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. 2.3.2(b) : Transition matrix

Example 2.3.3

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.

Σ = {0, 1, 2}

q_0 = current remainder is 0

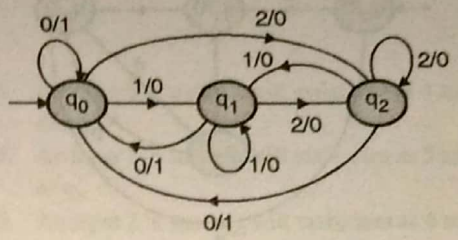
q_1 = current remainder is 1

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



(a) State transition diagram

Current state	Input		
	0	1	2
→q_0	q_0/1	q_1/0	q_2/0
q_1	q_0/1	q_1/0	q_2/0
q_2	q_0/1	q_1/0	q_2/0

(b) State transition table

	0	1	2
→q_0	1	0	0
q_1	1	0	0
q_2	1	0	0

(c) MAF

	0	1	2
→q_0	q_0	q_1	q_2
q_1	q_0	q_1	q_2
q_2	q_0	q_1	q_2

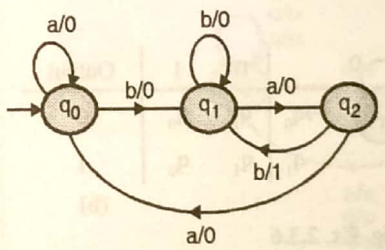
(d) STF

Fig. Ex. 2.3.3

Example 2.3.4

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

Solution :



(a) State diagram

	a	b
→q_0	q_0, 0	q_1, 0
q_1	q_2, 0	q_1, 0
q_2	q_0, 0	q_1, 1

(b) State table

Fig. Ex. 2.3.4

Meaning of various states :

q_0 – Start state

q_1 – Previous character is b

q_2 – Preceding two character are ba

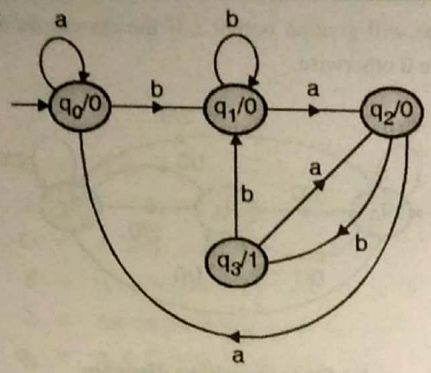
A transition from q_2 to q_1 will make the preceding three characters as bab and hence the output 1.

Example 2.3.5

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

Scanned by CamScanner

Solution :



(a) State diagram

Present state	Input a	Input 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. 2.3.5 : Moore machine for the Example 2.3.5

Meaning of various states :

q_0 – Start state

q_1 – Previous character is b

q_2 – Preceding two character are ab

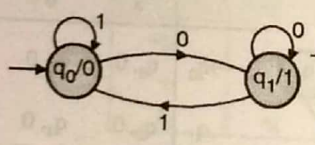
q_3 – Previous character is b

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

Example 2.3.6

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

Solution :



(a)

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

(b)

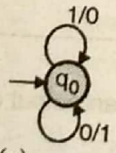
Fig. Ex. 2.3.6

- Next input as 0, sends the machine to state q_1 with output of 1. Complement of 0 is 1.
- Next input as 1, sends the machine to state q_0 with output of 0. Complement of 1 is 0.

Example 2.3.7

Design a Mealy machine to generate 1's complement of a given binary number.

Solution :



(a)

	0	1
$\rightarrow q_0$	q_0, 1	q_0,0

(b)

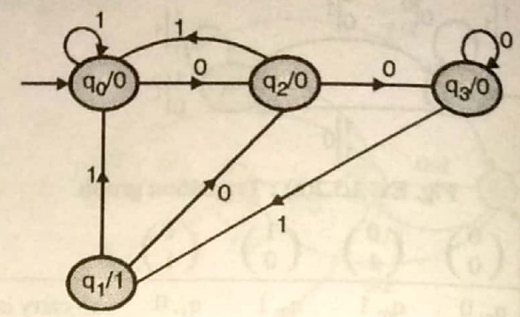
Fig. Ex. 2.3.7

- Every input is simply complemented in state q_0.

Example 2.3.8

Design a Moore machine for generating output 1 if input of binary sequence 1 is preceded with exactly two zeros.

Solution :



(a)

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

(b)

Fig. Ex. 2.3.8

Meaning of various states :

q_0 – initial state, associated with output 0

q_2 – preceding symbol is 0 of sequence 001, associated with output 0.

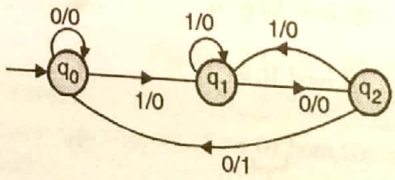
q_3 – preceding two symbols as 00 of sequence 001, associated with output 0.

q_1 – preceding three symbols as 001 of sequence 001, associated with output 1.

Example 2.3.9

Design a mealy machine for a binary input sequence such that if the sequence ends with 100 the output is 1 otherwise output is 0.

Solution :



(a) State diagram

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

(b) State table

Fig. Ex. 2.3.9

Meaning of various states :

q_0 – start state

q_1 – previous symbol is 1

q_2 – preceding two symbols are 10

A transition from q_2 to q_0 will make the preceding three symbol as 100 and hence the output 1.

Example 2.3.10

Design a mealy machine for incrementing the value of any binary number by one. The output should also be a binary number with value one more than the given number.

Scanned by CamScanner

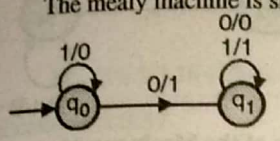
Solution :

When a binary number is incremented then trailing 1's become 0's and the first zero immediately before trailing 1's become 1.

For example, if we add 1 to 0101111 then the incremented value will be 0110000.

```
  0 1 0 1 1 1 1
  +           1
  ---------------
  0 1 1 0 0 0 0
```

The mealy machine is shown in Fig. Ex. 2.3.10.



Input should be applied, starting from the rightmost symbol.

Fig. Ex. 2.3.10

	0	1
$\rightarrow q_0$	q_1, 1	q_0, 0
q_1	q_1, 0	q_1, 1

Example 2.3.11

Design a Moore machine for the following problem :

Input alphabet = {0, 1, 2}

Output alphabet = {a, b, c}

If the sum of 3 consecutive number is 0, 1, 4; output a.

If the sum is 2, 3; output b.

If the sum is 5, 6; output c.

Solution :

Sum of three consecutive symbols will depend on :

1. Preceding two symbols
2. Next input.

Preceding two symbols could be :

Symbols		State
00	—	q_{00}
01	—	q_{01}
02	—	q_{02}
10	—	q_{10}
11	—	q_{11}
12	—	q_{12}
20	—	q_{20}
21	—	q_{21}
22	—	q_{22}

The next state and the output is given below :

1. An input 0 in state q_{00} will make sum as 0 and next state as q_{00}
2. An input 1 in state q_{00} will make sum as 1 and next state as q_{01}
3. An input 2 in state q_{00} will make sum as 2 and next state as q_{02}
4. An input 0 in state q_{01} will make sum as 1 and next state as q_{10}
5. An input 1 in state q_{01} will make sum as 2 and next state as q_{11}
6. An input 2 in state q_{01} will make sum as 3 and next state as q_{12}

:

:

:

25. An input 0 in state q_{22} will make sum as 4 and next state as q_{20}
26. An input 1 in state q_{22} will make sum as 5 and next state as q_{21}
27. An input 2 in state q_{22} will make sum as 6 and next state as q_{22}

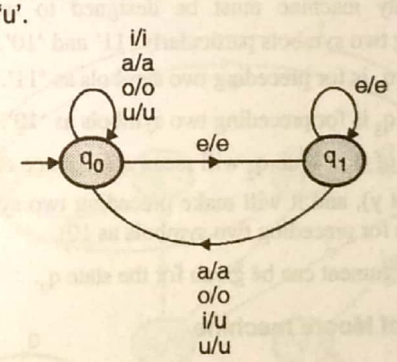
State transition table,

	0	1	2
$\rightarrow q_{00}$	q_{00}/a	q_{01}/a	q_{02}/b
q_{01}	q_{10}/a	q_{11}/b	q_{12}/b
q_{02}	q_{20}/b	q_{21}/b	q_{22}/a
q_{10}	q_{00}/a	q_{01}/b	q_{02}/b
q_{11}	q_{10}/b	q_{11}/b	q_{12}/a
q_{12}	q_{20}/b	q_{21}/a	q_{22}/c
q_{20}	q_{00}/b	q_{01}/b	q_{02}/a
q_{21}	q_{10}/b	q_{11}/a	q_{12}/c
q_{22}	q_{20}/a	q_{21}/c	q_{22}/c

Example 2.3.12

Design a Mealy machine that will read sequences made up of vowels of English language, it will give output in same sequences, except in cases where a 'i' follows 'e', it will be changed to 'u'.

Solution :



(a) State diagram

Current state	Next state output				
	a	e	i	o	u
$\rightarrow q_0$	q_0, a	q_1, e	q_0, i	q_0, o	q_0, u
q_1	q_0, a	q_1, e	q_0, u	q_0, o	q_0, u

(b) State table

Fig. Ex. 2.3.12

- Machine must convert every occurrence of ei into eu. In other cases, output will be same as input.
- State q_1 implies that the preceding character is e.
- An input i in stat q_1 will generate an output u.

Scanned by CamScanner

Example 2.3.13 **MU - May 2014, 10 Marks**

Design a Mealy machine over the alphabet {0, 1} which outputs EVEN, ODD according to the number of 1's encountered as even or odd.

Solution :

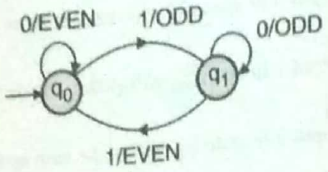


Fig. Ex. 2.3.13

Example 2.3.14

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

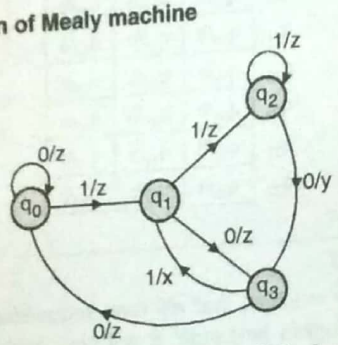


Fig. Ex. 2.3.14 : State diagram of Mealy machine

- 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

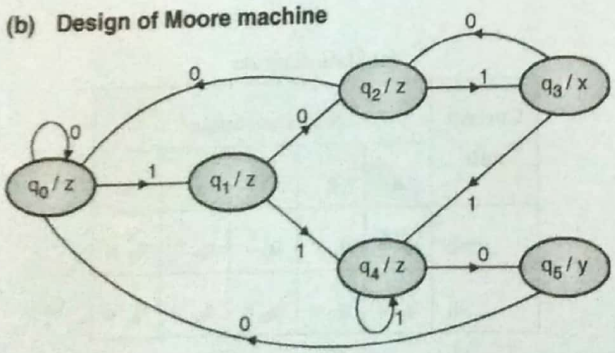


Fig. Ex. 2.3.14(b) : State diagram of the Moore machine

Example 2.3.15

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

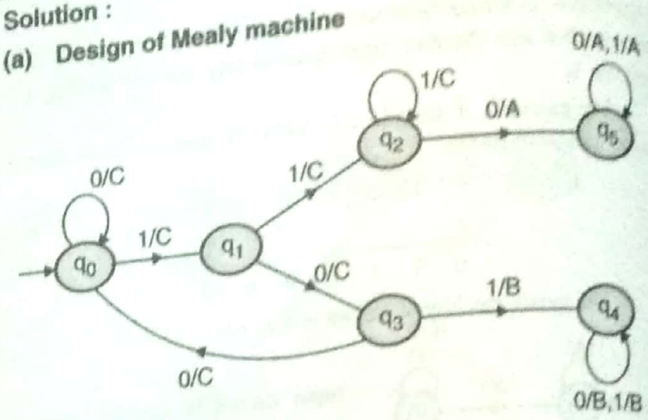


Fig. Ex. 2.3.15(a) : State diagram of the Mealy machine

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

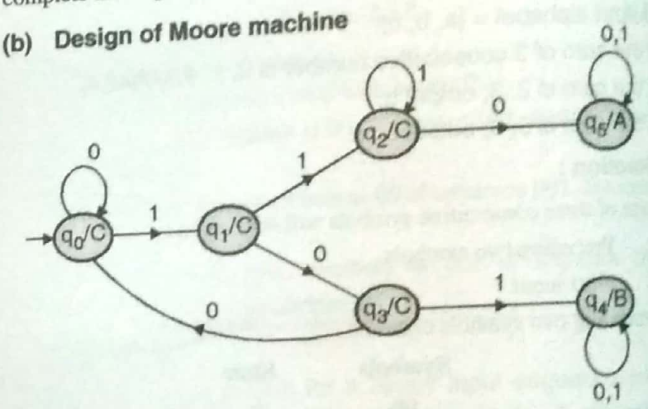


Fig. Ex. 2.3.15(b) : State diagram of the Moore machine

Example 2.3.16

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 ith 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

Scanned by CamScanner

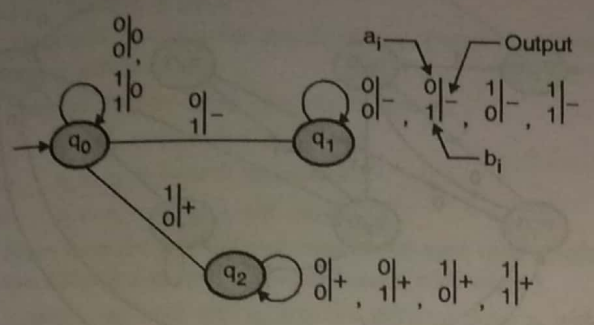


Fig. Ex. 2.3.16 : State diagram of the Mealy machine

Example 2.3.17

Design a Moore Machine for checking divisibility by 3 of a given decimal number (residue of 3).

Solution : The decimal system has 10 alphabet.

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The running remained when a given decimal number is divided by 3 could be :

0 → associated state is q_0.

1 → associated state is q_1

2 → associated state is q_2.

The required Moore machine is given Fig. Ex. 2.3.17.

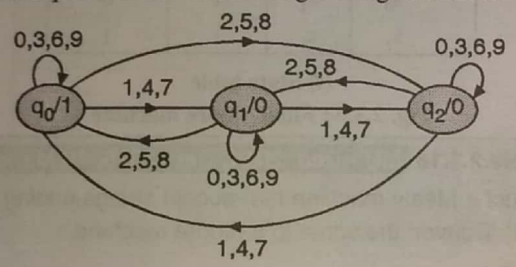


Fig. Ex. 2.3.17

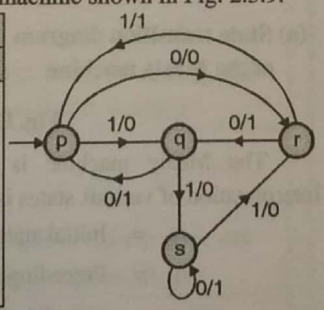
Syllabus Topic : Mealy and Moore Machine Conversion

2.3.5 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. 2.3.9.

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. 2.3.9 : Mealy machine considered for conversion into an equivalent Moore machine

Step 1 : Splitting of states

It may be necessary to split some of the states of a Mealy machine to get an equivalent Moore machine. Splitting is based on incoming lines into a state and the associated outputs. We must split a state to handle different output values associated with incoming lines.

- There are two incoming lines into the state p.
  - (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.

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

- There are two incoming lines into the state r.
  - (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.

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

Step 2 : Drawing of an equivalent Moore machine.

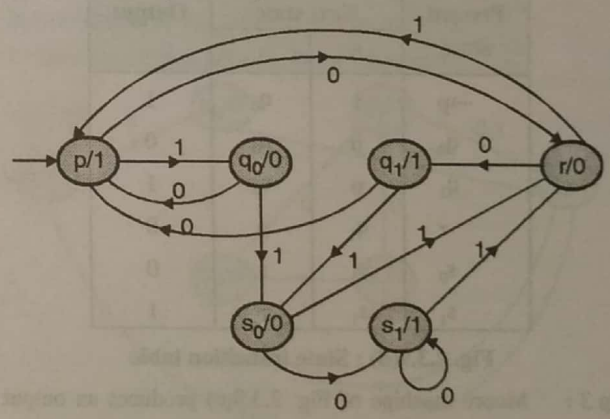


Fig. 2.3.9(c) : State diagram of Moore machine

- Every transition into q with output 0 is directed to q_0.
- Every transition into q with output 1 is directed to q_1.
- Every transition into s with output 0 is directed to s_0.
- Every transition into s with output 1 is directed to s_1.
- Every transition from q is duplicated both from q_0 and q_1.
- Every transition from s is duplicated both from s_0 and s_1.

Transitions for Mealy Machine	Equivalent transitions for Moore Machine
δ (p, 0) = r, 0	δ (p, 0) = r

Scanned by CamScanner

Transitions for Mealy Machine	Equivalent transitions for Moore Machine
$\delta(p, 1) = q, 0$	$\delta(p, 1) = q_0$ 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. 2.3.9(d).

Present state	Next state		Output
	0	1	
$\rightarrow p$	r	q_0	1
q_0	p	s_0	0
q_1	p	s_0	1
r	q_1	p	0
s_0	s_1	r	0
s_1	s_1	r	1

Fig. 2.3.9(d) : State transition table

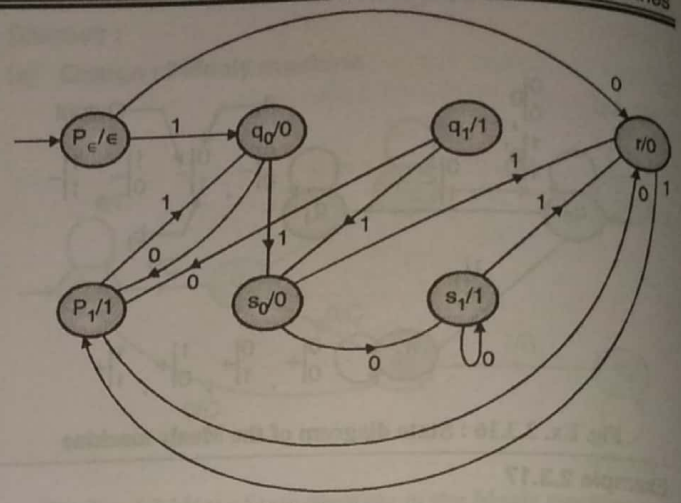
Step 3 : Moore machine of Fig. 2.3.9(c) produces an output 1 without any input. Thus, the Moore machine gives an output 1 on a zero length sequence. But the Mealy machine will produce no output for a zero length sequence.

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

p_ε – Giving no output

p_1 – Giving an output 1.

Final Moore machine is drawn in Fig. 2.3.9(e).



(e) 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

(f) State table

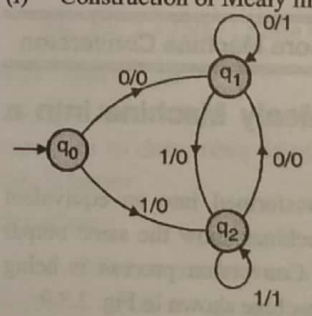
Fig. 2.3.9 : Final Moore machine

Example 2.3.18 MU - Dec. 2014, Dec. 2016, 10 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. 2.3.18

The Mealy machine is shown in the Fig. Ex. 2.3.18. 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.

Scanned by CamScanner

(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.

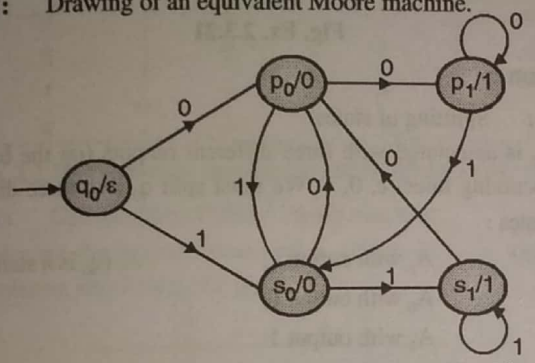
(b) q_1 to q_1 on input 0, with output 1.

(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 0	Input 1	Output
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. 2.3.18

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 2.3.19

Consider the following Mealy machine. Convert it to Moore.

Present state	Next state a = 0 State	Next state a = 0 Output	Next state a = 1 State	Next state a = 1 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.

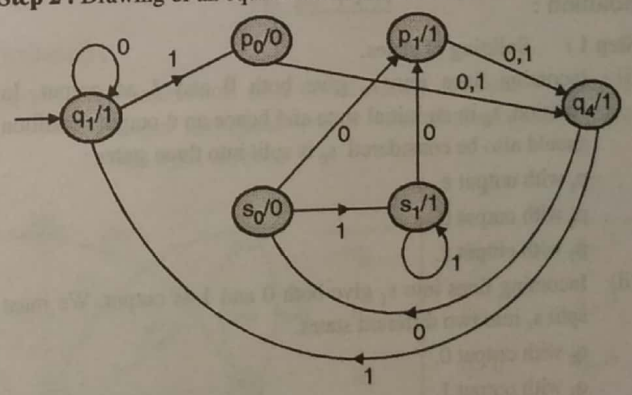


Fig. Ex. 2.3.19(a) : Moore machine

Step 3 : Moore machine of Fig. Ex. 2.3.19(a) produces an output of 1 without any input. But the Mealy machine given in Example 2.3.19 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. 2.3.19(b) and 2.3.19(c).

Scanned by CamScanner

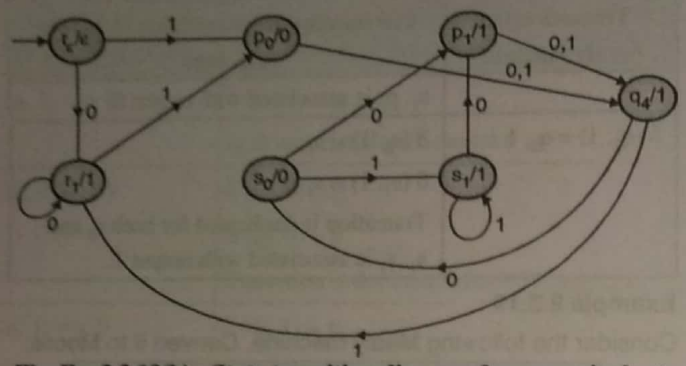


Fig. Ex. 2.3.19(b) : State transition diagram for an equivalent Moore machine

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. 2.3.19(c) : State transition table for an equivalent Moore machine

Example 2.3.20

Convert the Mealy machine shown in Fig. Ex.2.3.20 to Moore machine.

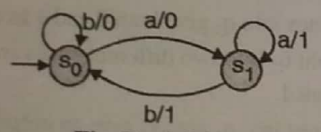


Fig. Ex. 2.3.20

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.

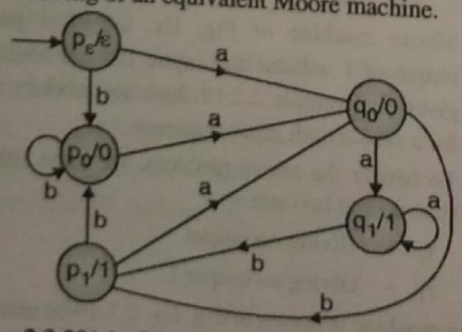


Fig. Ex. 2.3.20(a) : Moore machine for Example 2.3.20

– 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 2.3.21

Convert the Mealy machine shown in Fig. Ex.2.3.21 to its equivalent Moore machine.

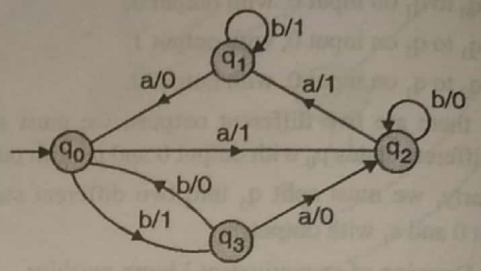


Fig. Ex. 2.3.21

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.

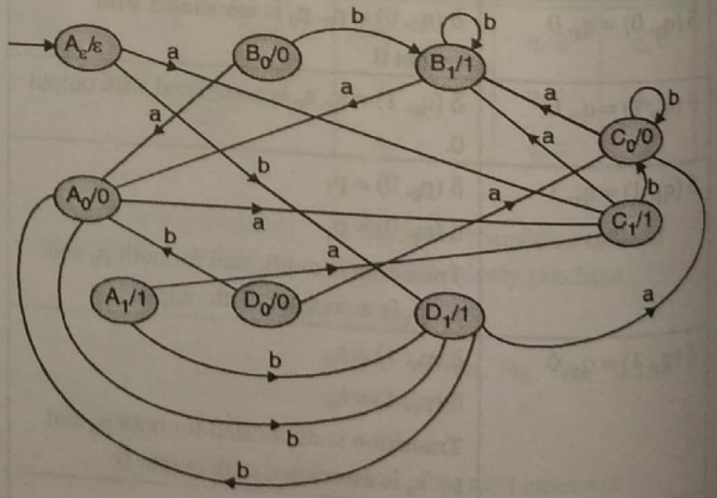


Fig. Ex. 2.3.21(a) : Mealy machine for Example 2.3.21

Scanned by CamScanner

2.3.6 Conversion of a Moore Machine into a Mealy Machine

A Moore machine can be transformed to a corresponding Mealy machine in two steps. These steps are :

1. Construction of a trivial Mealy machine -
   By moving output associated with a state to transitions entering into that state.
2. Minimization of the trivial Mealy machine obtained in step 1.

These two steps are being explained with the help of a Moore machine shown in Fig. 2.3.10(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. 2.3.10(a) : Moore machine considered for conversion

Step 1 : Construction of trivial Mealy machine.

(i) Moving the output associated p, which is 0, to transitions entering into state p, we get :

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

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

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

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

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

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

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

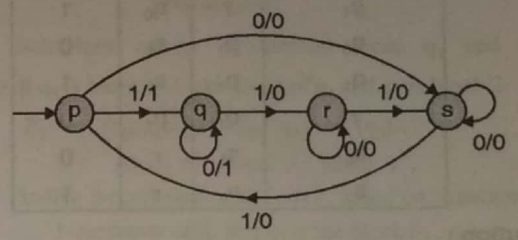
Fig. 2.3.10(b) : A trivial Mealy machine

Step 2 : The trivial Mealy machine obtained in Fig. 2.3.10(b) can not be minimized further.

Final Mealy machine is given in Fig. 2.3.10(c) and 2.3.10(d).

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

(c) Transition table for final Mealy machine



(d) Transition diagram for final Mealy machine

Fig. 2.3.10

Example 2.3.22

Change the given Moore machine into Mealy machine.

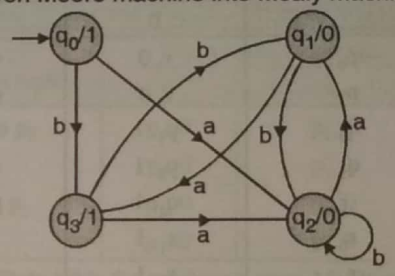
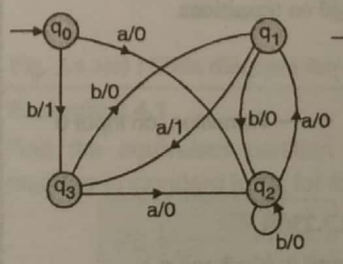


Fig. Ex. 2.3.22

Solution :

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



(a)

	a	b
→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

(b)

Fig. Ex. 2.3.22

Step 2 : Minimization of Mealy machine obtained in Fig. Ex. 2.3.22(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.

Scanned by CamScanner

∴ Behavior of q_2 is different from q_3.

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

Mealy machine given in Fig. Ex. 2.3.22(a) is the final Mealy machine.

Example 2.3.23

Change the given Moore machine into Mealy machine.

Present state	Next state		Output
	0	1	
$\rightarrow p_0$	r	q_0	ε
p_1	r	q_0	1
q_0	p_1	s_0	0
q_1	p_1	s_0	1
r	q_1	p_1	0
s_0	s_1	r	0
s_1	s_1	r	1

Solution :

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

Present state	(Next state, Output)	
	0	1
$\rightarrow p_0$	r, 0	q_0, 0
p_1	r, 0	q_0, 0
q_0	p_1, 1	s_0, 0
q_1	p_1, 1	s_0, 0
r	q_1, 1	p_1, 1
s_0	s_1, 1	r, 0
s_1	s_1, 1	r, 0

Fig. Ex. 2.3.23(a) : A trivial Mealy machine

Step 2 : Minimization of the trivial Mealy machine obtained in Fig. Ex. 2.3.23(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.

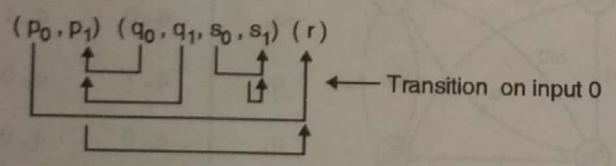


Fig. Ex. 2.3.23(b)

On input 0, q_0 and q_1 are mapped to block (p_0, p_1)

On input 0, s_0 and s_1 are mapped to block (q_0, q_1, s_0, s_1)

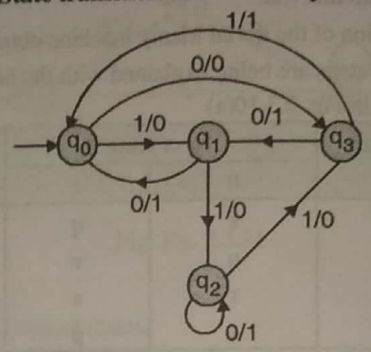
∴ Behavior of q_0, q_1 is different from s_0, s_1.

$$P_2 = (p_0, p_1)(q_0, q_1)(s_0, s_1)(r)$$

Further partitioning is not possible. The minimum state Mealy machine is given in Fig. Ex. 2.3.23(c) and 2.3.23(d).

Present state	(Next state, Output)	
	0	1
$(p_0, p_1) = \rightarrow q_0$	q_3, 0	q_1, 0
$(q_0, q_1) = q_1$	q_0, 1	q_2, 0
$(s_0, s_1) = q_2$	q_2, 1	q_3, 0
$(r) = q_3$	q_1, 1	q_0, 1

(c) State transition table of Mealy machine



(d) State transition diagram of Mealy machine

Fig. Ex. 2.3.23

Example 2.3.24

Consider the Moore machine described by the transition table given below. Construct corresponding Mealy machine.

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

Solution :

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

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

Step 2 : Minimization of the trivial Mealy machine obtained in step1.

States q_2 and q_3 are equivalent as they have the same output and transition behaviour.

The minimum state machine is given below :

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

Example 2.3.25

Convert the following Moore machine to Mealy machine.

State	Input		Output
q_0	q_1	q_3	1
q_1	q_3	q_1	0
q_2	q_0	q_3	0
q_3	q_3	q_2	1

Scanned by CamScanner

Solution :

Step 1 : Construction of trivial mealy machine by moving output associated with states on incoming transitions.

State	Input	
	a	b
q_0	$q_1/0$	$q_3/1$
q_1	$q_3/1$	$q_1/0$
q_2	$q_0/1$	$q_3/1$
q_3	$q_3/1$	$q_2/0$

Step 2 : Minimization of trivial mealy machine.

State are grouped on the basis of output.

$(q_0)\ (q_1, q_3)\ (q_2)$

The transition behaviour of q_1 and q_3 are different under the input 'b'. Hence the states q_1 and q_3 are distinguishable.

∴ The final mealy machine is same as the trivial mealy machine and it is given Fig. Ex. 2.3.25.

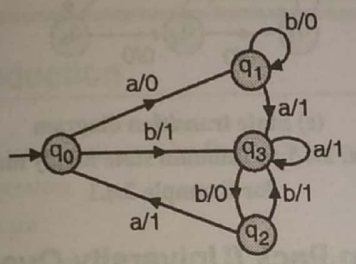


Fig. Ex. 2.3.25

2.4 Minimization of a Mealy Machine

Minimization of a Mealy machine is almost similar to minimization of a DFA.

- 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. 2.4.1 : Mealy machine being considered for minimization

Minimization process is being explained step wise for the Mealy machine given in Fig. 2.4.1.

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.

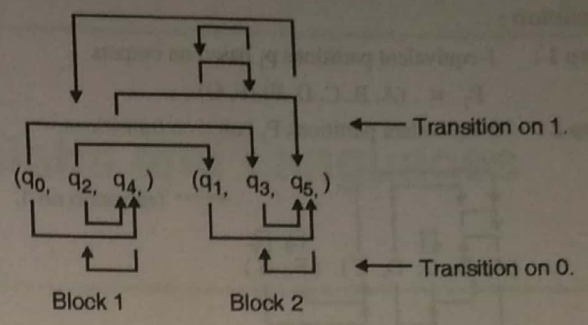


Fig. 2.4.1(a)

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. 2.4.1(b) and 2.4.1(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. 2.4.1(b) : State table for minimum state Mealy machine

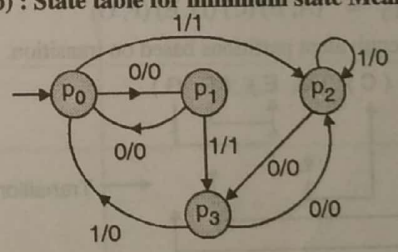


Fig. 2.4.1(c) : State diagram for minimum state Mealy machine

Example 2.4.1

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

Scanned by CamScanner

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.

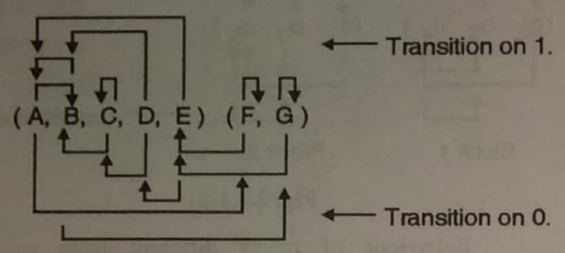


Fig. Ex. 2.4.1(a)

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)

∴ Behavior of A, B is different from C, D, E

$$P_2 = (A, B)\ (C, D, E)\ (F, G)$$

Step 3 : 3-equivalent partitions based on transition.

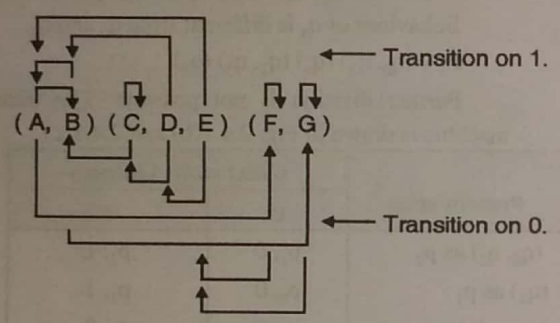


Fig. Ex. 2.4.1(b)

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.

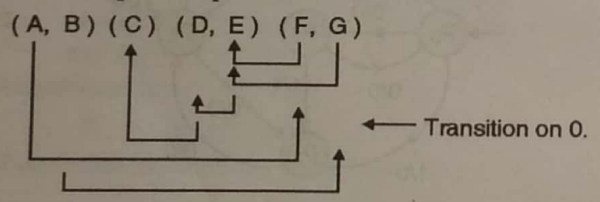


Fig. Ex. 2.4.1(c)

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.

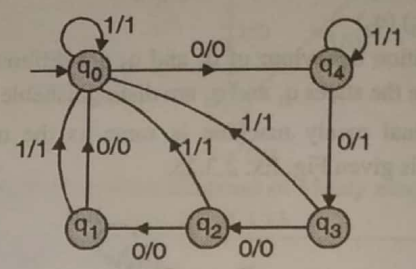
$$P_4 = (A, B)\ (C)\ (D)\ (E)\ (F, G)$$

Further division is not possible.

The minimum state Mealy machine is drawn in Fig. Ex. 2.4.1(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. 2.4.1 : Minimum state Mealy machine for Example 2.4.1

2.5 Exam Pack (University Questions)

Example 2.1.10 (5 Marks)	**(May 2014)**
Example 2.1.16 (5 Marks)	**(May 2014)**
Example 2.1.22 (5/10 Marks)	**(May 2014, May 2015)**
Example 2.1.38 (10 Marks)	**(Dec. 2016)**
Example 2.2.11 (10 Marks)	**(Dec. 2014)**
Example 2.2.14 (5 Marks)	**(Dec. 2016)**
Example 2.2.17 (10 Marks)	**(May 2015)**
Example 2.2.20 (10 Marks)	**(May 2014)**
Example 2.3.2 (5 Marks)	**(May 2015)**
Example 2.3.13 (10 Marks)	**(May 2014)**

☞ **Syllabus Topic : Mealy and Moore Machine Conversion**

Example 2.3.18 (10 Marks)	**(Dec. 2014, Dec. 2016)**

□□□

Scanned by CamScanner

CHAPTER 3

Regular Expressions and Languages

Syllabus Topics

Regular Expression, Regular Languages, RE to NFA, FA to RE, Closure properties.

Syllabus Topic : Regular Expression, Regular Languages

3.1 Introduction

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.

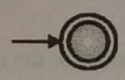
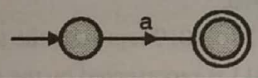
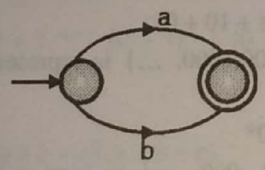
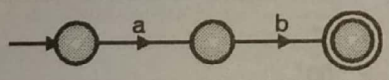
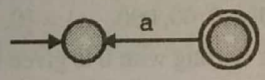
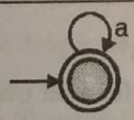
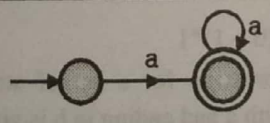
Automata	Language	Regular expression
	{ε}	R.E. = ε
	{a}	R.E. = a
	{a, b}	R.E = a + b
	{ab}	R.E. = a · b or simply ab
	ϕ	R.E. = ϕ
	{ε, a, aa, aaa, …}	R.E. = a^*
	{a, aa, aaa, …}	R.E. = aa^* or a^+

Fig. 3.1.1 Contd...

Scanned by CamScanner

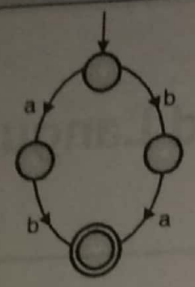
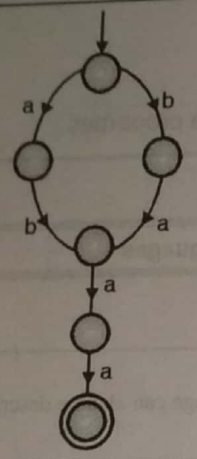
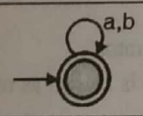
Automata	Language	Regular expression
	{ab, ba}	R.E. = ab + ba
	{abaa, baaa}	R.E. = (ab + ba) aa
	{ε, a, b, aa, ab, ba, bb, ...}	R.E. = (a + b)*

Fig. 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

Scanned by CamScanner

(i) $(a + b)^3$ stands for $(a + b)(a + b)(a + b) = aaa + aab + aba + abb + baa + bab + bba + bbb$.

Example 3.1.2

Write a regular expression to identify valid decimal integer constant for 'C' language. Justify RE with example.

Solution :

The RE is being given for an unsigned decimal integer constant.

$(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)^*$

The first term ensures that the integer constant is of non-zero length. The second term can generate any number of digits.

Any valid integer constant like 1234 will be generated by the above RE.

Example 3.1.3

Write a regular expression to search .dat file (s) having starting character "p" and ending with "zw". Justify RE with example (s).

Solution : The required regular expression in UNIX will be written as P * z w · dat

- Star matches one or more characters.
- Star(*) is a wild character.
- The corresponding command in UNIX can be written as
  $ *ls -l* p*zw·dat

Example 3.1.4 MU - Dec. 2016, 6 Marks

Give regular expressions for -

(i) Binary strings containing atleast one 11 and atleast one 00.

(ii) Strings with even number of a's

(iii) Strings in which third symbol from end is 'c' over {a, b, c}

Solution :

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

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

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

Example 3.1.5 MU - Dec. 2016, 4 Marks

Describe Regular Language for given Regular Expressions.

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

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

Solution :

(i) Any subset of even length of the string will have equal number of a's and b's.

(ii) The string starts with '1' and ends with '0' and $|\omega| \geq 5$.

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

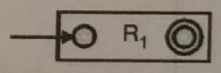
(1) $R_1 + R_2$ (2) $R_1 \cdot R_2$ (3) R_1^*

Then we can construct finite state automata (FA) for any regular expression.

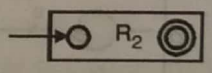
3.2.1 Composite Finite State Automata

Let us assume the existence of FAs for regular expressions R_1 and R_2.

FA for R_1 is given by

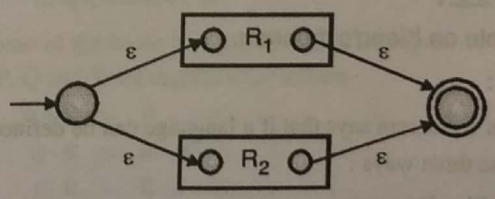


FA for R_2 is given by

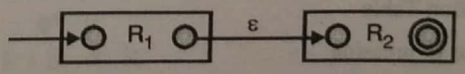


Each FA is assumed to have one final state.

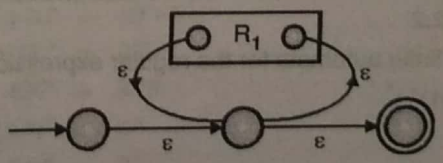
1. FA for $R_1 + R_2$ is given by



2. FA for $R_1 \cdot R_2$ is given by



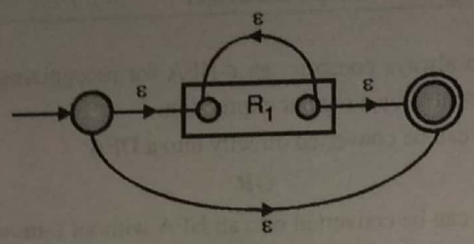
3. FA for R_1^* is given by



Scanned by CamScanner

OR

FA for R_1^* can also be shown as



Finite automata for some primitive strings are shown in Fig. 3.2.1.

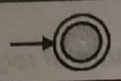
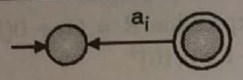
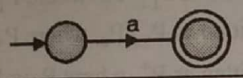
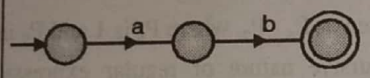
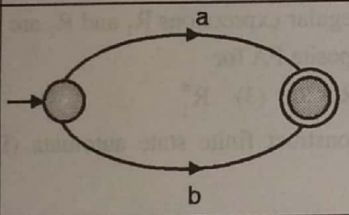
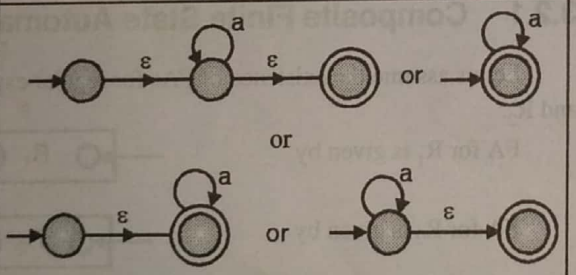
String	Finite Automata
ε	
ϕ	 where $a_i \in \Sigma$
a	 where $a \in \Sigma$
ab	 where $a,b \in \Sigma$
a + b	 where $a,b \in \Sigma$
a*	

Fig. 3.2.1

Example 3.2.1

Write a note on Kleen's theorem.

Solution :

Kleen's theorem says that if a language can be defined by any one of these three ways :

(i) Transition Diagram

(ii) Transition Table

(iii) Regular Expression

Then, it can be defined by other two ways.

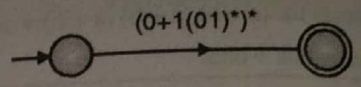
Example 3.2.2

Construct a finite automata for the regular expression

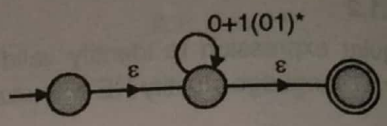
$R = (0 + 1(01)^*)^*$

Solution :

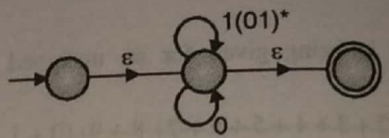
Step 1 :



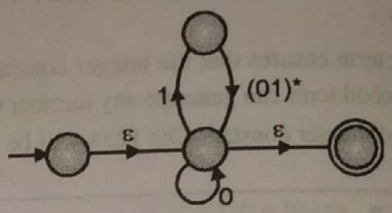
Step 2 :



Step 3 :



Step 4 :



Step 5 :

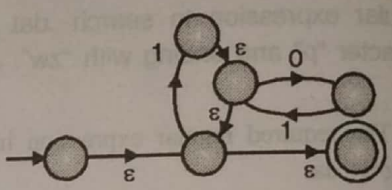
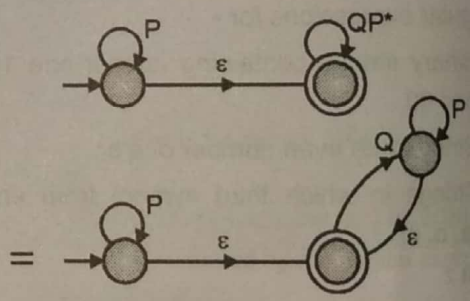


Fig. Ex. 3.2.2

Example 3.2.3

Show that $P^*(QP^*)^* = (P + Q)^*$

Solution : A finite automata recognizing $P^*(QP^*)^*$ is given by :



Naming the states, we get the Fig. Ex. 3.2.3.

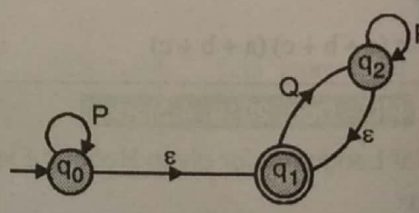


Fig. Ex. 3.2.3 : A finite automata for $P^*(QP^*)^*$

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.

Scanned by CamScanner

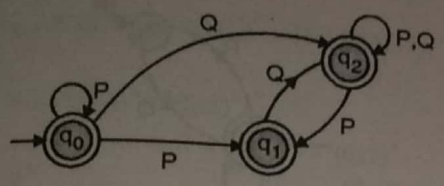
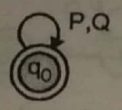


Fig. Ex. 3.2.3(a) : An equivalent finite automata without ε-moves

Since all the three states q_0, q_1 and q_2 are final states (in Fig. Ex. 3.2.3(a)), they can be merged into a single state.



$= (P + Q)^*$

Fig. Ex. 3.2.3(b) : An equivalent DFA

Thus both expressions $P^*(QP^*)^*$ and $(P + Q)^*$ can be recognized by equivalent DFAs and hence we can infer that $P^*(QP^*)^* = (P + Q)^*$

Example 3.2.4

Prove or disprove the following for a regular expression.

Solution :

Step 1 : ε-NFA for $(r^*s^*)^*$

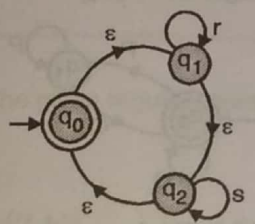


Fig. Ex. 3.2.4

Step 2 : ε-NFA to DFA

ε-closure of q_0 = $\{q_0, q_1, q_2\}$

ε-closure of q_1 = $\{q_0, q_1, q_2\}$

ε-closure of q_2 = $\{q_0, q_1, q_2\}$

The equivalent DFA is given shown in Fig. Ex. 3.2.4(a).

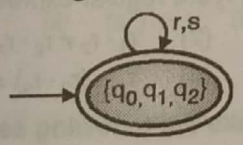


Fig. Ex. 3.2.4(a)

Step 3 : The regular expression for the DFA

$= (r + s)^* =$ R.H.S.

Example 3.2.5

Describe in English the language represented by following expressions :

(i) $(a + ab)^*$ (ii) $(a + b)^* a (a + b)^*$

(iii) $(a^*ab^*ab^*) + b^*$ (iv) $a^+b^*c^+$

Solution :

(i) $(a + ab)^*$:

The language is "Every b is preceded by an a".

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

The language is "The string contains at least one a".

(iii) $(a^*ab^*ab^*) + b^*$:

The language is "The string contains at least two a's or a string of only b's including null string".

(iv) $a^+b^*c^+$:

The language is "One or more a's followed by any number of b's followed by one or more number of c's".

Syllabus Topic : RE to NFA

3.3 Determination of Regular Expression

Formal definition

A regular expression over alphabet

$q = (a_1, a_2, \ldots a_n)$ 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 $L(r_3) = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$

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.

$$\phi + R = R \quad \ldots(3.1)$$
$$\phi \cdot R = R \cdot \phi = \phi \quad \ldots(3.2)$$
$$\varepsilon \cdot R = R \cdot \varepsilon = R \quad \ldots(3.3)$$
$$\varepsilon^* = \varepsilon \quad \ldots(3.4)$$
$$\phi^* = \varepsilon \quad \ldots(3.5)$$
$$R + R = R \quad \ldots(3.6)$$
$$PQ + PR = P(Q + R) \quad \ldots(3.7)$$
$$QP + RP = (Q + R)P \quad \ldots(3.8)$$
$$R^*R^* = R^* \quad \ldots(3.9)$$
$$RR^* = R^*R \quad \ldots(3.10)$$
$$(R^*)^* = R^* \quad \ldots(3.11)$$
$$\varepsilon + RR^* = R^* \quad \ldots(3.12)$$

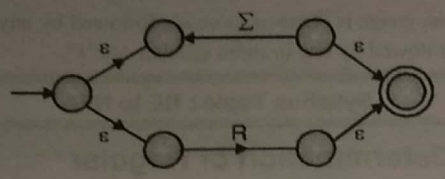
Scanned by CamScanner

$$(PQ)^*P = P(QP)^* \quad ...(3.13)$$
$$(P+Q)^* = (P^*Q^*)^* = (P^*+Q^*)^* \quad ...(3.14)$$
$$(P^*Q)^* = \varepsilon + (P+Q)^*Q \quad ...(3.15)$$

Proof for basic properties :

1. $\phi + R = R$

Machine for $\phi + R$ is given by

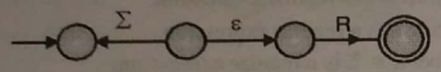


There is only one path from the start state to the final state, recognizing $\varepsilon \cdot R \cdot \varepsilon = R$.

Hence, $\phi + R = R$

2. $\phi \cdot R = \phi$

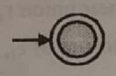
A composite machine for $\phi \cdot R$ is given by

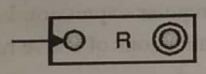


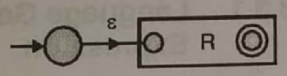
There is no way of reaching the final state from the start state.

$\therefore \quad \phi \cdot R = \phi$

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 machine accepts R and only R.

$\therefore \quad \varepsilon \cdot R = R.$

4. $\varepsilon + RR^* = R^*$

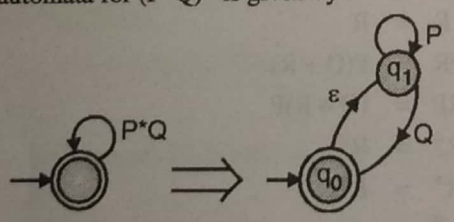
$$\begin{aligned} \text{L.H.S.} &= \varepsilon + RR^* \\ &= \varepsilon + R\{\varepsilon + R + RR + RRR + \ldots\} \\ &= \varepsilon + R + RR + RRR + \ldots \\ &= R^* = \text{R.H.S.} \end{aligned}$$

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.

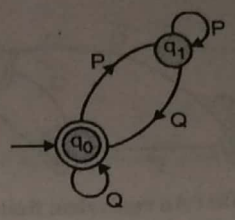
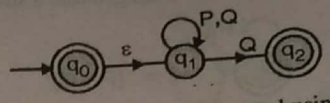
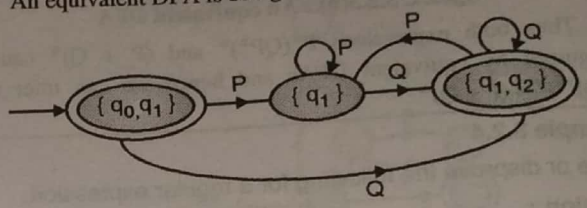


Fig. 3.3.1 : FA for $(P^*Q)^*$

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.

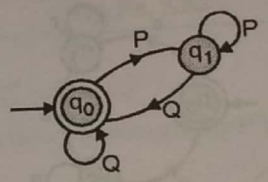


Fig. 3.3.2 : FA for $\varepsilon + (P+Q)^*Q$

DFAs for $(P+Q)^*$ given in Fig. 3.3.1 and $\varepsilon + (P+Q)^*Q$ given in Fig. 3.3.2 are same.

$\therefore \quad (P^*Q)^* = \varepsilon + (P+Q)^*Q.$

Example 3.3.1

Give the examples of sets that demonstrates the following inequality. Here r_1, r_2, r_3 are regular expression :

(1) $r_1 + \varepsilon \neq r_1$ (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

$r_1 = 0, \quad r_2 = 1, \quad r_3 = 1$

Sr. No.	R.E.	L.H.S. of R.E.	R.H.S. of R.E.	Remark
1.	$r_1 + \varepsilon \neq r_1$	$0 + \varepsilon$	0	L.H.S. ≠ R.H.S.
2.	$r_1 \cdot r_1 \neq r_1$	0.0	0	L.H.S. ≠ R.H.S.
3.	$r_1 \cdot r_2 \neq r_2 \cdot r_1$	0.1	1.0	L.H.S. ≠ R.H.S.
4.	$r_1 (r_2 \cdot r_3) \neq (r_1 + r_2)(r_1 \cdot r_3)$	$0 + 10$	$(0+1) \cdot 00 = 000 + 100$	L.H.S. ≠ R.H.S.

Scanned by CamScanner

Example 3.3.2

Prove that : $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$

Solution :

$$\begin{aligned} \text{L.H.S.} &= \varepsilon + 1^*(011)^*(1^*(011)^*)^* \\ &= \varepsilon + RR^*, \text{ where R is taken as } 1^*(011)^* \\ &= R^* && \text{... [By Equation (3.12)]} \\ &= (1^*(011)^*)^* \\ &= (1 + 011)^* && \text{...[By Equation (3.14)]} \\ &= \text{R.H.S.} \end{aligned}$$

Example 3.3.3

Prove that : $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$

Solution :

$$\begin{aligned} \text{L.H.S.} &= (1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) \\ &= (1 + 00^*1)[\varepsilon + (0 + 10^*1)^*(0 + 10^*1)] \\ & && \text{... [By Equation (3.7)]} \\ &= (1 + 00^*1)(0 + 10^*1)^* && \text{... [By Equation (3.12)]} \\ &= [(\varepsilon + 00^*)1](0 + 10^*1)^* && \text{...[By Equation (3.8)]} \\ &= 0^*1(0 + 10^*1)^* && \text{... [By Equation (3.12)]} \\ &= \text{R.H.S.} \end{aligned}$$

Example 3.3.4

Prove or disprove the given regular expression $(rs + r)^* = r(sr + r)^*$

Solution :

ε and rs are in $(rs + r)^*$

but ε and rs $\notin$ $r(sr + r)^*$

Hence, $(rs + r)^* \neq r(sr + r)^*$

Example 3.3.5

If s = {aa, b} write all the strings in s^* which are having length 4 or less, also say the following is true or false.

(i) $(s^+)^* = (s^*)^*$ (ii) $(s^+)^+ = s^+$ (iii) $(s^*)^+ = (s^+)^*$

Solution :

We have to find all strings of length 4 or less in $(aa + b)^*$

(i) String of length 0 = {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

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 3.3.7

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.

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. = $(1 + 0)^*(00 + 11 + 10)$

Example 3.3.8 MU - May 2017, 5 Marks

Write regular expression to denote a language L which accepts all the strings which begins or end with either 00 or 11.

Solution :

String beginning with 00 is $00(0 + 1)^*$

String beginning with 11 is $11(0 + 1)^*$

∴ Strings beginning with either 00 or 11 is

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

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

String ending with 00 is $(0 + 1)^*00$

String ending with 11 is $(0 + 1)^*11$

∴ Strings ending with 00 or 11 is

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

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

∴ Required regular expression is

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

Scanned by CamScanner

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 in Fig. Ex. 3.3.9

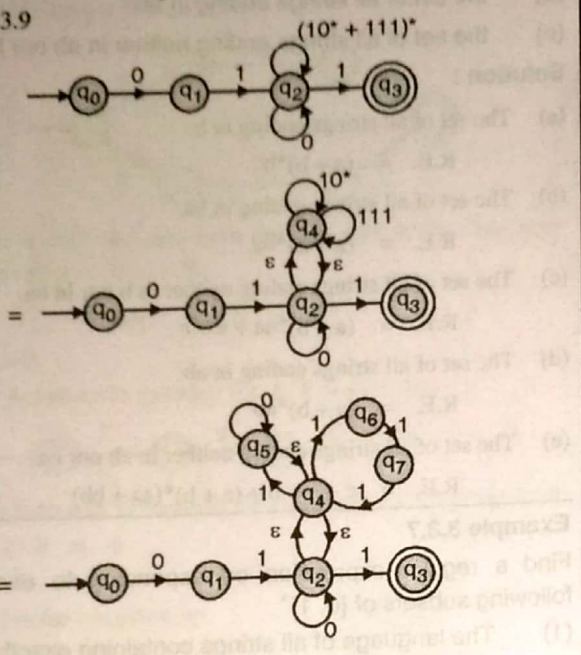


Fig. Ex. 3.3.9 : ε-NFA for Example 3.3.9

DFA is being constructed from ε-NFA. It is shown in Fig. Ex. 3.3.9(a).

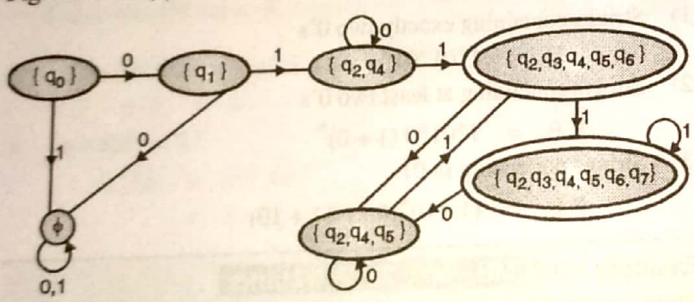


Fig. Ex. 3.3.9(a) : DFA for Example 3.3.9

Example 3.3.10

Construct an NFA to accept the language represented by a*b*c*. Convert the NFA to its equivalent DFA.

Solution :

Step 1 : RE to NFA

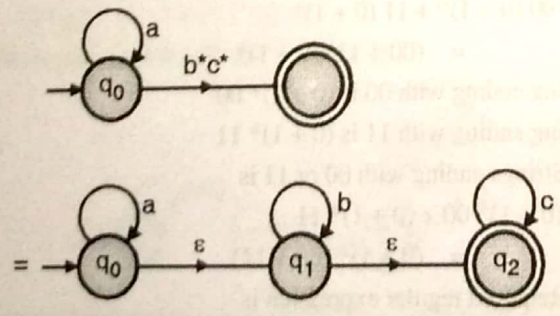


Fig. Ex. 3.3.10(a)

Step 2 : NFA to DFA

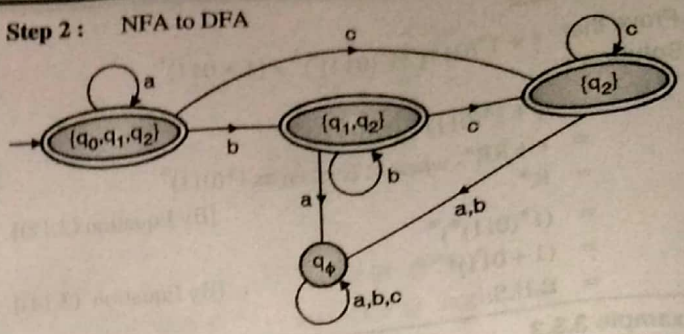


Fig. Ex. 3.3.10(b)

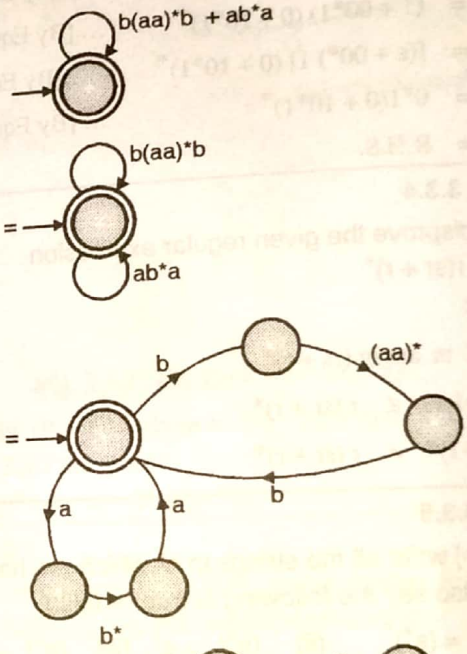
Example 3.3.11

Construct an NFA with ε-moves for the regular expression (b(aa)*b + ab*a)*

Convert this NFA to its equivalent DFA.

Solution :

Step 1 : RE to NFA



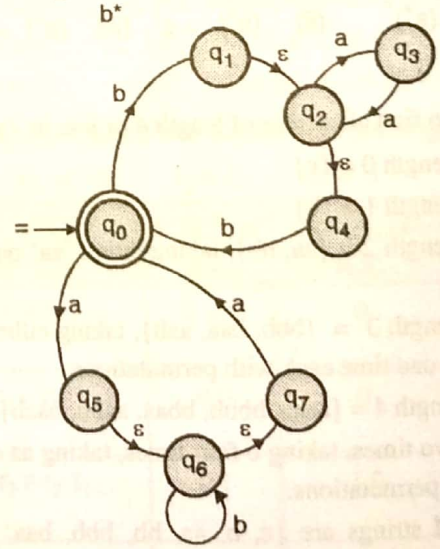


Fig. Ex. 3.3.11(a)

Scanned by CamScanner

Step 2 : **NFA to DFA :** ε-closure of states :

State	ε-closure
q_0	$\{q_0\}$
q_1	$\{q_1, q_2, q_4\}$
q_2	$\{q_2, q_4\}$
q_3	$\{q_3\}$
q_4	$\{q_4\}$
q_5	$\{q_5, q_6, q_7\}$
q_6	$\{q_6, q_7\}$
q_7	$\{q_7\}$

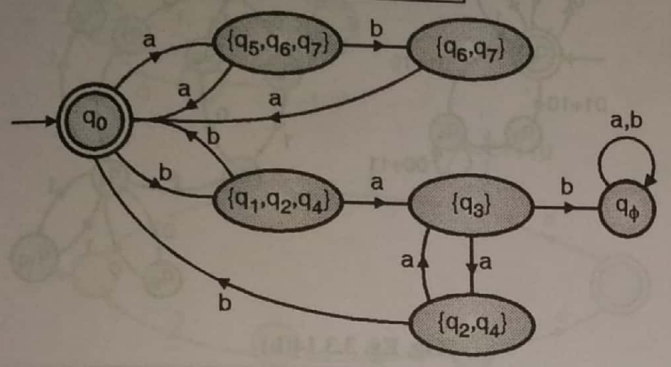


Fig. Ex. 3.3.11(b)

Example 3.3.12

Which of the following are true ? Explain.

(i) $baa \in a^*b^*a^*b^*$

(iii) $a^*b^* \cap b^*c^* = \phi$

(ii) $b^*a^* \cap a^*b^* = a^* \cup b^*$

(iv) $abcd \in (a\,(cd)^*b)^*$

Solution :

(i) $baa \in a^*b^*a^*b^*$

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

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) $b^*a^* \cap a^*b^* = a^* \cup b^*$

$$b^*a^* = (\varepsilon + bb^*)(\varepsilon + aa^*) \quad \text{...[From Equation (3.12)]}$$
$$= \varepsilon + aa^* + bb^* + bb^*aa^*$$
$$= (\varepsilon + aa^*) + (\varepsilon + bb^*) + bb^*aa^*$$
$$= a^* + b^* + bb^*aa^* \quad \text{...(1)}$$

Similarly,

$$a^*b^* = a^* + b^* + aa^*bb^* \quad \text{...(2)}$$

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 3.3.13

For each of the following draw DFA of following regular expressions :

a) $(11 + 00)^*$ b) $(111 + 100)^*0$

Solution :

a) DFA for $(11 + 00)^*$

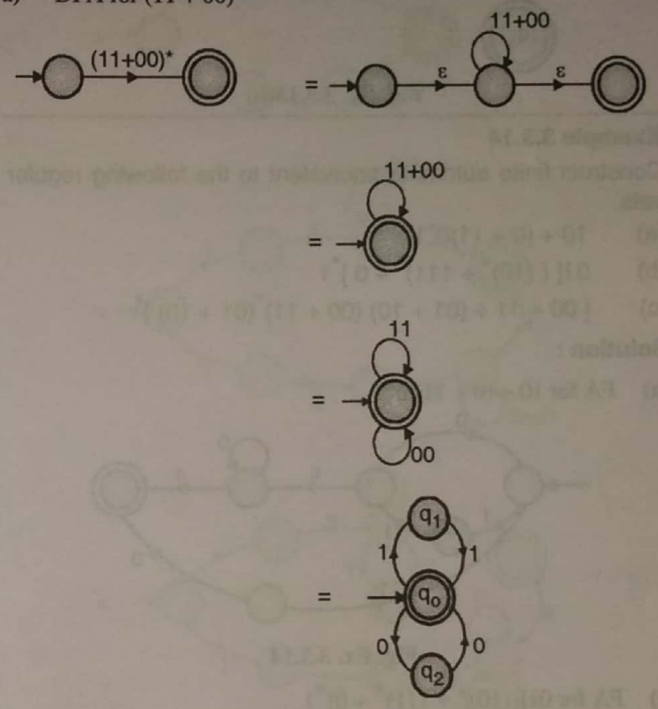
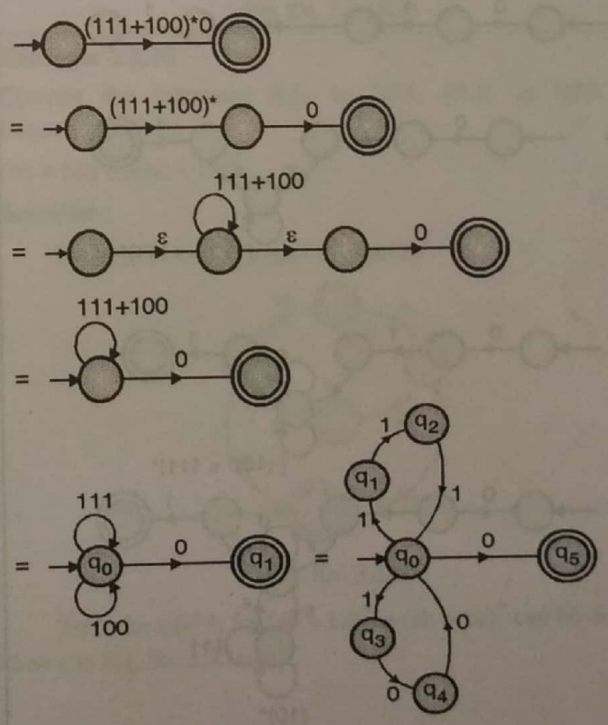


Fig. Ex. 3.3.13(a)

b) DFA for $(111 + 100)^*0$



Ex. 3.3.13(b) Contd...

Scanned by CamScanner

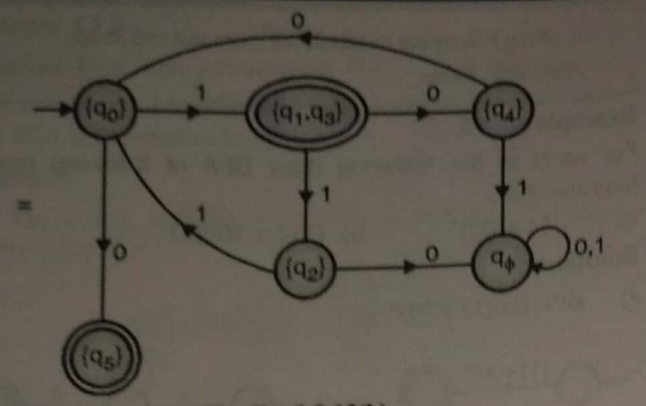


Fig. Ex. 3.3.13(b)

Example 3.3.14

Construct finite automata equivalent to the following regular sets.

(a) $10 + (0 + 11)0^*1$

(b) $01[((10)^* + 111)^* + 0]^*1$

(c) $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

Solution :

(a) FA for $10 + (0 + 11)0^*1$

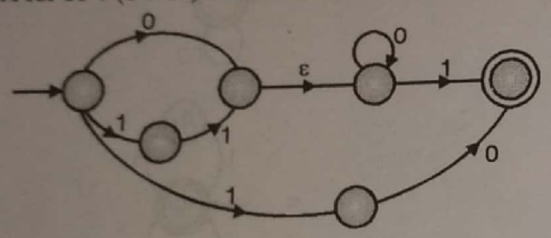


Fig. Ex. 3.3.14

(b) FA for $01[((10)^* + 111)^* + 0]^*1$

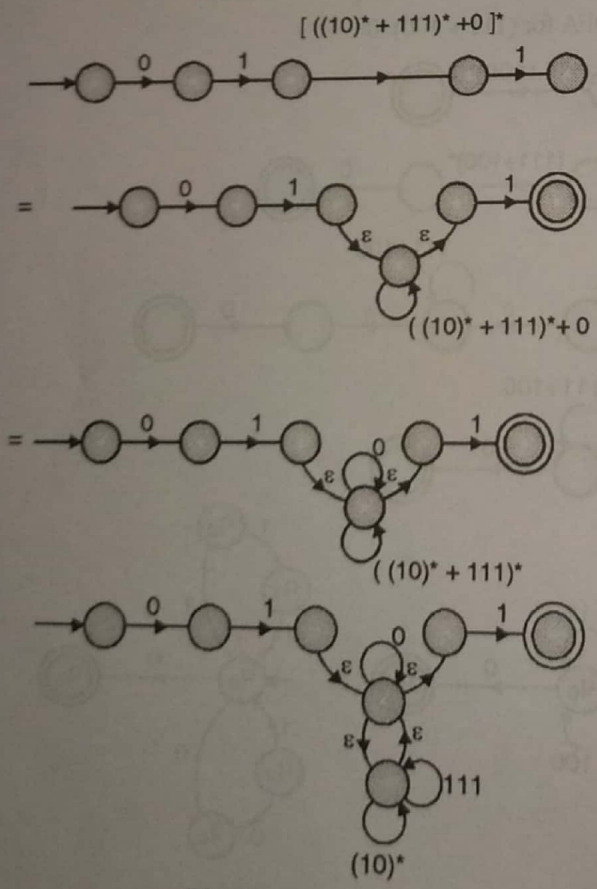


Fig. Ex. 3.3.14(a)

(c) FA for $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

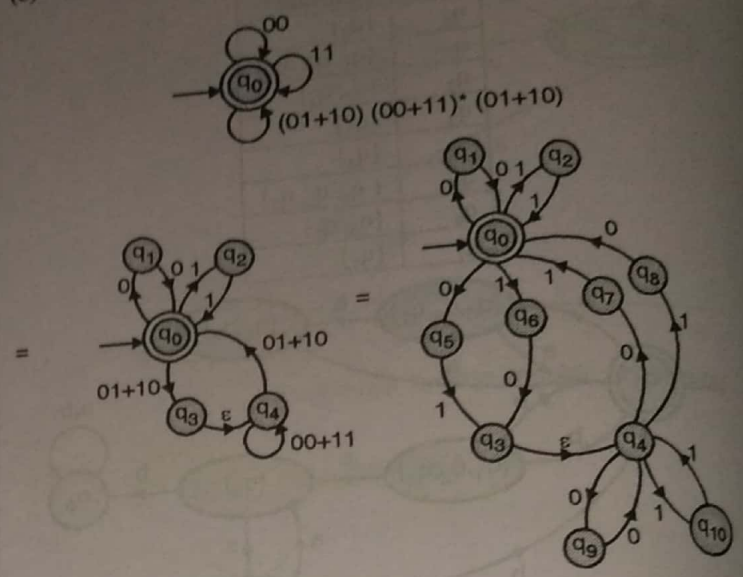


Fig. Ex. 3.3.14(b)

Example 3.3.15

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 $(1 + 10 + 110)^*0$

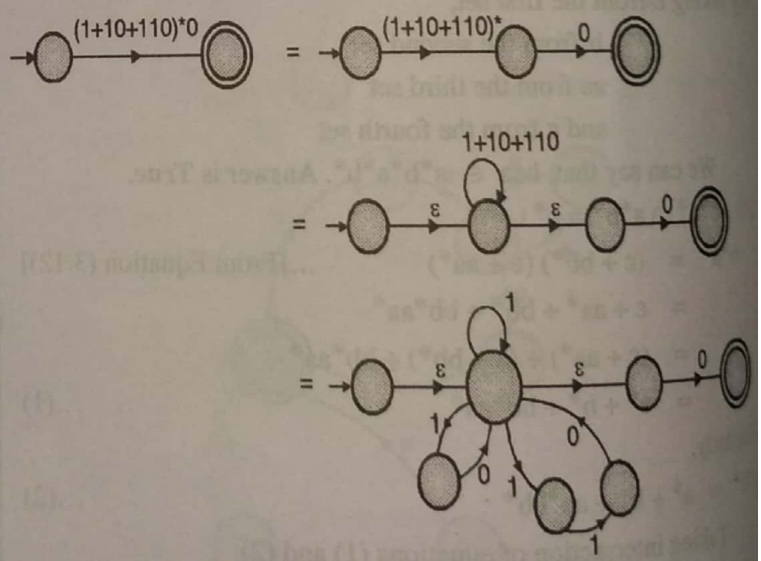


Fig. Ex. 3.3.15(a)

Scanned by CamScanner

2. F.A. for 1(01 + 10)* + 0(11 + 10)*

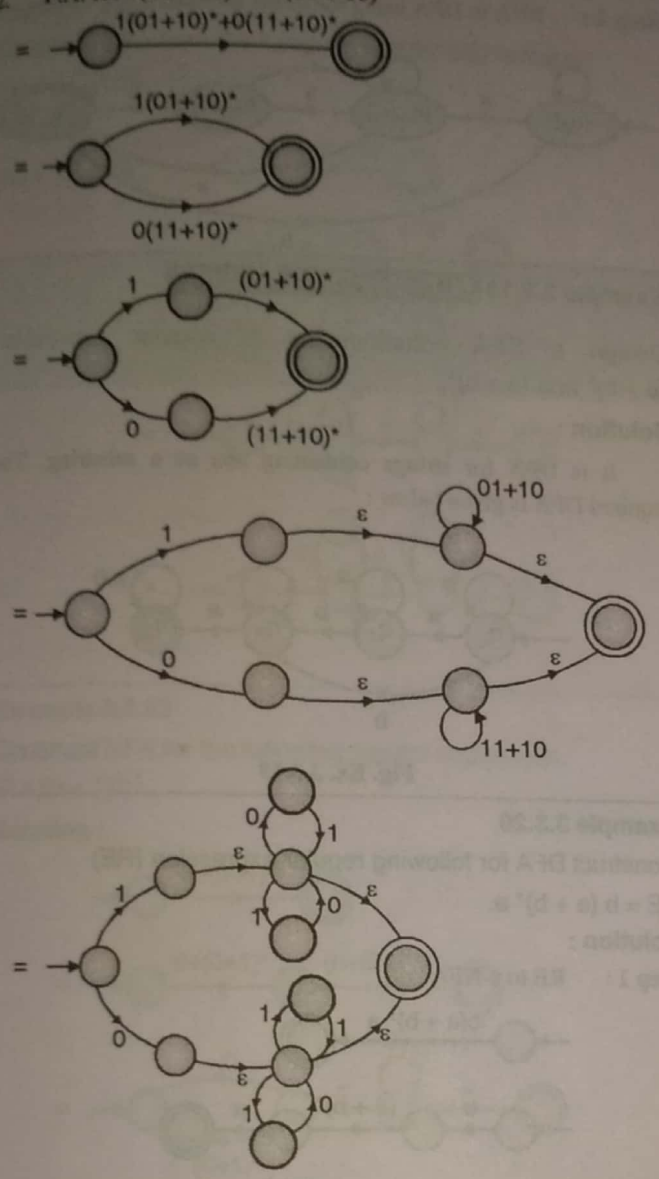


Fig. Ex. 3.3.15(b)

3. F.A. for (010 + 00)*(10)

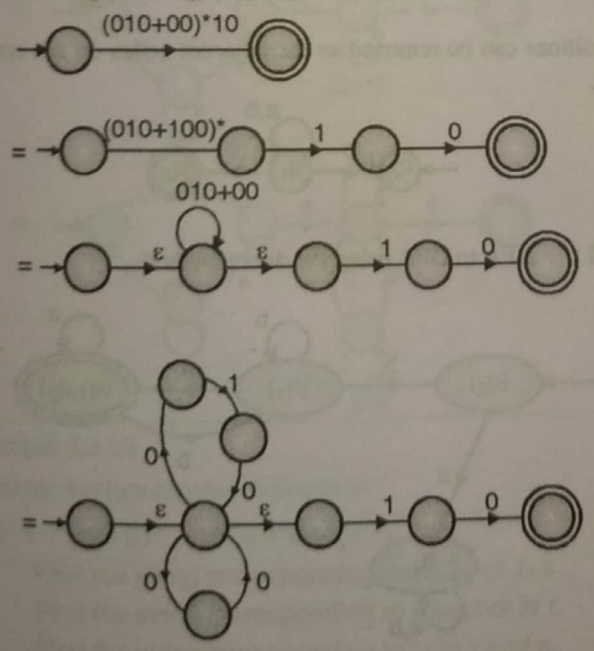


Fig. Ex. 3.3.15(c)

4. F.A. for 1(1 + 10)* + 10(0 + 01)*

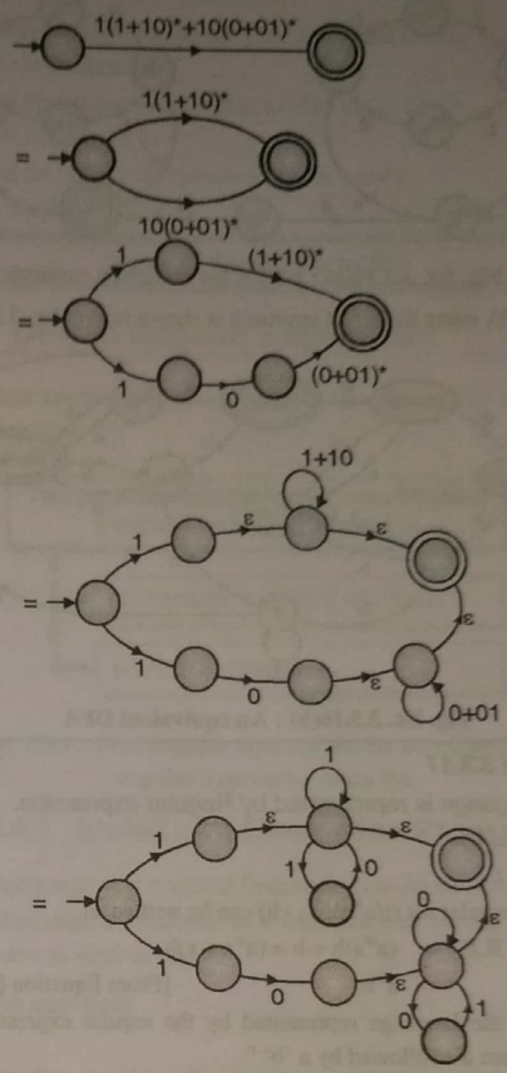


Fig. Ex. 3.3.15(d)

Example 3.3.16

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

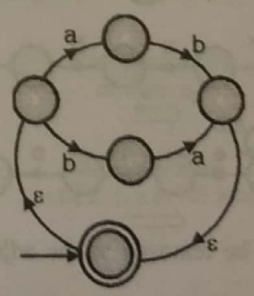


Fig. Ex. 3.3.16

Thus the ε-NFA for $(ab + ba)^* aa (ab + ba)^*$ can be drawn as shown in Fig. Ex. 3.3.16(a)

Scanned by CamScanner

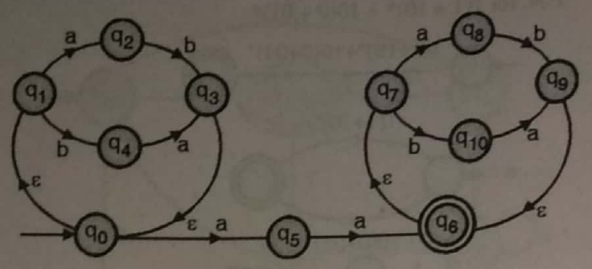


Fig. Ex. 3.3.16(a) : ε-NFA for the given example

DFA using the direct approach is shown in Fig. Ex. 3.3.16(b).

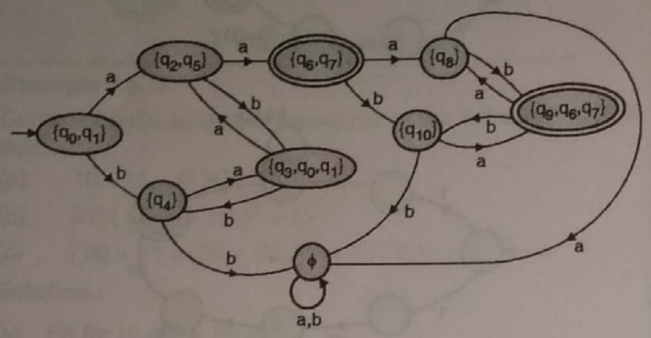


Fig. Ex. 3.3.16(b) : An equivalent DFA

Example 3.3.17

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

$$\text{R.E.} = (a^*a)b + b = (a^*a + \varepsilon)b$$
$$= a^*b \qquad \text{[From Equation (3.12)]}$$

Thus the language represented by the regular expression is "zero or more a's followed by a 'b' ".

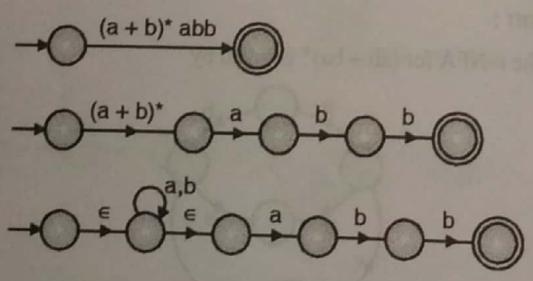
Example 3.3.18

Construct DFA for following regular expression (RE)

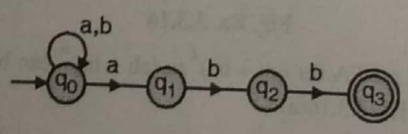
RE = (a + b) * a b b.

Solution :

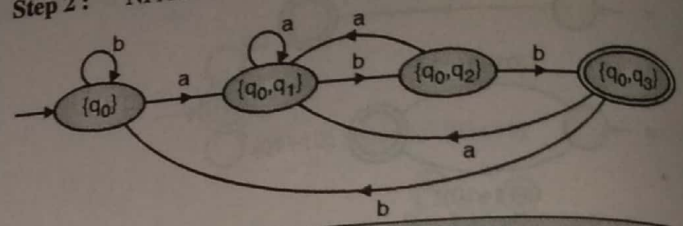
Step 1 : RE to ε-NFA



ε-transitions can be removed as the adjacent nodes do not contain a loop.



Step 2 : NFA to DFA using the direct method.



Example 3.3.19 MU - May 2017, 10 Marks

Design a DFA corresponding to regular expression $(a + b)^* aba (a + b)^*$.

Solution :

It is DFA for strings containing aba as a substring. The required DFA is given below :

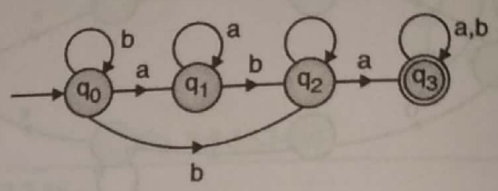


Fig. Ex. 3.3.19

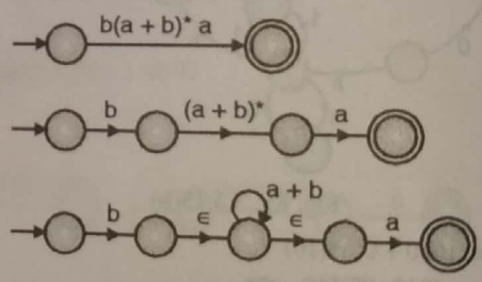
Example 3.3.20

Construct DFA for following regular expression (RE)

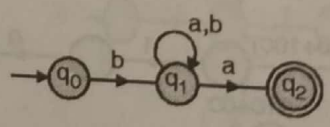
RE = b (a + b)* a.

Solution :

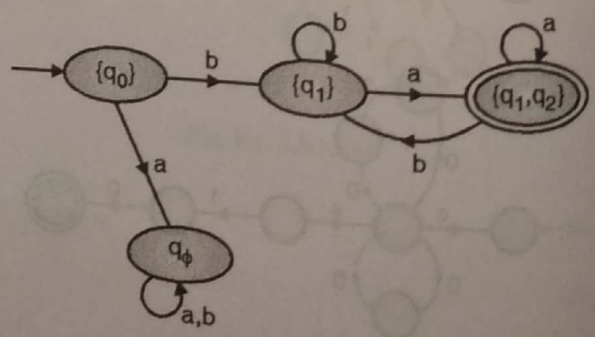
Step 1 : RE to ε-NFA



ε-transitions can be removed as the adjacent nodes do not contain loops.



Step 2 : NFA to DFA using the direct method.



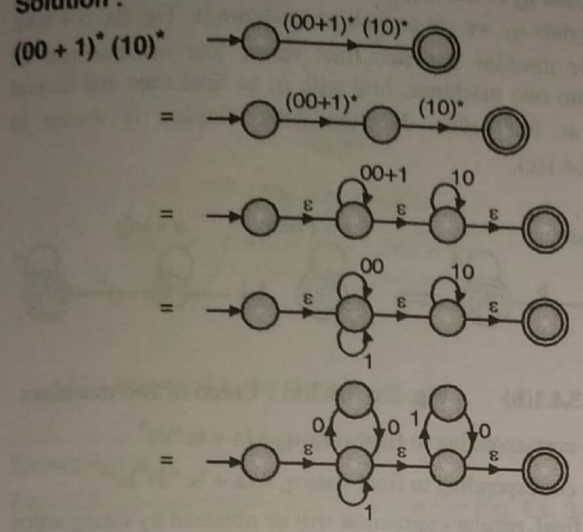
Scanned by CamScanner

Example 3.3.21

Construct NFA from the following regular expressions :

$(00 + 1)^* (10)^*$

Solution :

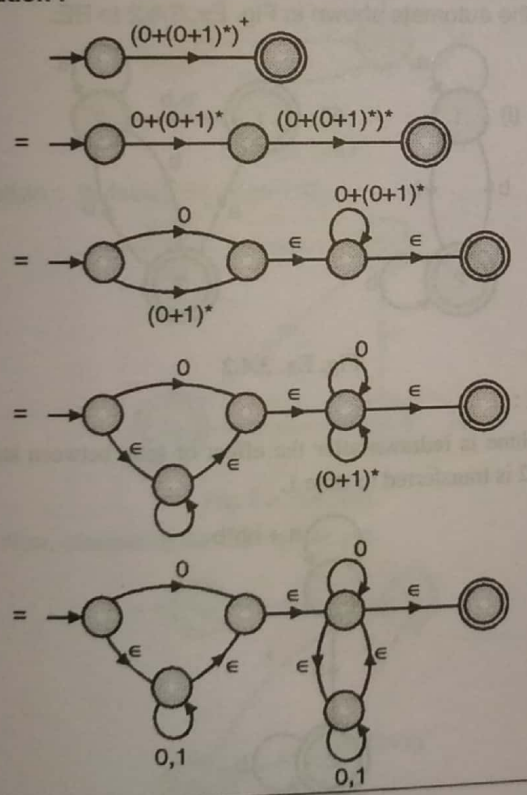


Example 3.3.22

Construct NFA for the following regular expression.

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

Solution :



Example 3.3.23

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.

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.

Syllabus Topic : FA to RE

3.4 DFA to Regular Expression

There are two popular approaches for constructing a regular expression from FA.

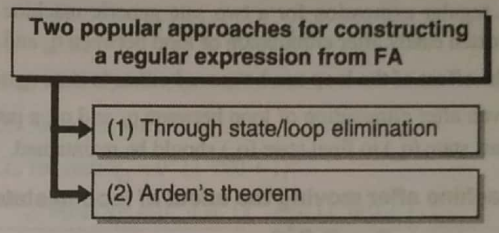


Fig. C3.1 : Two popular approaches for constructing a regular expression from FA

➜ 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**

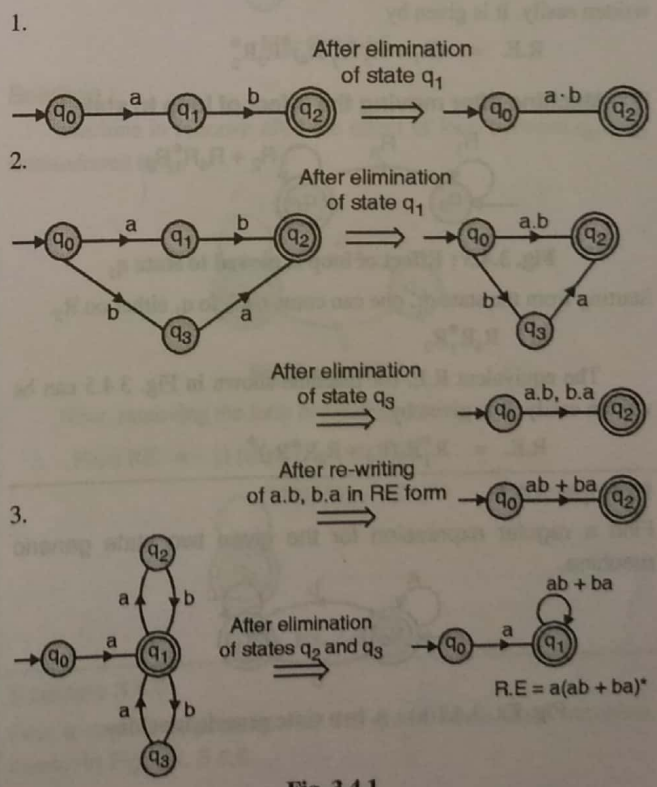


Fig. 3.4.1

Scanned by CamScanner

3.4.1(a) A Generic One State Machine

The regular expression for strings accepted by a one state generic machine is R^*.

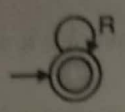


Fig. 3.4.2 : A generic one-state machine

3.4.1(b) A Generic Two State Machine

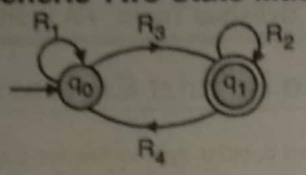


Fig. 3.4.3 : A generic two-state machine

- 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

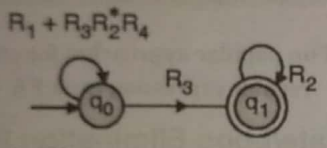


Fig. 3.4.4 : Effect of loop is moved to state q_0

Starting from the start state q_0, one can come back to q_0 either on R_1,

or $R_3R_2^*R_4$

The equivalent R.E. for machine shown in Fig. 3.4.4 can be written easily. It is given by

$$\text{R.E.} = (R_1 + R_3 R_2^* R_4)^* R_3 R_2^*$$

Machine after moving the effect of loop to state q_1

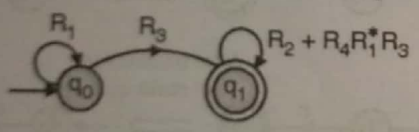


Fig. 3.4.5 : Effect of loop is moved to state q_1

Starting from the state q_1, one can come back to q_1 either on R_2.

or $R_4R_1^*R_3$

The equivalent R.E. for machine shown in Fig. 3.4.5 can be written easily. It is given by :

$$\text{R.E.} = R_1^* R_3 (R_2 + R_4 R_1^* R_3)^*$$

Example 3.4.1

Find a regular expression for the given two state generic machine.

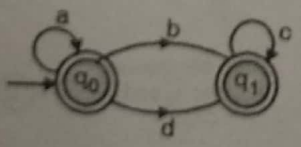


Fig. Ex. 3.4.1(a) : A two state generic machine

Solution :

Neither state q_0 nor state q_1 can be eliminated as both are final states. Effect of the loop between q_0 and q_1 can be moved either to state q_0 or state q_1. Moving the effect of loop between q_0 and q_1 on state q_0, we get a machine as shown in Fig. Ex. 3.4.1(b). The above machine has two final states. The machine can be divided into two machines, first with q_0 as final state and second with q_1 as final state. Machine after division is shown in Fig. Ex. 3.4.1(c).

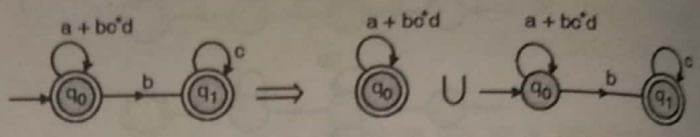


Fig. Ex. 3.4.1(b) **Fig. Ex. 3.4.1(c) : Union of two machines**

R.E. corresponding to final state $q_0 = (a + bc^*d)^*$

R.E. corresponding to final state $q_1 = (a + bc^*d)^* bc^*$

The final, regular expression will be obtained by taking union of R.E. for state q_0 and q_1.

$$\therefore \text{Final R.E.} = (a + bc^*d)^* + (a + bc^*d)^* bc^*$$
$$= (a + bc^*d)^* [\varepsilon + bc^*]$$

Example 3.4.2

Convert the automata shown in Fig. Ex. 3.4.2 to RE.

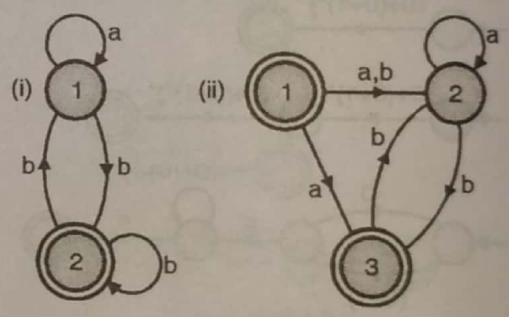


Fig. Ex. 3.4.2

Solution :

(i) Machine is redrawn after the effect of loop between state 1 and 2 is transferred to state 1.

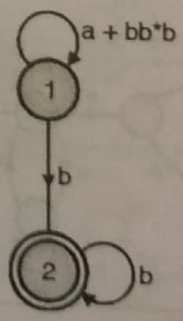


Fig. Ex. 3.4.2(a)

$$\therefore \text{R.E.} = (a + bb^*b)^* b^* b^*$$

(ii) Machine is redrawn after the effect of loop between state 2 and 3 is transferred to state 2.

Scanned by CamScanner

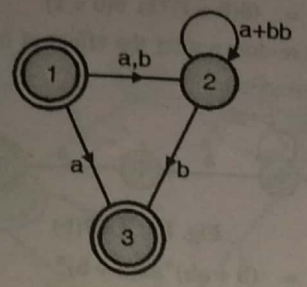


Fig. Ex. 3.4.2(b)

$$\text{R.E.} = \underbrace{\varepsilon}_{\text{for state 1}} + \underbrace{(a+b)(a+bb)^*b + a}_{\text{for state 3}}$$

Example 3.4.3

Find R.E. corresponding to T.G. shown in Fig. Ex. 3.4.3.

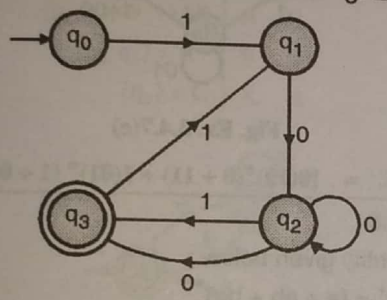


Fig. Ex. 3.4.3

Solution : Reducing the given T.G.

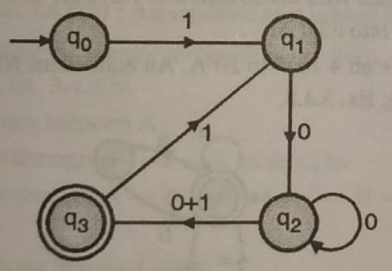


Fig. Ex. 3.4.3(a)

Now, eliminating the state q_2, we get,

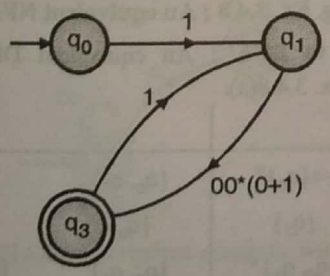


Fig. Ex. 3.4.3(b)

Machine is redrawn after the effect of loop between q_1 and q_3 is transferred to q_1.

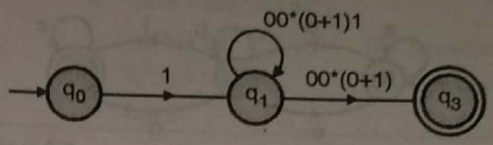


Fig. Ex. 3.4.3(c)

$\therefore$ Final R.E. $= 1[00^*(0+1)1]^*\ 00^*\ (0+1)$

Example 3.4.4

Find R.E. for shown in Fig. Ex. 3.4.4 FA

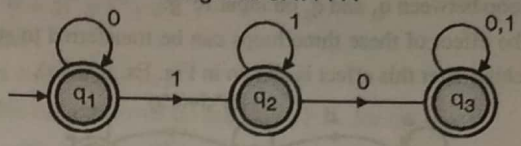


Fig. Ex. 3.4.4

Solution : R.E. for state $q_1 = 0^*$

R.E. for state $q_2 = 0^*11^*$

R.E. for state $q_3 = 0^*11^*0(0+1)^*$

$\therefore$ Final RE $= 0^* + 0^*11^* + 0^*11^*0(0+1)^*$

Example 3.4.5

Consider the transition diagram shown in Fig. Ex. 3.4.5. Convert it to equivalent RE.

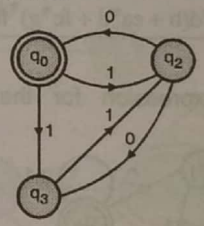


Fig. Ex. 3.4.5

Solution :

Machine in redrawn after the effect of loop between q_0, q_3, q_2 is transferred to q_0.

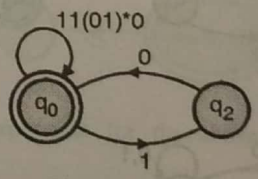


Fig. Ex. 3.4.5(a)

Now, removing the loop between q_0 and q_2,

$\therefore$ Final RE $= [11(01)^*0 + 1(01)^*0]^*$

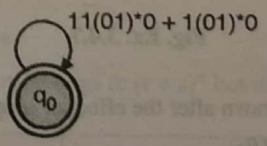


Fig. Ex. 3.4.5(b)

Example 3.4.6

Find a regular expression for the given three state machine shown in Fig. Ex. 3.4.6.

Scanned by CamScanner

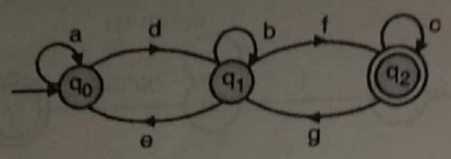


Fig. Ex. 3.4.6

Solution : The state q_1 is at the crossing point of three loops :

1. q_1 to q_1 on input b.
2. Loop between q_1 and q_0 on input ea*d
3. Loop between q_1 and q_2 on input fc*g.

The effect of these three loops can be transferred to state q_1; the machine after this effect is shown in Fig. Ex. 3.4.6(a).

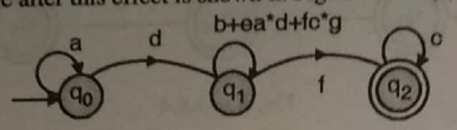


Fig. Ex. 3.4.6(a) : Moving the effect of loops on state q_1

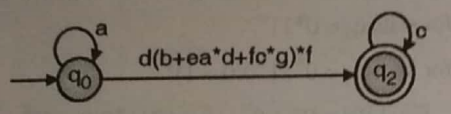


Fig. Ex. 3.4.6(b) : After elimination of state q_1

The state q_1 can be eliminated.

Now, the regular expression can be written as,

$$\text{R.E.} = a^*d(b + ea^*d + fc^*g)^*fc^*$$

Example 3.4.7

Find the regular expression for the DFA's, shown in Fig. Ex. 3.4.7.

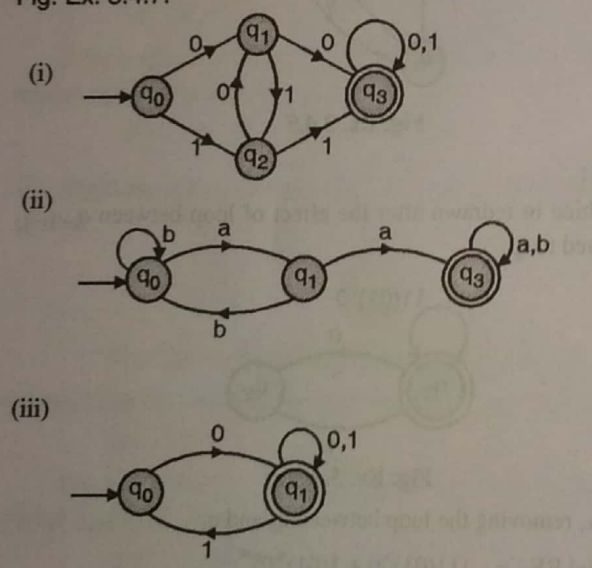


Fig. Ex. 3.4.7

Solution :

(i) Machine is redrawn after the effect of loop between q_0 and q_1 is transferred to q_0.

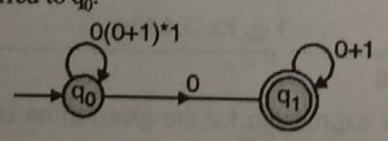


Fig. Ex. 3.4.7(a)

$$\text{R.E.} = (0(0 + 1)^*1)^*0(0 + 1)^*$$

(ii) Machine is re-drawn after the effect of loop between q_0 and q_1 is transferred to q_0

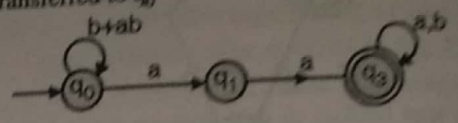


Fig. Ex. 3.4.7(b)

$$\text{R.E.} = (b + ab)^*aa(a + b)^*$$

(iii) Machine is redrawn after the effect of loop between q_1 and q_2 is transferred to both q_1 and q_2. There are two paths from q_0 and q_3 [0 + 11]. There are two paths from q_2 to q_3 [1 + 00].

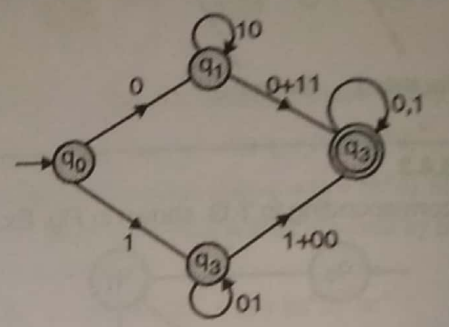


Fig. Ex. 3.4.7(c)

$$\text{R.E.} = [0(10)^*(0 + 11) + 1(01)^*(1 + 00)](0 + 1)^*$$

Example 3.4.8

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. 3.4.8.

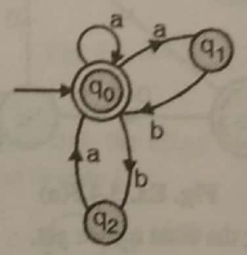


Fig. Ex. 3.4.8 : An equivalent NFA

Step 2 : **NFA to DFA :** An equivalent DFA is given in Fig. Ex. 3.4.8(a).

	a	b
$\rightarrow\{q_0\}^*$	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_2\}$	$\{q_0\}$	ϕ
$\{q_0, q_1\}^*$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}^*$	$\{q_0, q_1\}$	$\{q_2\}$

Scanned by CamScanner

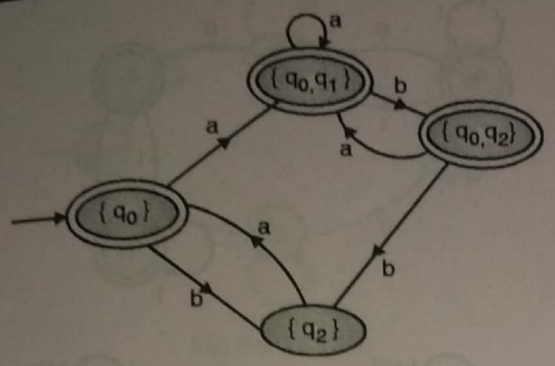


Fig. Ex. 3.4.8(a) : An equivalent DFA

Step 3 : Minimization of DFA obtained from Fig. Ex. 3.4.8(a). Behaviour of $\{q_0\}$ and $\{q_0, q_2\}$ are identical as their a-successors and b-successors are same. Merging the two states $\{q_0\}$ and $\{q_0, q_2\}$, we get, the minimum state DFA as shown in Fig. Ex. 3.4.8(b).

	a	b
$\{q_0, q_2\} = \rightarrow A^*$	B	C
$\{q_0, q_1\} = B^*$	B	A
$\{q_2\} = C$	A	ϕ

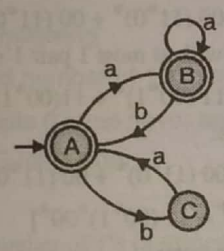


Fig. Ex. 3.4.8(b) : An equivalent minimal DFA

Step 4 : Writing of regular expression for DFA derived from Fig. Ex. 3.4.8(b).

- There are two loops on A.
  1. Loop through $A \rightarrow C \rightarrow A$ on input ba
  2. Loop through $A \rightarrow B \rightarrow B \rightarrow \ldots \rightarrow B \rightarrow A$ on input aa*b
  3. There are two final states.

An equivalent machine without a loop is shown in Fig. Ex. 3.4.8(c).

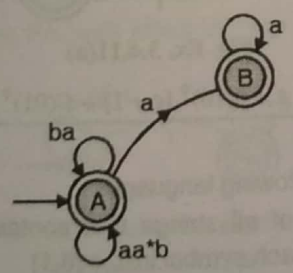


Fig. Ex. 3.4.8(c) : An equivalent machine without a loop

Regular expression for the machine shown in Fig. Ex. 3.4.8(c) is given by :

$$\text{R.E.} = \underbrace{(aa^*b + ba)^*}_{\text{Corresponding to state A}} + \underbrace{(aa^*b + ba)^* aa^*}_{\text{Corresponding to state B}}$$

$$= (aa^*b + ba)^*[\varepsilon + aa^*]$$

$$= (aa^*b + ba)^* a^* \quad \ldots \text{[By Equation (3.12)]}$$

$$= (a^*ab + ba)^* a^* \quad \ldots \text{[By Equation (3.10)]}$$

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

Example 3.4.9

Prove or disprove the following for regular expression r, s and t.

(a) $(rs + r)^*r = r(sr + r)^*$

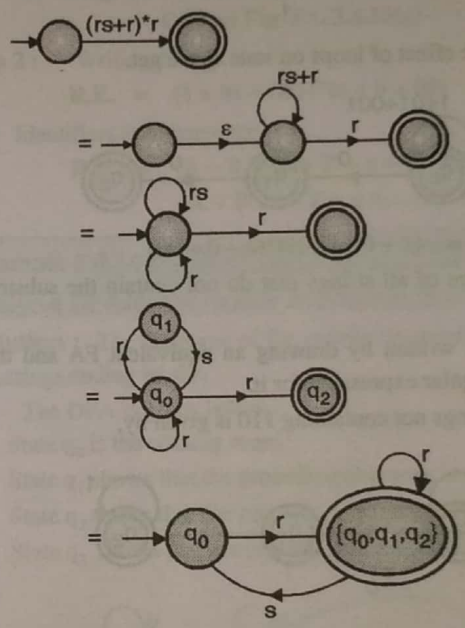
(b) $s(rs + s)^*r = rr^*s(rr^*s)^*r$

(c) $(r + s)^* = r^* + s^*$

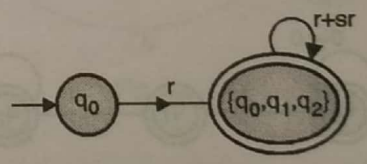
Solution :

(a) $(rs + r)^*r = r(sr + r)^*$

This can be proved by drawing a F.A. for $(rs + r)^*r$



Moving the effect of the loop on $\{q_0, q_1, q_2\}$, we get,



The regular expression for the above FA is given by $r(sr + r)^* = \text{R.H.S.}$

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 3.4.10

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.

Scanned by CamScanner

b) The language of all strings that do not contain the substring 110.

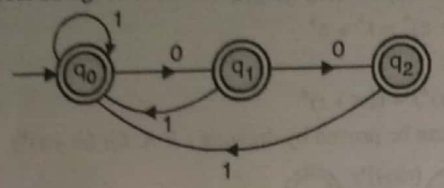
c) The language of all strings containing both 101 and 010 as substring.

Solution :

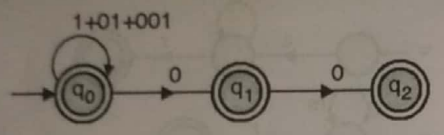
a) The language of all strings not containing the substring 000.

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,



Moving the effect of loops on state q_0 we get,

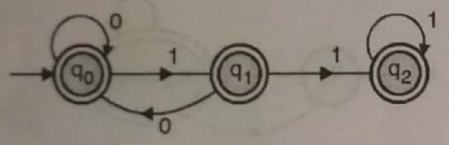


$\therefore$ R.E. $= (1 + 01 + 001)^*(\varepsilon + 0 + 00)$

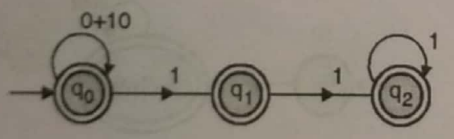
b) The language of all strings that do not contain the substring 110.

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,



Moving the effect of loop on state q_0, we get,



$\therefore$ 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. $= (0 + 1)^*101(0 + 1)^*010(0 + 1)^*$
$+ (0 + 1)^*010(0 + 1)^*101(0 + 1)^*$

Example 3.4.11

Write RE for the following $\Sigma = \{0,1\}$

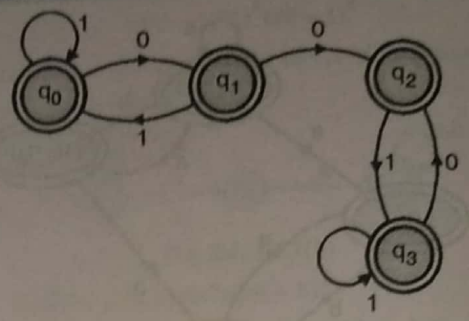
(i) At most 1 pair of consecutive '0' or 1 pair of consecutive '1'.

(ii) Without any double letter.

Solution :

(i) At most 1 pair of consecutive '0' or 1 pair of consecutive '1'.

A finite automata with at most 1 pair of consecutive 0's is shown in Fig. Ex. 3.4.11.



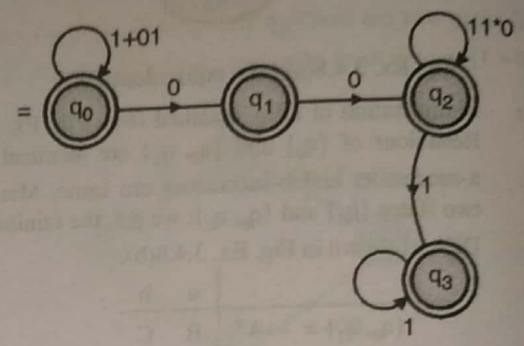


Fig. Ex. 3.4.11

$\therefore$ R.E. for strings with at most 1 pair of consecutive 0's is

$(1 + 01)^* [\varepsilon + 0 + 00 (11^*0)^* + 00 (11^*0)^* \ 11^*]$

$\therefore$ R.E. for strings with at most 1 pair 1's is

$(0 + 10)^* [\varepsilon + 1 + 11 (00^*1)^* + 11(00^*1)^*00^*]$

$\therefore$ Required R.E. is:

$(1 + 01)^* [\varepsilon + 0 + 00 (11^*0)^* + 00 (11^*0)^* \ 11^*] + (0 + 10)^*$
$[\varepsilon + 1 + 11 (00^*1)^* + 11 (00^*1)^*00^*]$

(ii) Without any double letter

A Finite Automata without any double letter is shown in Fig. Ex. 3.4.11(a).

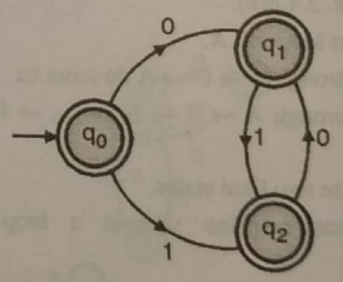


Fig. Ex. 3.4.11(a)

$\therefore$ R.E. $= \varepsilon + 0(10)^* [\varepsilon + 1] + 1 (01)^* [\varepsilon + 0]$

Example 3.4.12

Write RE for the following languages :

(i) The language of all strings that contains at least one occurrence of each symbol in $\Sigma = \{0,1\}$

(ii) $\Sigma = \{a, b\}$ such that ab is not a substring of any strings.

Solution :

(i) The FA for language of all strings containing at least one occurrence of each symbol in $\Sigma = \{0, 1\}$ is shown in Fig. Ex. 3.4.12(a) :

Scanned by CamScanner

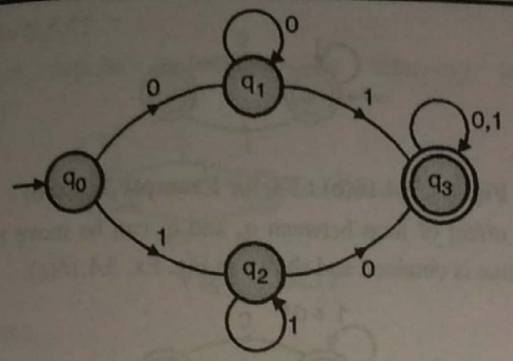


Fig. Ex. 3.4.12(a)

The RE can be written from the DFA.

$$\text{R.E.} = 00^*1(0+1)^* + 11^*0\,(0+1)^*$$

(ii) The DFA for language of all strings not containing 'ab' as a substring is shown in Fig. Ex. 3.4.12(b) :

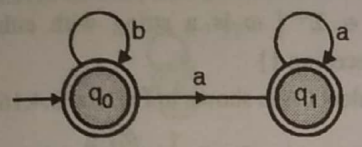


Fig. Ex. 3.4.12(b)

$$\therefore \text{R.E.} = b^* + b^*aa^*$$

Example 3.4.13

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.

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

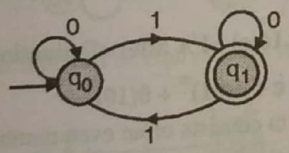


Fig. Ex. 3.4.13(a) : DFA for strings having odd number of 1's

An equivalent FA with effect of loop between q_0 and q_1 moved to q_0.

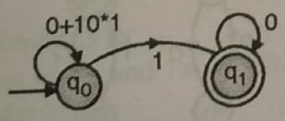


Fig. Ex. 3.4.13(b) : An equivalent machine of DFA given in Fig. Ex. 3.4.13(a)

Step 2 : Writing of R.E. from Fig. Ex. 3.4.13(b).

$$\text{R.E.} = (0 + 10^*1)^*10^*$$

(ii) $\Sigma = \{0, 1\}$ Triple 0 must never appear in string.

Step 1 : Drawing an equivalent DFA.

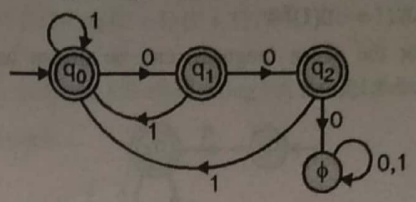


Fig. Ex. 3.4.13(c) : DFA for strings not having 000

An equivalent FA with effect of loops moved to q_0 is shown in Fig. Ex. 3.4.13(d).

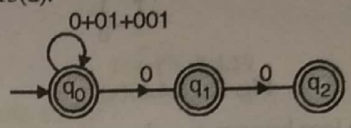


Fig. Ex. 3.4.13(d) : An equivalent FA for DFA of Fig. Ex. 3.4.13(c)

Step 2 : Writing of R.E.

$$\text{R.E.} = (1 + 01 + 001)^*(\varepsilon + 0 + 00)$$

(iii) Identifiers in C language.

$$\text{R.E.} = (A + B + \ldots + Z + a + b + z)$$
$$(A + B \ldots + Z + a + \ldots z + 0 + \ldots 9)^*$$

Example 3.4.14 MU - Dec. 2014, 10 Marks

Design a DFA for the regular expression (a + b)*aba

Solution : The language of the regular expression (a + b)*aba is all strings ending in aba.

The DFA is given below :

State q_0 is the starting state.

State q_1 shows that the preceding character is a

State q_2 shows that the preceding characters are ab

State q_3 shows that the preceding characters are aba.

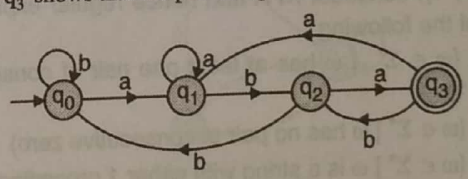


Fig. Ex. 3.4.14

Example 3.4.15

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.

Solution :

(i) R.E. = a^*c^*

(ii) L(r) = {0, 2, 01, 21, 011, 211, 0111, ...}

$$r = 0 + 2 + 01 + 21 + 011 + 211 + 0111 + \ldots$$
$$= 0\,(\varepsilon + 1 + 11 + 111 + \ldots)$$
$$+ 2\,(\varepsilon + 1 + 11 + \ldots) = 01^* + 21^*$$
$$r = (0 + 2)1^*$$

Scanned by CamScanner

(iii) $L(r) = \{00, 010, 0110, 01110, \ldots\}$

R.E. = $0(1)^*0$

(iv) DFA for the given language can be drawn as shown in Fig. Ex. 3.4.15(a).

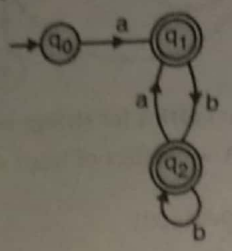


Fig. Ex. 3.4.15(a)

Moving the loop between q_1 and q_2 and q_1 we get :

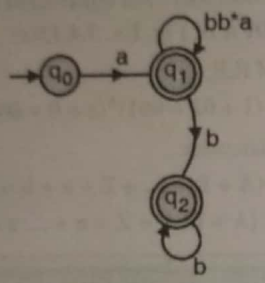


Fig. Ex. 3.4.15(b)

R.E. corresponding to q_1 = $a\,(bb^*a)^*$

R.E. corresponding to q_2 = $a\,(bb^*a)^*bb^*$

$\therefore$ Combined R.E. = $a\,(bb^*a)^* + a\,(bb^*a)^*bb^*$

= $\mathbf{a\,(bb^*a^*) + [\varepsilon + bb^*]}$

Example 3.4.16

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) $L_4 = \{\omega \in \Sigma^* \mid \omega$ 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. 3.4.16(a).

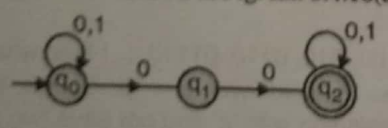


Fig. Ex. 3.4.16(a) : FA for Example 3.4.16(a)

R.E. = $(0+1)^*00\,(0+1)^*$

(b) $L_2 = \{\omega \in \Sigma^* \mid \omega$ has no pair of consecutive zero$\}$

The equivalent FA is shown in Fig. Ex. 3.4.16(b)

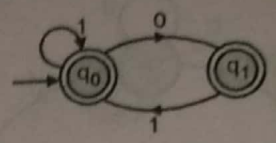


Fig. Ex. 3.4.16(b) : FA for Example 3.4.16(b)

The effect of loop between q_0 and q_1 can be move to q_0. A new machine is obtained and shown in Fig. Ex. 3.4.16(c).

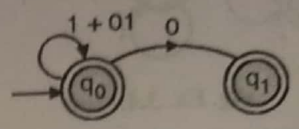


Fig. Ex. 3.4.16(c) : FA after the effect of loop moved to q_0

R.E. for state q_0 = $(1+01)^*$

R.E. for state q_1 = $(1+01)^*0$

$\therefore$ 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. 3.4.16(d).

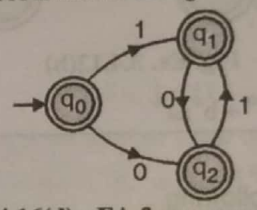


Fig. Ex. 3.4.16(d) : FA for example 3.4.16(c)

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. 3.4.16(e).

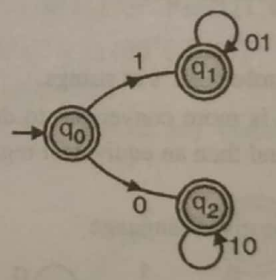


Fig. Ex. 3.4.16(e) : FA after elimination of loop

$\therefore$ **R.E = $\varepsilon + 1(01)^* + 0(10)^*$**

(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. 3.4.16(f)

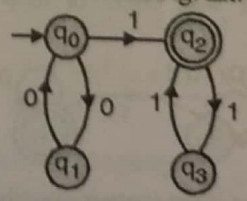


Fig. Ex. 3.4.16(f) : FA for Example 3.4.16(d)

The effect of loop between q_0 and q_1, between q_2 and q_3 are removed and FA is redrawn as shown in Fig. Ex. 3.4.16(g).

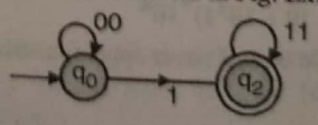


Fig. Ex. 3.4.16(g)

$\therefore$ **R.E. = $(00)^*1(11)^*$**

Scanned by CamScanner

Example 3.4.17

Construct regular expression for following transition diagram :

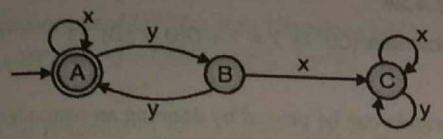
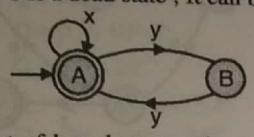


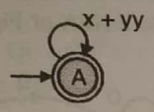
Fig. Ex. 3.4.17

Solution :

Step 1 : The state C is a dead state ; It can be eliminated



Step 2 : The effect of loop between the states A and B can be transferred on the state A.



Step 3 : The regular expression for the given transition diagram $= (x + yy)^*$

Example 3.4.18

Construct regular expression for following transition diagram.

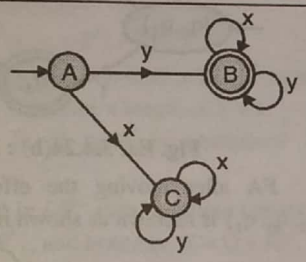
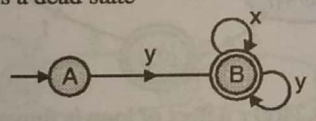


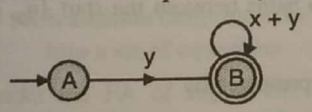
Fig. Ex. 3.4.18

Solution :

The state C is a dead state



The above machine can be equivalently redrawn as



The RE for the above FSM is given by $y\,(x + y)^*$

Example 3.4.19

Write regular expressions for

(i) Set of strings of 0's and 1's whose tenth symbol from the right end is 1.

(ii) Set of string of 0's and 1's not containing 101 as substring.

(iii) Set of strings with even number of a's followed by odd number of b's that is for language.
$L = \{a^{2n}\, b^{2m+1} : n \geq 0, m \geq 0\}$

(iv) Set of strings of an equal number of 0's and 1's such that in every prefix, the number of 0's differs from the number of 1's by at most 1.

Solution :

(i) $(0 + 1)^*\, 1\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)\, (0 + 1)$

(ii) We can write the RE after giving FA for the given language.

Finite automata :

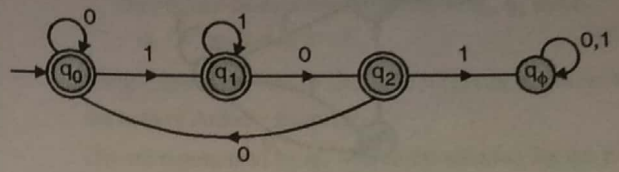


Fig. Ex. 3.4.19

Finite automata to RE :

1. The state q_ϕ can be discarded.
2. The effect of loop among $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$ can be transferred on the state q_0.

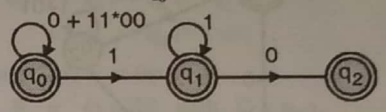


Fig. Ex. 3.4.19(a)

We can write the RE from the above diagram.

$$RE = (0 + 11^*00)^*\, [\varepsilon + 11^* + 11^*0]$$

(iii) $RE = (aa)^*\, (bb)^*\, b$

(iv) $(01 + 10)^*$

Example 3.4.20

Construct the regular expressions for the transition diagrams given.

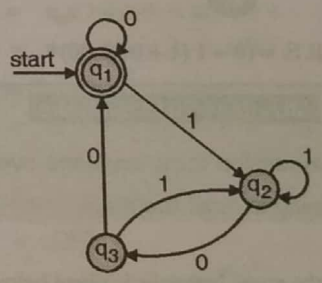
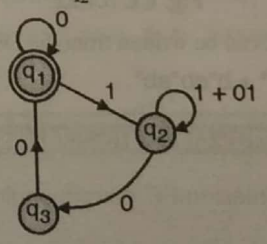


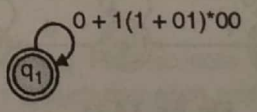
Fig. Ex. 3.4.20

Solution :

1. The effect of loop between the states q_2 and q_3 can be transferred on the state q_2.



2. The effect of loop can be moved on the state q_1.



3. The $RE = (0 + 1\,(1 + 01)^*\, 00)^*$

Scanned by CamScanner

Example 3.4.21

Construct a regular expression corresponding to the state diagram given below :

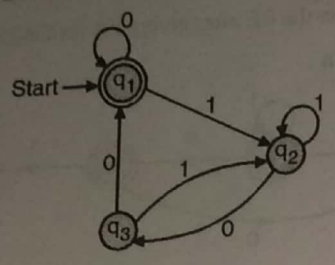


Fig. Ex. 3.4.21

Solution :

Step 1 : Removing loop between the states q_2 and q_3, we get

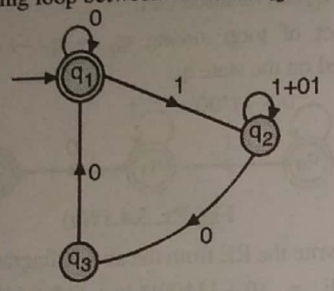
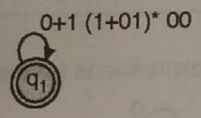


Fig. Ex. 3.4.21(a)

Step 2 : Eliminating states q_2 and q_3, we get



$\therefore$ Required R.E. = $(0 + 1\,(1 + 01)^*\,00)^*$

Example 3.4.22 MU - May 2015, 5 Marks

Give a regular expression for a language over the alphabet $\sum = \{a, b\}$ containing at most two a's.

Solution :

The DFA for the given language is given below

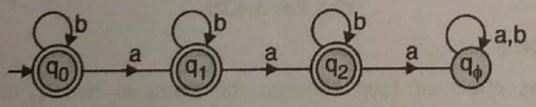


Fig. Ex. 3.4.22

Now, the R. E. can be written from the DFA. The R. E. is

$b^* + b^*ab^* + b^*ab^*ab^*$

Example 3.4.23 MU - May 2015, 5 Marks

Find a regular expression RE corresponding to the following FA.

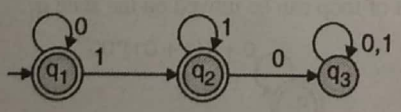


Fig. Ex. 3.4.23

Solution :

The R. E. corresponding to state $q_1 = 0^*$

The R. E. corresponding to state $q_2 = 0^*11^*$

$\therefore$ Required R. E. = $0^* + 0^*\,11^*$

Example 3.4.24

Prove the formula $(00^*1)^*1 = 1 + 0(0 + 10)^*11$

Solution :

The formula can be proved by drawing an equivalent DFA.

DFA for $(00^*1)^*1$

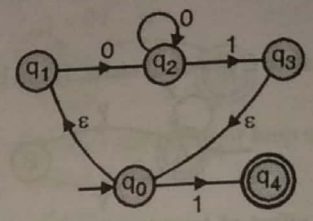


Fig. Ex. 3.4.24(a) : An equivalent ε-NFA

DFA corresponding to ε-NFA of Fig. Ex. 3.4.24(b) is given in Fig. Ex. 3.4.24(b).

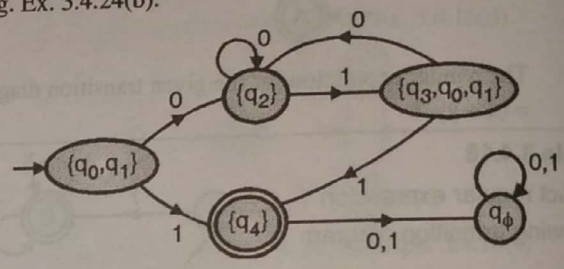


Fig. Ex. 3.4.24(b) : An equivalent DFA

FA after moving the effect of loop between $\{q_2\}$ and $\{q_3, q_0, q_1\}$ is redrawn as shown in Fig. Ex. 3.4.24(c).

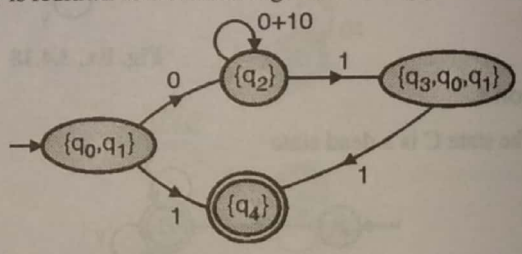


Fig. Ex. 3.4.24(c) : Affect of loop is moved to $\{q_2\}$

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 shown in Fig. Ex. 3.4.24(c) is,

R.E. = $0(0 + 10)^*11 + 1$

Hence the formula is proved.

➜ 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

$$R = Q + RP \quad \text{...(3.16)}$$

has a unique solution given by,

$$R = QP^* \quad \text{...(3.17)}$$

Scanned by CamScanner

Proof : Let us verify whether QP^* is a solution to the equation.

$$R = Q + RP$$

On substitution of QP^* for R in Equation (3.16),

$$R = Q + RP = Q + QP^*P = Q^*(\varepsilon + P^*P) = QP^*$$

Thus the Equation (3.16) is satisfied by $R = QP^*$. We still do not know whether QP^* is a unique solution or not.

To establish uniqueness, we expand

$$\begin{aligned} R &= Q + RP \\ &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \\ &= Q + QP + (Q + RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &\vdots \\ &= Q + QP + QP^2 + \ldots + QP^i + RP^{i+1} \\ &= Q(\varepsilon + P + P^2 + \ldots + P^i) + RP^{i+1} \qquad \ldots(3.18) \end{aligned}$$

here, i is any arbitrary integer.

Let us take a string $\omega \in R \mid$ length of $\omega = k$.

Substituting k for i in Equation (3.18)

$$R = Q(\varepsilon + P + P^2 + \ldots + P^k) + RP^{k+1}$$

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 ω 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(a) Application of Arden's Theorem

A finite automata can be represented using a system of equations :

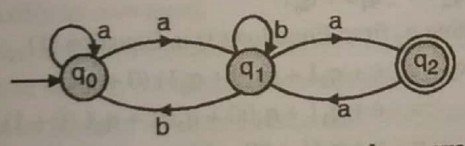


Fig. 3.4.6 : A transition diagram to be converted into a set of equations

Let us consider the FA of Fig. 3.4.6. The set of strings accepted by this FA will be described by a path.

Starting from q_0 and ending q_2.

- A path from q_0 to q_2 will be reached through q_1.
- These strings will end in 'a' with a prefix generated during coming to q_1.
- If the set of strings generated by coming to q_1 from q_0 is $\mathbf{q_1}$ 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 ...(3.19)

$$\mathbf{q_2} = \mathbf{q_1}\, a$$

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 $\mathbf{q_2}$.

$$\mathbf{q_1} = \mathbf{q_0}\, a + \mathbf{q_1}\, b + \mathbf{q_2}\, a \qquad \ldots(3.20)$$

Similarly, the state q_0 can be reached from :

(1) q_0 on a,

(2) q_1 on b,

(3) Without any input as q_0 is a start state.

Thus $\mathbf{q_0}$ can be expressed in terms of $\mathbf{q_0}$, $\mathbf{q_1}$ and ε.

$$\mathbf{q_0} = \mathbf{q_0}\, a + \mathbf{q_1}\, b + \varepsilon \qquad \ldots(3.21)$$

- These three Equations (3.19, 3.20, 3.21) can be solved with the help of Arden's theorem.
- The set represented by $\mathbf{q_2}$ will be the solution for the FA of Fig. 3.4.6. q_2 is a final state.

Solving of equations using Arden's theorem

1. Substituting for $\mathbf{q_2}$ in Equation (3.20). $\mathbf{q_2}$ is given by Equation (3.19)

$$\begin{aligned} \mathbf{q_1} &= \mathbf{q_0}\, a + \mathbf{q_1}\, b + \mathbf{q_1}\, aa \\ &= \mathbf{q_0}\, a + \mathbf{q_1}\,(b + aa) \qquad \ldots(3.22) \end{aligned}$$

Equation (3.22) 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, $$\mathbf{q_1} = \mathbf{q_0} a\,(b + aa)^* \qquad \ldots(3.23)$$

2. Substituting the value of $\mathbf{q_1}$ from Equation (3.23) into Equation (3.21)

$$\begin{aligned} \mathbf{q_0} &= \mathbf{q_0} a + \mathbf{q_0} a\,(b + aa)^* b + \varepsilon \\ &= \varepsilon + \mathbf{q_0}(a + a\,(b + aa)^* b) \qquad \ldots(3.24) \end{aligned}$$

The Equation (3.24) is of the form

$$R = Q + RP \text{ from } Q = \varepsilon,\ R = \mathbf{q_0}$$

$$\text{and } P = (a + a\,(b + aa^*)^* b)$$

and its solution is given by,

$$R = QP^*$$

Thus, $$\begin{aligned} \mathbf{q_0} &= \varepsilon \cdot (a + a\,(b + aa)^* b)^* \\ &= (a + a\,(b + aa)^* b)^* \qquad \ldots(3.25) \end{aligned}$$

3. Substituting the value of $\mathbf{q_0}$ from Equation (3.25) into the Equation (3.23).

$$\mathbf{q_1} = (a + a(b + aa)^*)^* a(b + aa)^* \qquad \ldots(3.26)$$

4. Substituting the value of $\mathbf{q_1}$ from Equation (3.26), into the Equation (3.19).

$$\mathbf{q_2} = (a + a(b + aa)^*)^* a(b + aa)^* a$$

Thus the R.E. representing the FA of Fig. 3.4.6 is given by :

$$\text{R.E.} = (a + a(b + aa)^*)^* a(b + aa)^* a$$

> To use Arden's theorem, the FA should not contain ε-moves.

Example 3.4.25

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.

Scanned by CamScanner

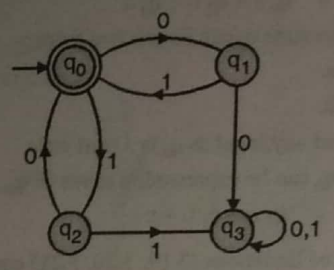


Fig. Ex. 3.4.25

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.

$$q_0 = \varepsilon + q_1 \cdot 1 + q_2 \cdot 0 \quad \ldots(1)$$

$$q_1 = q_0 \cdot 0 \quad \ldots(2)$$

$$q_2 = q_0 \cdot 1 \quad \ldots(3)$$

Substituting values of q_1 and q_2 from Equation (2) and (3) in Equation (1)

$$q_0 = \varepsilon + q_0 01 + q_0 10$$

$$= \varepsilon + q_0(01 + 10) \quad \ldots(4)$$

The Equation (4) is of the form R = Q + RP with Q = ε, 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 3.4.26

Consider the following transition diagram, convert it to an equivalent regular expression using Arden's theorem.

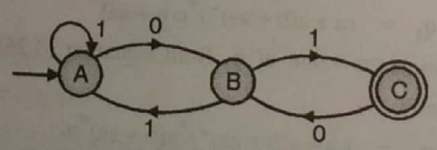


Fig. Ex. 3.4.26

Solution :

The set of equations for regular sets represented by the states A, B and C are given below :

$$A = \varepsilon + A1 + B1 \quad \ldots(1)$$

$$B = A0 + C0 \quad \ldots(2)$$

$$C = B1 \quad \ldots(3)$$

Rewriting Equation (1),

A = (ε + B1) + A1, it is of the from R = Q + RP with R = A; Q = (ε + B1) and P = 1 and its solution is given by R = QP*

$$\therefore A = (\varepsilon + B\,1)1^* \quad \ldots(4)$$

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,

$$B = 1^*0(11^*0 + 10)^* \quad \ldots(5)$$

Substituting the value of B from Equation (5) in Equation (3).

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

Since state C is a final state, the regular expression represented by C describes the FA given in the Example 3.4.26.

$$\therefore \text{R.E.} = 1^*0(11^*0 + 10)^*1$$

Example 3.4.27

Find the regular expression for the set strings recognized by the FA shown in Fig. Ex. 3.4.27. Use Arden's theorem.

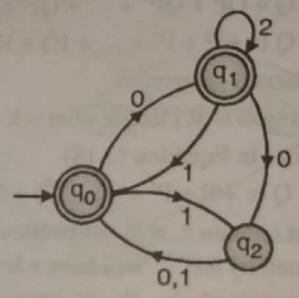


Fig. Ex. 3.4.27

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]

$$= [\varepsilon + (1\,(0 + 1))^*0[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]^*(1 + 00 + 01)]\,(1(0 + 1))^* + (1(0 + 1))^*0\,[2 + (1 + 00 + 01)\,(1(0 + 1))^*0]$$

Scanned by CamScanner

Example 3.4.28

Obtain regular expression for the following FA.

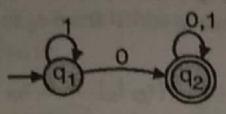


Fig. Ex. 3.4.28

Solution :

The set of equations for regular sets represented by the states q_1 and q_2 are given below.

$$q_1 = \varepsilon + q_1 \cdot 1 \quad \ldots(1)$$

$$q_2 = q_1 \cdot 0 + q_2 \cdot 0 + q_2 \cdot 1 \quad \ldots(2)$$

Applying Arden's theorem on Equation (1), we get.

$$q_1 = 1^* \quad \ldots(3)$$

Putting the value of q_1 from (3) in Equation (1), we get

$$q_2 = 1^* \cdot 0 + q_2 (0 + 1)$$

$$\therefore \quad q_2 = 1^* \cdot 0 \cdot (0 + 1)^*$$

$\therefore$ The regular expression for the given F.A. $= 1^* \cdot 0 \cdot (0 + 1)^*$

3.5 FA Limitations

➔ **(Dec. 2016)**

Q. What are limitations of finite automata ?

MU - Dec. 2016, 5 Marks

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 : Closure Properties

3.6 Closure Properties of Regular Language

➔ **(May 2014)**

Q. State and explain closure properties of regular languages.

MU- May 2014, 5 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.

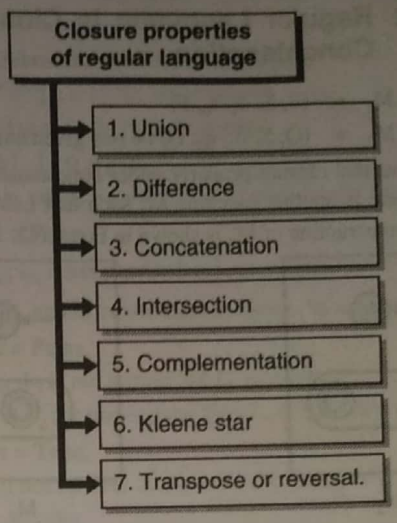


Fig. C3.2 : Closure Properties of Regular Language

➔ 3.6.1 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. 3.6.1.

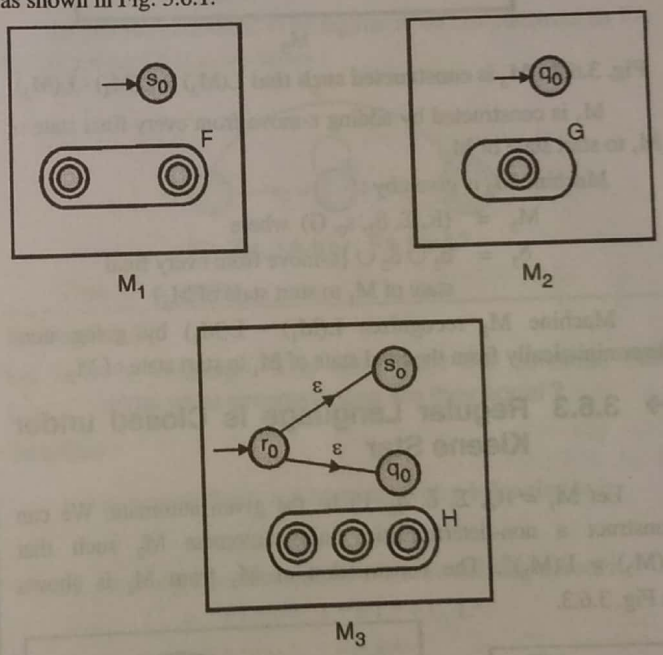


Fig. 3.6.1 : M_3 is constructed such the $L(M_3) = L(M_1) \cup L(M_2)$

Machine M_3 is constructed to accept $L(M_1) \cup L(M_2)$.

$M_3 = (R, \Sigma, \delta_3, r_0, H)$ where r_0 is a new start state. Two ε-moves, one from r_0 to s_0 and

$$R = S \cup Q \cup (r_0\}$$

$$H = F \cup G$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(r_0, \varepsilon, s_0), (r_0, \varepsilon, q_0)\}$$

Machine M_3 can non-deterministically choose either M_1 or M_2. Therefore,

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

Scanned by CamScanner

→ 3.6.2 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. 3.6.2.

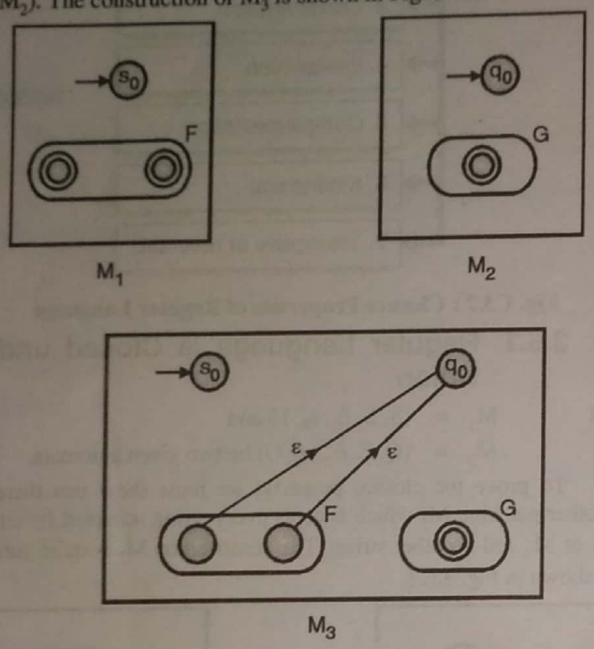


Fig. 3.6.2 : M_3 is constructed such that $L(M_3) = L(M_1) \cdot L(M_2)$

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.

→ 3.6.3 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. 3.6.3.

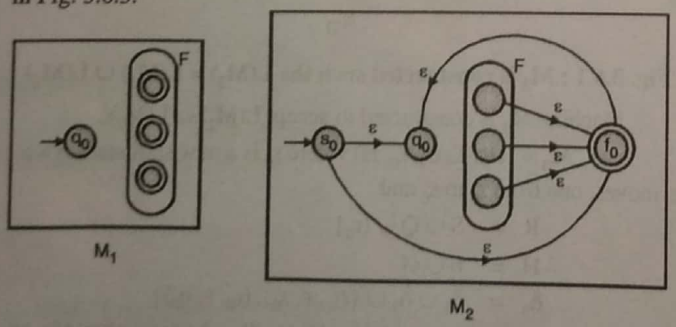


Fig. 3.6.3 : M_2 is constructed such that $L(M_2) = L(M_1)^*$

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 $L(M_1)^*$.

→ 3.6.4 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

$$\underbrace{\overline{L(M)}}_{\text{Given machine}} = \underbrace{L(\overline{M})}_{\text{Machine after complementation}} = \Sigma^* - L(M)$$

If M is a deterministic finite automata then $\overline{M}$ can be constructed by interchanging final and non final states of M.

$$\therefore \overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

→ 3.6.5 Regular Language is Closed under Intersection

If L_1 and L_2 are two regular languages, then

$$L_1 \cap L_2 = ((L_1 \cap L_2)')' = (\overline{L}_1 \cup \overline{L}_2)'$$
$$= \Sigma^* - [(\Sigma^* - L_1) \cup (\Sigma^* - L_2)]$$

Closeness under intersection follows directly from closeness under union and complementation.

→ 3.6.6 Regular Languages are Closed under Difference

Let L_1 and L_2 are two regular languages. The difference $L_1 - L_2$ is the set of strings that are in language L_1 but not in L_2. Construction of a composite automata for $L(M_1) - L(M_2)$ is already explained in Chapter 2. Thus regular languages are closed under difference.

→ 3.6.7 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.

Scanned by CamScanner

Example 3.6.1

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 :

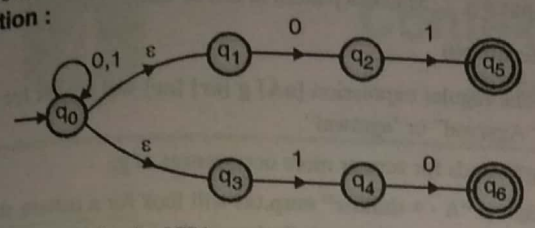


Fig. Ex. 3.6.1(a) : An NFA accepting strings ending in 01 or 10

If L is a language representing strings ending in 01 or 10. A machine for L^R is shown in Fig. Ex. 3.6.1(b).

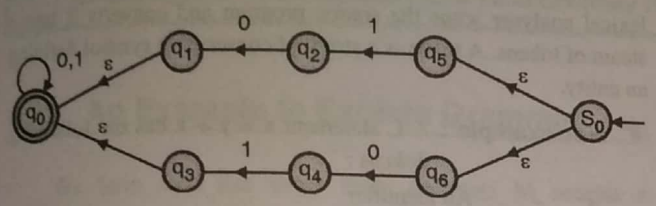


Fig. Ex. 3.6.1(b) : An NFA accepting L^R

Example 3.6.2 :

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 3.6.3

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 3.6.4

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 :

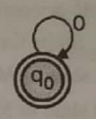


Fig. Ex. 3.6.4 : DFA for L

L* can be considered to be regular if we can construct an FA for L*. FA for L* is given below :

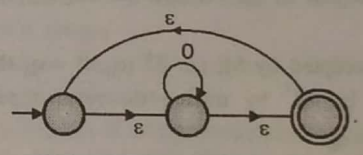


Fig. Ex. 3.6.4(a) : FA for L*

Thus, L* is regular.

Example 3.6.5

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.

Example 3.6.6

Are the following True or False. Support your answer with proofs or counter examples.

(i) If $L_1 \cup L_2$ is regular and L_1 is regular then L_2 is regular.

(ii) If $L_1 \cdot L_2$ is regular and L_1 is regular then L_2 is regular.

(iii) If L* is regular, then L is regular.

Scanned by CamScanner

Solution :

(i) If $L_1 \cup L_2$ is regular and L_1 is regular then L_2 is regular.
– We can always construct a combined
FA for $M(L_1 \cup L_2) - M(L_1)$
$M(L_2)$ is same as $M(L_1 \cup L_2) - M(L_1)$
$\therefore L_2$ is regular (True)

(ii) If $L_1 \cdot L_2$ is regular and L_1 is regular then L_2 is regular.
– True [If L_2 is not regular then $L_1 \cdot L_2$ can not be regular]

(iii) If L^* is regular, then L is regular.
– True
Existence of an FA for L^* will trivially imply existence of an FA for L.

Example 3.6.7

If L is regular, prove L^T is also regular.

Solution : Let us assume L = L(M) for some FA = (Q,Σ,δ,q_0,F)
For simplicity, we can assume that F contains only one state q_n.

$$\therefore F = \{F_n\}$$

The set of states

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

The Finite Automate M^T accepting L^T can be derived from the given Finite Automata M. Steps for the same are given below :

1. q_n will become the start state in M^T.
2. q_0 will become the final state in M^T.
3. For every transition $\delta(q_i,a) = q_j$ (where $a \in \Sigma$), a transition $\delta(q_j, a) = q_i$ is added to M^T.

– It is clear that M^T is FA and the language of M^T must be regular.

– If ω is accepted by M, i.e. $\delta^*(q_0,\omega) = q_n$ then ω^T will be accepted by M^T by tracing the reverse path. Therefore, $\delta^*(q_n,\omega) = q_0$.

3.7 Application of RE

3.7.1 R.E. in Unix

The UNIX regular expression lets us specify a group of characters using a pair of square brackets [ ]. The rules for character classes are :

1. **[ab]** Stand for a + b
2. **[0 – 9]** Stand for a digit from 0 to 9
3. **[A – Z]** Stands for an upper-case letter
4. **[a – z]** Stands for a lower-case letter
5. **[0 – 9A-Za – z]** Stands for a letter or a digit.

The **grep** utility in UNIX, scans a file for the occurrence of a pattern and displays those lines in which the given pattern is found.

For example :

$ grep president emp.txt

It will list those lines from the file emp.txt which has the pattern "president". The pattern in grep command can be specified using regular expression.

6. * matches zero or more occurrences of previous character.
7. ● matches a single character.
8. **[^ pqr]** Matches a single character which is not a p, q or r.
9. **^ pat** Matches pattern **pat** at the beginning of a line
10. **pat $** Matches pattern at end of line.

☞ **Example**

a) The regular expression **[aA] g [ar] [ar] wal** stands for either "Agarwal" or 'agrawal".

b) **g*** stands for zero or more occurrences of g.

c) **$grep "A · * thakur" emp.txt** will look for a pattern starting with **A.** and ending with **thakur** in the file emp.txt.

3.7.2 Lexical Analysis

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.

3.8 Exam Pack (University Questions)

☞ **Syllabus Topic : Regular Expression, Regular Languages**

Example 3.1.4 (6 Marks) **(Dec. 2016)**

Example 3.1.5 (4 Marks) **(Dec. 2016)**

Example 3.3.8 (5 Marks) **(May 2017)**

Example 3.3.19 (10 Marks) **(May 2017)**

☞ **Syllabus Topic : FA to RE**

Example 3.4.14 (10 Marks) **(Dec. 2014)**

Example 3.4.22(5 Marks) **(May 2015)**

Example 3.4.23 (10 Marks) **(May 2015)**

Q. What are limitations of finite automata ?
(Refer section 3.5) (5 Marks) **(Dec. 2016)**

☞ **Syllabus Topic : Closure Properties**

Q. State and explain closure properties of regular languages. *(Refer section 3.6)* (5 Marks) **(May 2014)**

□□□

Scanned by CamScanner

CHAPTER 4

Context Free Grammars and Languages

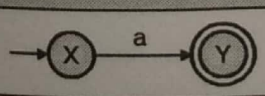
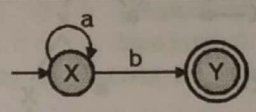
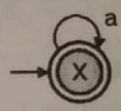
Syllabus Topics

Context Free Languages : CFG, Leftmost and Rightmost derivations, Ambiguity, Simplification and Normalization (CNF) and Chomsky Hierarchy (Types 0 to 3)

4.1 An Example to Explain Grammar

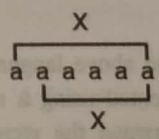
- We have seen that every finite automata M accepts a language L, which is represented by L (M).
- We have seen that a regular language can be described by a regular expression.
- We have seen that there are several languages which are not regular.
  1. $L1 = \{a^p \mid p \text{ is a prime}\}$ is not regular.
  2. $L2 = \{a^n b^n \mid n \geq 0\}$ is not regular.
  3. $L3 = \{ai^2 \mid i \geq 1\}$ 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 finite automata.
  2. Using a regular expression.
- If a language is non-regular, it can not be represented either using a FA or using a regular expression. Hence there was a need for representing such languages.
- Grammar is another approach for representing a language.
  1. In this approach, a language is represented using a set of equations.
  2. Equations are recursive in nature.
  3. A finite automata has a set of states. A context free grammar has a set of variables.
  4. Finite automata is defined over an alphabet. Similarly, a grammar is defined over a set of terminals.
  5. A finite automata has a set of transitions; a grammar has a set of equations (productions).

Table 4.1.1 : A set of DFA represented using an equivalent set of productions

Sr. No.	DFA	Equivalent grammar
1.		$X \rightarrow a$
2.		$X \rightarrow aX$ $X \rightarrow b$
3.		$X \rightarrow aX$ $X \rightarrow \varepsilon$

- The production $X \rightarrow a$ should read as X produces a or X derives a or X gives a.
- The production $X \rightarrow aX$ is a way of defining a string containing one or more a's.

If X stands for a set of strings containing one or more a's then a can be expressed in terms of X. The concept is shown below in Fig. 4.1.1.



A string of 6 a's can be generated by X
A string of 5 a's can be generated by X.

Fig. 4.1.1 : Recursive definition of $X \rightarrow aX$

A string of 6 a's = a followed by a string of 5 a's.

Or $X = aX$

Every recursive definition has a base case or termination case.

Let us consider the DFA given below in Fig. 4.1.2.

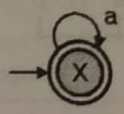


Fig. 4.1.2 : DFA taken as an example

The language of the DFA is given below

$$L = \{\varepsilon, a, aa, aaa, \ldots\}$$

Now, let us consider the grammar given below and see if it can generate the language generated by the DFA of Fig. 4.1.2.

Scanned by CamScanner

$$X \rightarrow aX \quad \ldots(4.1)$$
$$X \rightarrow \varepsilon \quad \ldots(4.2)$$

- The empty string ε can be generated by the production $X \rightarrow \varepsilon$
- A string "a" can be generated as shown in Fig. 4.1.3.

First by writing X as aX and then by replacing X on the right hand side with an ε.

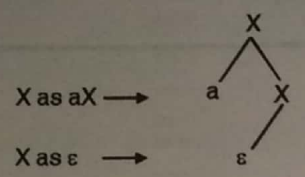


Fig. 4.1.3

- A string "aa" can be generated as shown in Fig. 4.1.4.

First by writing X as aX, secondly by replacing X on RHS with aX and finally replacing X on RHS with ε.

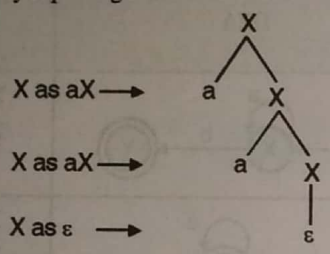


Fig. 4.1.4

- 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

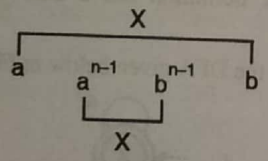
Let us consider a language

$$L = \{0^n 1^n \mid n \geq 1\}$$

We know from pumping lemma that the above language is not regular; but this language can be represented using a set of productions. It will be worthwhile to understand the recursive nature of the above language.

1. A string $0, 1 \in L$ [Base case or termination case]
2. $a^n b^n$ can be written as $a(a^{n-1}b^{n-1})b$

If $a^n b^n$ is generated by X then $a^{n-1}b^{n-1}$ can also be generated by X.



Now, we can say that,

$$X \rightarrow aXb$$

i.e. If we have a string of the form $a^n b^n$, after removing the first a and the last b, the string remains in the form $a^n b^n$.

Thus the language.

$$L = \{a^n b^n \mid n \geq 1\} \text{ can be represented using}$$

1. $X \rightarrow aXb \quad \ldots(4.3)$
2. $X \rightarrow ab \quad \ldots(4.4)$

The production $X \rightarrow aXb$ can recursively generate a string $a^n b^n$ and the second production $X \rightarrow ab$ is used for termination of recursion.

Let us try to generate a string $a^3 b^3$ using the two productions (4.3) and (4.4).

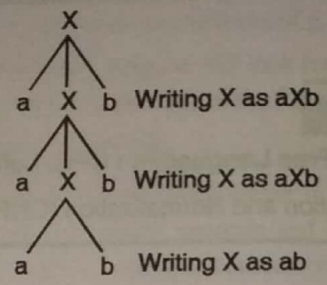


Fig. 4.1.5 : Generation of $a^3 b^3$

☞ **Conclusion :** Grammar can be used to represent both.

1. Regular languages 2. Non-regular languages.

Syllabus Topic : CFG

4.2 Context Free Grammar

➔ **(Dec. 2016, May 2017)**

Q. Define CFG. MU - Dec. 2016, May 2017, 2/5 Marks

A context free grammar G is a quadruple (V, T, P, S), where,

V is a set of variables.
T is a set of terminals.
P is a set of productions.
S is a special variable called the start symbol $S \in V$.

A production is of the form

$V_i \rightarrow \alpha_i$ where $V_i \in V$ and α_i is a string of terminals and variables.

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. Soham plays.
3. Ram reads.

Scanned by CamScanner

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> → <noun> <verb>

Here, noun can be replaced with Mohan, Soham or Ram and <verb> can be replaced with reads, plays or eats. We can write :

<noun> → Mohan
<noun> → Soham
<noun> → Ram
<verb> → eats
<verb> → plays
<verb> → 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> → <noun> <verb>,
<noun> → Mohan,
<noun> → Soham,
<noun> → Ram,
<verb> → eats,
<verb> → plays,
<verb> → 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> → <noun> <verb>,
<noun> → Mohan | Soham | Ram,
<verb> → eats | plays | reads
}

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

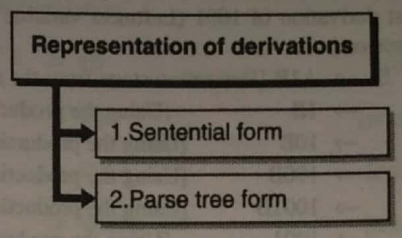


Fig. C4.1 : Representation of derivations

Syllabus Topic : Leftmost and Rightmost Derivations

1. Sentential Form

Let us consider a grammar given below :

S → A1B (Production 4.5)
A → 0A | ε (Production 4.6)
B → 0B | 1B | ε (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 → A1B [Starting production]
→ 0A1B [Using the production A → 0A]
→ 00A1B [Using the production A → 0A]
→ 001B [Using the production A → ε]
→ 0010B [Using the production B → 0B]
→ 00101B [Using the production B → 1B]
→ 00101 [Using the production B → ε]

Thus the string 00101 ∈ 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 → A1B A → 0A | ε B → 0B | 1B | ε

Give leftmost and rightmost derivation of the string 1001.

Scanned by CamScanner

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 $A \rightarrow \varepsilon$]

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

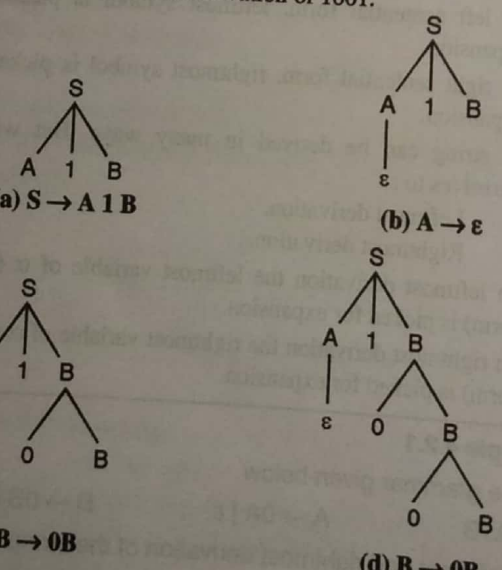
For the grammar given below

$S \rightarrow A1B$
$A \rightarrow 0A \mid \varepsilon$
$B \rightarrow 0B \mid 1B \mid \varepsilon$

Give parse tree for leftmost and rightmost derivation of the string 1001.

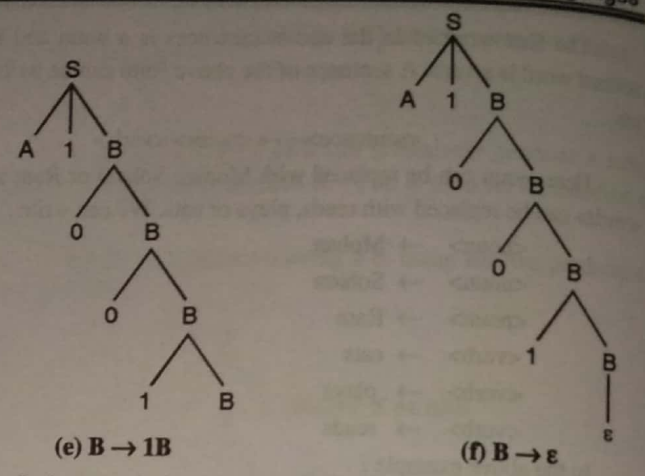
Solution :

(i) Parse tree for leftmost derivation of 1001.



(a) $S \rightarrow A\,1\,B$ (b) $A \rightarrow \varepsilon$ (c) $B \rightarrow 0B$ (d) $B \rightarrow 0B$

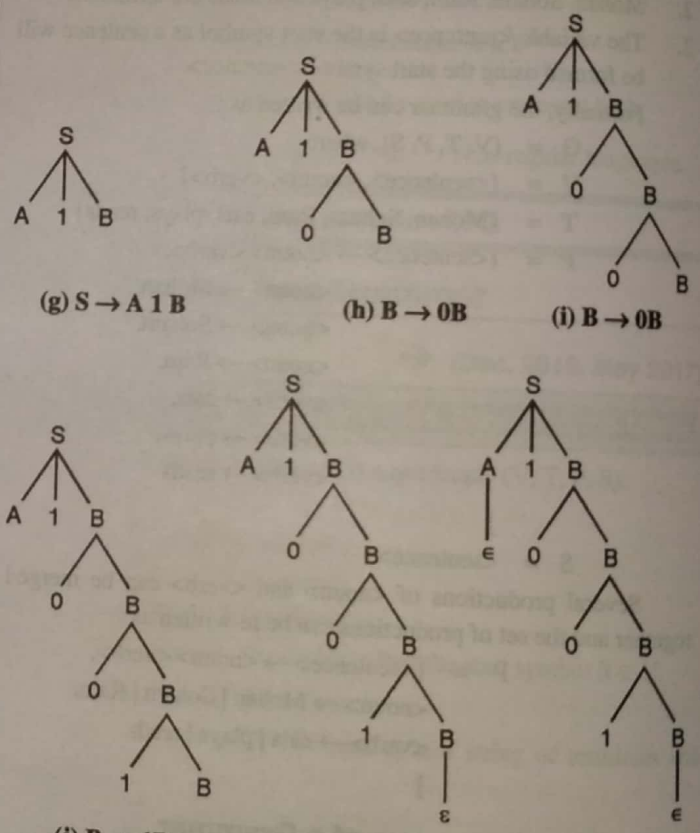
Fig. Ex. 4.2.2



(e) $B \rightarrow 1B$ (f) $B \rightarrow \varepsilon$

Leftmost derivation of 1001 is shown by series of Figs. 4.2.2(a) to 4.2.2(f).

(ii) Parse tree for rightmost derivation of 1001.



(g) $S \rightarrow A\,1\,B$ (h) $B \rightarrow 0B$ (i) $B \rightarrow 0B$

(j) $B \rightarrow 1B$ (k) $B \rightarrow \varepsilon$ Fig. (*l*) : $A \rightarrow \varepsilon$

Fig. Ex. 4.2.2

Rightmost derivation of 1001 is shown by a series of Fig. Ex. 4.2.2(g) to Ex. 4.2.2(*l*).

It may be noted that the final parse trees Ex. 4.2.2(*l*) and Ex. 4.2.2(*l*) are identical.

Example 4.2.3

For the grammar given below $S \rightarrow 0S1 \mid 01$

Give derivation of 000111.

Solution :

1. Derivation in sentential form

$S \rightarrow 0S1$ [Using production $S \rightarrow 0S1$]
$S \rightarrow 00S11$ [Using production $S \rightarrow 0S1$]

Scanned by CamScanner

$S \rightarrow 000111$ [Using production $S \rightarrow 01$]

2. Derivation using parse tree.

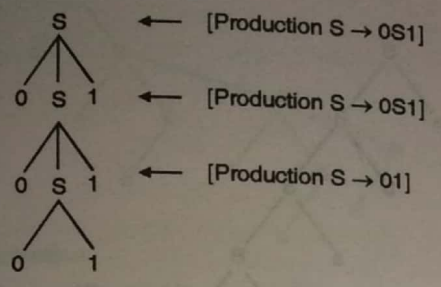


Fig. Ex. 4.2.3 : Derivation of 000111

Example 4.2.4

Find whether the string aabbb is in $L = L(G)$, where G is given by $S \rightarrow XY$, $X \rightarrow YY|a$, $Y \rightarrow XY \mid b$

Solution :

Parse tree for the string aabbb is shown in Fig. Ex. 4.2.4.

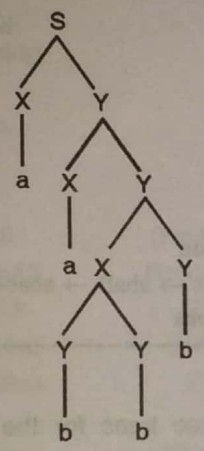


Fig. Ex. 4.2.4 : Derivation of aabbb

Hence, aabbb $\in$ L(G).

Example 4.2.5 MU - Dec. 2014. 10 Marks

Let G be the grammar. Find the leftmost derivation, rightmost derivation and parse tree for the string 001222

G : $S \rightarrow 0S \mid 1A \mid 2B \mid \varepsilon$
$A \rightarrow 1A \mid 2B \mid \varepsilon$
$B \rightarrow 2B \mid \varepsilon$

Solution :

(i) Parse Tree for the string 001222

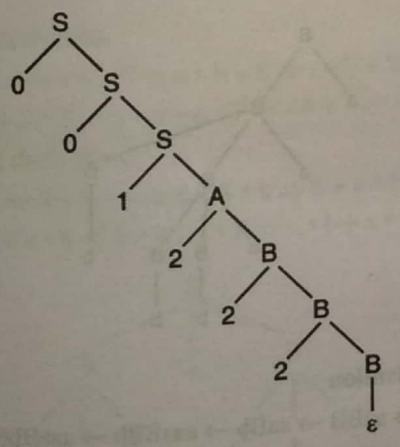


Fig. Ex. 4.2.5

(ii) Leftmost Derivation

$S \Rightarrow 0S$ [using $S \rightarrow 0S$]

$\Rightarrow 00S$ [using $S \rightarrow 00S$]

$\Rightarrow 001A$ [using $S \rightarrow 1A$]

$\Rightarrow 0012B$ [using $A \rightarrow 2B$]

$\Rightarrow 00122B$ [using $B \rightarrow 2B$]

$\Rightarrow 001222B$ [using $B \rightarrow 2B$]

$\Rightarrow 001222$ [using $B \rightarrow \varepsilon$]

Example 4.2.6

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 the derivation of $(a + b) * a + b$.

Solution :

For the grammar given in Example 4.2.5,

Set of variables V = {E, T, F}

Set of terminals Σ = {+, *, (,), a, b}

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 $F \rightarrow a \mid b$]

Scanned by CamScanner

(ii) Derivation using parse tree :

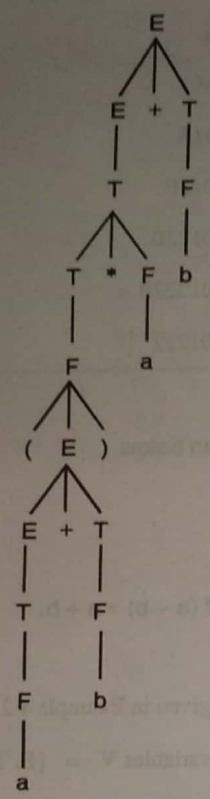


Fig. Ex. 4.2.6 : Parse tree for (a + b) * a + b

Example 4.2.7

Construct the parse trees for the strings using specified derivation format for given grammar G.

G = ({S, A, B}, {a, b}, P, {S})

P = {S → aB, S → bA,

A → a, A → aS, A → bAA,

B → b, B → bS, B → aBB}

Strings :

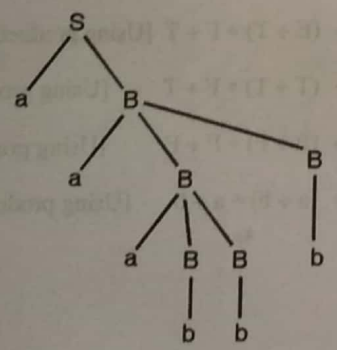
(I) a a a b b b (leftmost derivation)

(II) a b a b a b b a (rightmost derivation).

Solution :

(I) aaabbb

Parse tree

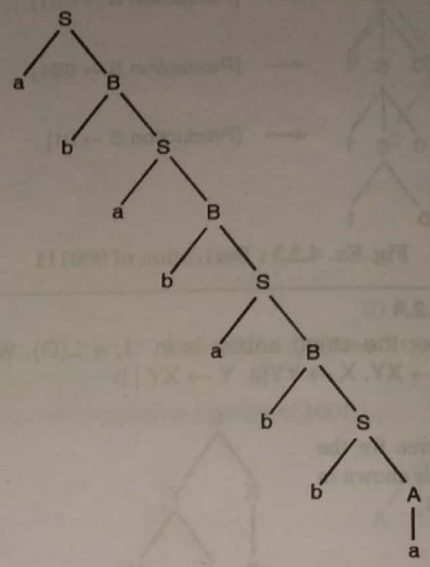


Leftmost derivation

S → aB → aaBB → aaaBBB → aaabBB → aaabbB → aaabbb

(II) abababba

Parse tree



Rightmost derivation

S → aB → abs → abaB → ababs → ababaB → abababs → ababababbA → ababababba

Example 4.2.8

Construct the parse trees for the strings using specified derivation format for given grammar G.

G = ({S, A, B}, {a, b}, P, {S})

P = {S → aB, S → bA,

A → a, A → aS, A → bAA,

B → b, B → bS, B → aBB}

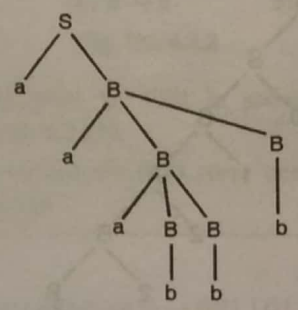
Strings : (I) a a a b b b (rightmost derivation)

(II) a a b a b b (leftmost derivation)

Solution :

(I) aaabbb (rightmost derivation)

Parse Tree



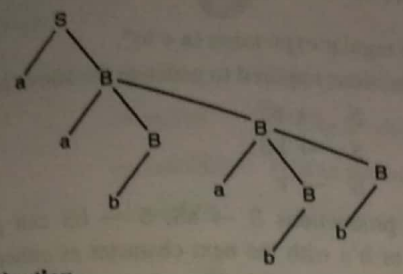
Rightmost derivation

S → aB → aaBB → aaBb → aaaBBb → aaaBbb → aaabbb

Scanned by CamScanner

(II) **aababb (leftmost derivation)**

Parse Tree



Leftmost derivation

$S \to aB \to aaBB \to aabB \to aabaBB \to aababB \to aababb$

Example 4.2.9

Consider the following grammar

$S \to aAS \mid a$

$A \to SbA \mid SS \mid ba$

Derive the string aabbaa using

(i) Leftmost derivation

(ii) Rightmost derivation.

Solution :

(i) Leftmost derivation

$S \to aAS$		[Production $S \to aAS$]
$\to aSbAS$		[Production $A \to SbA$]
$\to aabAS$		[Production $S \to a$]
$\to aabbaS$		[Production $A \to ba$]
$\to aabbaa$		[Production $S \to a$]

(ii) Rightmost derivation

$S \to aAS$		[Production $S \to aAS$]
$\to aAa$		[Production $S \to a$]
$\to aSbAa$		[Production $A \to SbA$]
$\to aSbbaa$		[Production $A \to ba$]
$\to aabbaa$		[Production $S \to a$]

Example 4.2.10 MU - May 2014, 5 Marks

Let G be the grammar. Find the leftmost derivation, rightmost derivation and parse tree for the expression a*b + a* b

$G : S \to S + S \mid S * S$

$S \to a \mid b$

Solution :

1. **Leftmost derivation**

$S \Rightarrow S + S \Rightarrow S * S + S \Rightarrow a * S + S \Rightarrow a * b + S$

$\Rightarrow a * b + S * S \Rightarrow a * b + a * S \Rightarrow a * b + a * b$

2. **Rightmost derivations**

$S \Rightarrow S + S \Rightarrow S + S * S \Rightarrow S + S * b \Rightarrow S + a * b$

$\Rightarrow S * S + a * b \Rightarrow S * b + a * b \Rightarrow a * b + a * b$

3. **Parse tree**

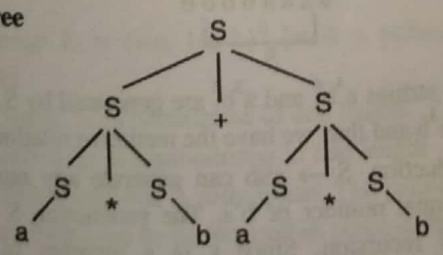


Fig. Ex. 4.2.10

4.2.3 Writing Grammar for a Language

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) $S \to aS$ is recursive

(2) $S \to b \mid aA$

$A \to c \mid bS$ is indirectly recursive

as,

$S \to aA \to abS$ [S gives A and A gives S]

$A \to bS \to baA$ [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 = {ε, a, aa, ...}**

The productions required to generate the above language are :

$S \to aS$

$S \to \varepsilon$

The production, $S \to \varepsilon$ is taken as ε is a member of L.

The production, $S \to aS$ can generate one or more a's.

☞ **Example :** Generation of the string aaaa.

Sentential form

$S \to$	aS	[Using production $S \to aS$]
$\to$	aaS	[Using production $S \to aS$]
$\to$	aaaS	[Using production $S \to aS$]
$\to$	aaaaS	[Using production $S \to aS$]
$\to$	aaaa	[Using production $S \to \varepsilon$]

Parse tree

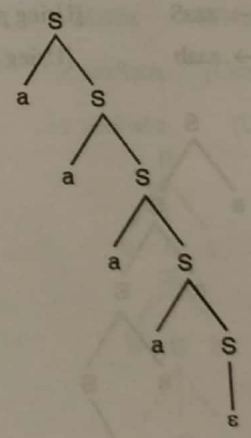


Fig. 4.2.1 : Parse tree for aaaa

(2) **Language L = {a, aa, aaa,...}**

The productions required to generate the above language are :

$S \to aS$

$S \to a$

The production $S \to aS$ can generate one or more a's. The production, $S \to a$ is for termination of recursion.

Scanned by CamScanner

☞ **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

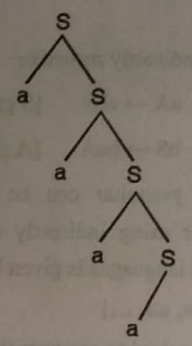


Fig. 4.2.2 : Parse tree for aaaa

(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

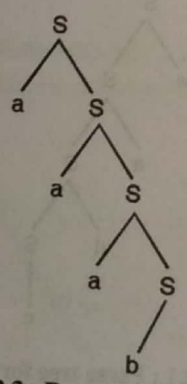


Fig. 4.2.3 : Parse tree for aaab

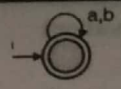
(4) Language $L = \{w \in \{a, b\}^*\}$

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

$S \rightarrow aS$ [Using production $S \rightarrow aS$]
$\rightarrow abS$ [Using production $S \rightarrow bS$]
$\rightarrow abbS$ [Using production $S \rightarrow bS$]
$\rightarrow abbaS$ [Using production $S \rightarrow aS$]
$\rightarrow abba$ [Using production $S \rightarrow \varepsilon$]

Parse tree

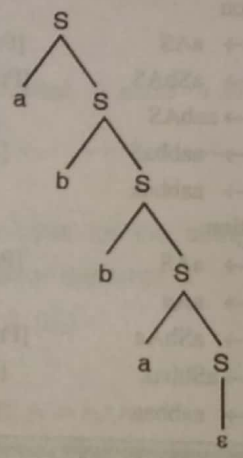


Fig. 4.2.4 : Parse tree for abba

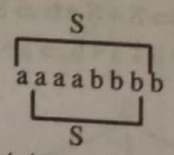
(5) Language $L = \{\varepsilon, ab, aabb, \ldots, a^n b^n\}$

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.

Scanned by CamScanner

☞ **Example :** Generation of the string aaabbb.

Sentential form

$S \rightarrow aSb$ [Using production $S \rightarrow aSb$]

$\rightarrow aaSbb$ [Using production $S \rightarrow aSb$]

$\rightarrow aaaSbbb$ [Using production $S \rightarrow aSb$]

$\rightarrow aaabbb$ [Using production $S \rightarrow \varepsilon$]

Parse tree

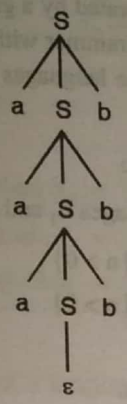


Fig. 4.2.5 : Parse tree for aaabbb or a^3b^3

(6) **Language** $L = \{ab, aabb, \ldots, a^nb^n\}$

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

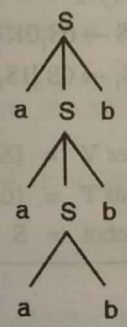


Fig. 4.2.6 : Parse tree for aaabbb or a^3b^3

(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 shown in Fig. 4.2.7.

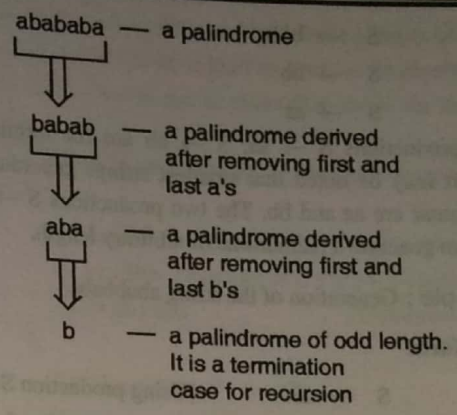


Fig. 4.2.7

Thus a palindrome can be generated recursively using the two productions :

$S \rightarrow aSa$

$S \rightarrow bSb$

Since, we are considering odd length palindromes, S should terminate in either 'a' or 'b' and the productions are :

$S \rightarrow a, S \rightarrow b.$

Thus the productions required to generate the above language are :

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow a$

$S \rightarrow b$

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

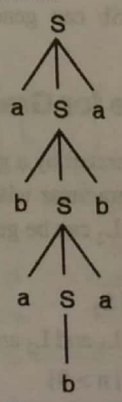


Fig. 4.2.8 : Parse tree for abababa

(8) **Language L = {w∈ {a, b}* | w is an even length palindrome with | w | > 0}**

The productions required to generate the above language are :

$S \rightarrow aSa$

Scanned by CamScanner

$S \rightarrow bSb$

$S \rightarrow bb$

$S \rightarrow aa$

The productions $S \rightarrow aa$, $S \rightarrow bb$ are for termination of recursion. It may be noted that smallest strings generated by the above grammar are aa and bb. The two productions $S \rightarrow aSa$ and $S \rightarrow bSb$ can generate a palindrome of arbitrary length.

Example : Generation of the string ababbaba

Sentential form

$S \rightarrow aSa$ [Using production $S \rightarrow aSa$]

$\rightarrow abSba$ [Using production $S \rightarrow bSb$]

$\rightarrow abaSaba$ [Using production $S \rightarrow aSa$]

$\rightarrow ababbaba$ [Using production $S \rightarrow bb$]

Parse tree or derivation tree

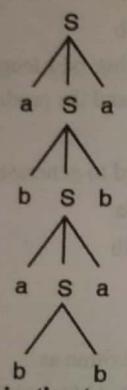


Fig. 4.2.9 : Derivation tree for ababbaba

(9) Language $L = \{w \in \{a, b\}^* \mid w$ is a palindrome of either even length or odd length with $|w| > 0\}$

The productions required to generate the above language are :

$S \rightarrow aSa \mid bSb \mid aa \mid bb \mid a \mid b$

- $S \rightarrow aa$ or $S \rightarrow bb$ are for termination of recursion for even length palindromes.
- $S \rightarrow a$ and $S \rightarrow b$ are for termination of recursion for odd length palindromes.
- $S \rightarrow aSa$ or $S \rightarrow bSb$ can generate a palindrome of an arbitrary length.

4.2.3(a) 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(b) 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 4.2.11

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 4.2.12

Construct the context free grammar corresponding to the regular expression.

$R = (0 + 1)1^*(1 + (01)^*)$

Solution : Grammar for $(0 + 1)$ is given by

$$S_1 \rightarrow 0 \mid 1 \quad \text{[union rule]}$$

Grammar for 1^* is given by

$$S_2 \rightarrow 1S_2 \mid \varepsilon$$

Grammar for $(1 + (01)^*)$ is given by

Scanned by CamScanner

$S_3 \rightarrow 1 \mid S_4$ 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

$$R = (0+1)1^*(1+(01)^*)$$

$$P = \{ \quad S \rightarrow S_1S_2S_3$$
$$S_1 \rightarrow 0 \mid 1$$
$$S_2 \rightarrow 1S_2 \mid \varepsilon$$
$$S_3 \rightarrow 1 \mid S_4$$
$$S_4 \rightarrow 01S_4 \mid \varepsilon$$
$$\}$$

where,

Set of variables V = $\{S_1, S_2, S_3, S_4\}$

Set of terminals T = $\{0, 1\}$

Start symbol = S

Example 4.2.13

Write a CFG to represent a language defined by the regular expression a*b*.

Solution : We can write the set of productions using the rule of concatenation.

$$S \rightarrow S_1 S_2$$
$$S_1 \rightarrow a S_1 \mid \in$$
$$S_2 \rightarrow b S_2 \mid \in$$

Note : S_1 generates a* and S_2 generates b*

∴ The required CFG

$$G = (\{S, S_1, S_2\}, \{a, b\}, \{S \rightarrow S_1 S_2, S_1 \rightarrow aS_1 \mid \in, S_2 \rightarrow a S_2 \mid \in\}, S)$$

Example 4.2.14

Construct a grammar for L which consists of strings over {0, 1}* with atleast one occurrence of '000'.

Solution :

The regular expression for the set of given strings is $(0+1)^*000\,(0+1)^*$.

We can write the set of productions using the rule of concatenation.

$S \rightarrow S_1 000 S_1$

$S_1 \rightarrow 0\,S_1 \mid 1\,S_1 \mid \in$

Note : S_1 generates strings from $(0+1)^*$

∴ The required CFG

$G = (\{S, S_1\}, \{0, 1\}, \{S \rightarrow S_1, 000\,S_1, S_1 \rightarrow 0\,S_1 \mid 1\,S_1 \mid \in\}, S)$

Example 4.2.15

Give the CFG for $L = \{a^i b^j, i \le j \le 2i, i \ge 1\}$

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 :

$$P = \{$$
$$S \rightarrow aSb \quad \text{[Number of b's = Number of a's]}$$
$$S \rightarrow aSbb \quad \text{[Number of b's = 2 × Number of a's]}$$
$$S \rightarrow ab \mid abb \quad \text{[Termination cases]}$$
$$\}$$

where, Set of variables V = {S}

Set of terminals T = {a, b}

Start symbol = S.

Example 4.2.16

Obtain a grammar to generate the language

$L = \{a^{n+2} b^n \mid n \ge 0 \text{ and } m > n\}$

Solution :

The above language can be written as

$$L = \{a^n a^2 b^n \mid n \ge 0\}$$

The set of productions for the above language are given by :

$$P = \{$$
$$S \rightarrow aSb \mid aa$$
$$\}$$

Thus the required grammar

$$G = (\{S\}, \{a, b\}, \{S \rightarrow aSb \mid aa\}, S)$$

Example 4.2.17

Give CFG for

$L = \{a^i b^j c^q ; i + j = q; i, j \ge 1\}$

Solution :

A string of the form $a^i b^j c^q$ with i + j = q can be written as $a^i b^j c^{i+j}$

A string of the form $a^i c^i$ can be generated with the help of the following productions :

$$S \rightarrow aSc$$
$$S \rightarrow X$$

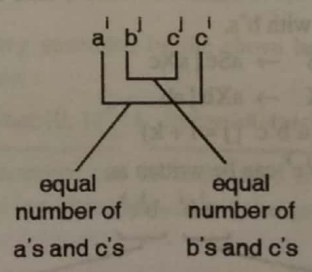


Fig. Ex. 4.2.17

Recursion is terminated by the production S → X, where X gives a string of the form $b^j c^j$.

$$X \rightarrow bXc$$
$$X \rightarrow bc$$

Scanned by CamScanner

Thus, the set of productions for the above language are given by :

$$P = \{ \quad S \to aSc \mid X, \quad X \to bXc \mid bc \quad \}$$

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

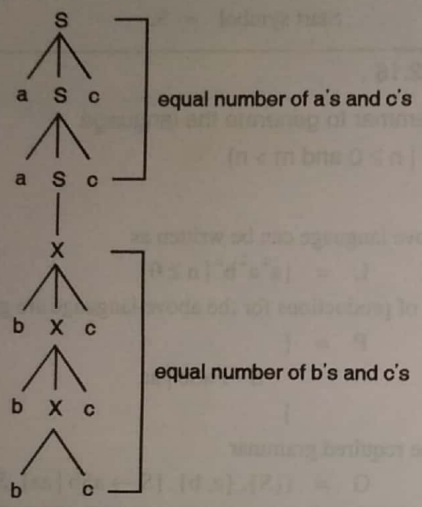


Fig. Ex. 4.2.17(a) : Parse tree for aabbbccccc

Example 4.2.18

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.

$$S \to aSc \mid aXc$$
$$X \to aXb \mid ab$$

(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

$$\underbrace{a^i b^i}_{\text{a's and b's are equal (generated by X)}} \quad \underbrace{b^k c^k}_{\text{B's and c's are equal (generated by Y)}}$$

$$S \to XY$$
$$X \to aXb \mid ab$$
$$Y \to bYc \mid bc$$

(3) $L_3 = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

$S \to AB$	[i = j]
$S \to CD$	[j = k]
$A \to aAb \mid ab$	[equal number of a's and b's followed by some c's]
$B \to CB \mid c$	
$C \to aC \mid a$	[Some a's followed by equal number of b's and c's]
$D \to b\,DC \mid bc$	

Example 4.2.19

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.

$$\therefore X \to 011X \mid 1X \mid \varepsilon$$

Let us assume that (01)* is generated by the variable Y.

$$\therefore Y \to 01Y \mid \varepsilon$$

Using the concatenation rule, the set of productions for (011 + 1)* (01)* can be written as :

$$P = \{ \quad S \to XY, \quad X \to 011X \mid 1X \mid \varepsilon, \quad Y \to 01Y \mid \varepsilon \quad \}$$

where,

the set of variable V = {S,X,Y}
the set of terminals T = {0,1}
the start symbol = S

(b) CFG for $0^i 1^{i+k} 0^k$ | 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 :

$$P = \{ \quad S \to XY, \quad X \to 0X1 \mid \varepsilon, \quad Y \to 1Y0 \mid \varepsilon \quad \}$$

Where,

the set of variable V = {S, X, Y}
the set of terminals T = {0, 1}
the start symbol = S

Scanned by CamScanner

Example 4.2.20

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.

(1) 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

$Y \rightarrow cY|c$ generates c^j]

(2) Productions for $L_2 = \{a^i b^j c^j \mid i,j >= 1\}$

$S \rightarrow XY$ [X generates a^i and Y

$X \rightarrow aX|a$ generates $b^j c^j$]

$Y \rightarrow bYc|bc$

Example 4.2.21

Give CFG for

$L = \{0^i 1^j 0^k \mid j > i + k\}$

Solution :

A string of the form $0^i 1^j 0^k$ with $j > i + k$ can be written as $0^i 1^{i+k+p} 0^k$ 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 :

$$P = \{ S \rightarrow XYZ$$
$$X \rightarrow 0X1 \mid \varepsilon$$
$$Y \rightarrow 1Y \mid 1$$
$$Z \rightarrow 1Z0 \mid \varepsilon$$
$$\}$$

where,

Set of variables V = {X, Y, Z}

Set of terminals T = {0, 1}

Start symbol = S

Example 4.2.22

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 :

$$P = \{S \rightarrow (S) \mid SS \mid \varepsilon\}$$

where, Set of variables V = {S}

Set of terminals T = {(,)}

Start symbol = S

Example 4.2.23

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.

$$\therefore X \rightarrow 1X \mid \varepsilon$$

Let us assume that $(0+1)^*$ is generated by the variable Y.

$$\therefore Y \rightarrow 0Y \mid 1Y \mid \varepsilon$$

Using the concatenation rule, the set of productions for $1^*01^*0(0+1)^*$ can be written as :

$$P = \{ S \rightarrow X0X0Y$$
$$X \rightarrow 1X \mid \varepsilon$$
$$Y \rightarrow 0Y \mid 1Y \mid \varepsilon$$
$$\}$$

where, the set of variables V = {S, X, Y}

the set of terminals T = {0, 1}

the start symbol = S.

Example 4.2.24

Give CFG for set of odd length strings in $\{0, 1\}^*$ with middle symbol '1'.

Solution : A string generated by the above language could be shown as given below :

$$\underbrace{\text{String of length n from } \{0,1\}^*}\ 1\ \underbrace{\text{String of length n from } \{0,1\}^*}$$

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 is given by :

$$P = \{$$
$$S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 1$$
$$\}$$

The production $S \rightarrow 1$, terminates the recursion with 1 as a middle character.

Scanned by CamScanner

where, the set of variables V = {S}
the set of terminals T = {0, 1}
the start symbol = S.

Example 4.2.25

Give CFG for set of even length strings in $\{a, b, c, d\}^*$ with two middle symbol equal.

Solution : A string generated by the above language could be shown as given below :

$$\underbrace{\text{String of length n from } \{a, b, c, d\}^*}\begin{bmatrix} aa \\ bb \\ cc \\ dd \end{bmatrix}\underbrace{\text{String of length n from } \{a, b, c, d\}^*}$$

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 :

$$\begin{aligned} P = \{ & S \to aSa \mid aSb \mid aSc \mid aSd \mid \\ & bSa \mid bSb \mid bSc \mid bSd \mid \\ & cSa \mid cSb \mid cSc \mid cSd \mid \\ & dSa \mid dSb \mid dSc \mid dSd \mid \\ & aa \mid bb \mid cc \mid dd \\ \} & \end{aligned}$$

where, set of variables V = {S}
set of terminals T = {a, b, c, d}
start symbol = S

Example 4.2.26

Give CFG for $L = \{a^m b^n c^p d^q \mid m + n = p + q\}$

Solution :

There could be three possibilities on number of a's and number of d's.

Case I : $m > q$, $a^m b^n c^p d^q$ can be written as

$$a^q \underbrace{a^{m-q} b^n c^p}_{X} d^q$$

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 :

$$X \to aXc \mid Z$$
$$Z \to bZc \mid \varepsilon$$

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 :

$$Y \to bYd \mid Z$$
$$Z \to bZc \mid \varepsilon$$

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 :

$$\begin{aligned} P = \{ & S \to aSd \mid X \mid Y \\ & X \to aXc \mid Z \\ & Y \to bYd \mid Z \\ & Z \to bZc \mid \varepsilon \\ \} & \end{aligned}$$

where, Set of variables V = {S, X, Y, Z}
Set of terminals T = {a, b, c, d}
Start symbol = S.

Example 4.2.27

Give CFG for $L = \{a^n b^m c^n \mid m, n >= 1\}$

Solution : The production $S \to aSc$ can generate equal number of a's and b's.

Thus the set of productions for the given language is :

$$\begin{aligned} P = \{ & S \to aSc \mid X \\ & X \to bX \mid b \\ \} & \end{aligned}$$

The production $S \to X$, terminates the recursion. The variable X can generate one or more b's.

where, Set of variables V = {S, X}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.28 MU - May 2015, 5 Marks

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 $S \to aB$,
  B represents a string in which number of b's is one more than a's. If we pretend an 'a' to 'B', both a's and b's will become equal.
- Similarly, the relation between S and A is given by

Scanned by CamScanner

$S \rightarrow bA$

- 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 :

$$P = \{ \quad S \rightarrow aB \mid bA, \quad A \rightarrow aS \mid bAA \mid a, \quad B \rightarrow bS \mid aBB \mid b \}$$

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

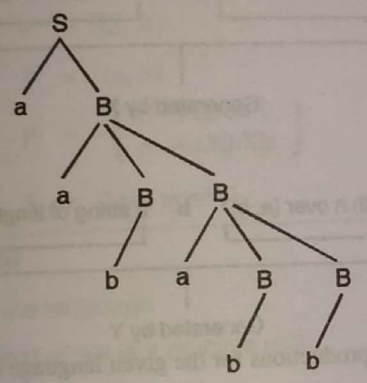


Fig. Ex. 4.2.28 : Parse tree for aababb

Example 4.2.29

Give CFG for strings in ab^*

Solution : Let us say that b^* is generated by variable B.

$\therefore B \rightarrow bB \mid \varepsilon$

The set of productions for the above language can be written using the rule of concatenations.

$$P = \{ \quad S \rightarrow aB, \quad B \rightarrow bB \mid \varepsilon \}$$

where, Set of variables V = {S, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.30

Give CFG for strings in a^*b^*.

Solution : Let us assume that a^* and b^* are generated by variable A and B respectively.

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

The set of productions for the above language can be written using the rule of concatenation.

$$P = \{ \quad S \rightarrow AB, \quad A \rightarrow aA \mid \varepsilon, \quad B \rightarrow bB \mid \varepsilon \}$$

where, Set of variables V = {S, A, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.31

Give CFG for $L = \{wcw^T \mid w \in \{a, b\}^*\}$

Solution : L is a language of odd length palindromes with middle character as c. The set of productions for the above language is given by :

$S \rightarrow aSa \mid bSb \mid c$

where,
Set of variables V = {S}
Set of terminals T = {a, b, c}
Start symbol = S.

Example 4.2.32

Give CFG for $L = \{a^n b^m \mid n \neq m\}$rob

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 :

$$P = \{ \quad S \rightarrow aSb \mid A \mid B, \quad A \rightarrow aA \mid a, \quad B \rightarrow bB \mid b \}$$

where, Set of variables V = {S, A, B}
Set of terminals T = {a, b}
Start symbol = S.

Example 4.2.33

Give CFG for the language $L = \{a^n b^m c^k \mid n = m \text{ or } m \leq k\}$

Solution : The language L is union of two languages L_1 and L_2.

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.

$A = aAb \mid \varepsilon$

$B = cB \mid \varepsilon$

The productions for language $\{L_2 = a^n b^m c^m c^p \mid p \geq 0\}$ are given as :

$S_2 = XYB,$

Scanned by CamScanner

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.

$$L = L_1 \cup L_2$$

$$P = \{ \quad S \rightarrow S_1 \mid S_2$$
$$S_1 \rightarrow AB$$
$$A \rightarrow aAb \mid \varepsilon$$
$$B \rightarrow cB \mid \varepsilon$$
$$S_2 \rightarrow XYB$$
$$X \rightarrow aX \mid \varepsilon$$
$$Y \rightarrow bYc \mid \varepsilon \quad \}$$

where, The set of variables V = $\{S, S_1, S_2, A, B, X, Y\}$

The set of terminals T = $\{a, b\}$

The start symbol = S.

Example 4.2.34

Give CFG for the language $L = \{0^m 1^n 0^{m+n} \mid m, n >= 0\}$

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 :

$$P = \{ \quad S \rightarrow XYX$$
$$X \rightarrow 0X \mid \varepsilon$$
$$Y \rightarrow 1Y0 \mid \varepsilon \quad \}$$

where, The set of variables V = $\{S, X, Y\}$

The set of terminals T = $\{0, 1\}$

The start symbol = S.

Example 4.2.35

Give CFG for the language

$L = \{0^a 1^b 2^c \mid a - c = b\}$

Solution : The string $0^a 1^b 2^c$ with $a - c = b$

Can be written as $0^{c+b} 1^b 2^c$

$$= \underbrace{0^c \underbrace{0^b \; 1^b} \; 2^c}$$

The set of productions for the given language is :

$$P = \{ S \rightarrow 0S2 \mid X \text{ [X generates a string of the form } 0^b 1^b]$$
$$X \rightarrow 0X1 \mid \varepsilon \quad \}$$

where, The set of variables V = $\{S, X\}$

The set of terminals T = $\{0, 1\}$

Start symbol = S.

Example 4.2.36

Given the context free grammar for the following languages :

(a) $L = \{a^n b^{2n} \mid n > 1\}$ (b) $L = \{a^m b^n \mid n > m\}$

Solution :

(a) $L = \{a^n b^{2n} \mid n > 1\}$

$$S \rightarrow aSbb \mid aabbbb$$

(b) $L = \{a^m b^n \mid n > m\}$

$$S \rightarrow aSb \mid B$$
$$B \rightarrow bB \mid b$$

Example 4.2.37

Give CFG for the following language : The set of odd length strings in $\{a, b\}^*$ whose first, middle and last symbols are all same.

Solution : A string belonging to the language will have any of the two forms, given below.

Form 1 :

$$\underbrace{a \; \underbrace{\text{a string of length n over } \{a, b\}^*} \; a \; \underbrace{\text{a string of length n over } \{a, b\}^*} \; a}_{\text{Generated by X}}$$

Form 2

$$\underbrace{b \; \underbrace{\text{a string of length n over } \{a, b\}^*} \; b \; \underbrace{\text{a string of length n over } \{a, b\}^*} \; b}_{\text{Generated by Y}}$$

The set of productions for the given language is :

$$P = \{ \quad S \rightarrow aXa \mid bYb$$
$$X \rightarrow aXa \mid aXb \mid bXa \mid bXb \mid a$$
$$Y \rightarrow aYa \mid aYb \mid bYa \mid bYb \mid b \quad \}$$

where, The set of variables V = $\{S, X, Y\}$

The set of terminals T = $\{a, b\}$

The start symbol = S.

Example 4.2.38

Write a CFG for the following languages

$L = \{ua\omega b \mid u, \omega \text{ are in } \{a, b\}^* \text{ and } |u| = |\omega|\}$

Solution : A string generated by the above language could be shown as given below :

$$\underbrace{\underbrace{\text{String of length n from } \{a, b\}^*} \; a \; \underbrace{\text{String of length n from } \{a, b\}^*} \; b}_{\text{Generated by X}}$$

The hidden recursion in X can be understood from the fact that deletion of first and last character will not change the basic nature of X.

Scanned by CamScanner

$\therefore$ X is a string of the given form then X are :

$$X \rightarrow aXa \mid aXb \mid bXa \mid bXb$$

Thus, the set of productions for the above language is given by,

$$P = \{ S \rightarrow Xb, \; X \rightarrow aXa \mid aXb \mid bXa \mid bXb \mid a \}$$

The production $X \rightarrow a$, terminates the recursion with a as the middle character of X.

The set of variables V = {S, X}
The set of terminals T = {a, b}
The start symbol = S

Example 4.2.39

Write a CFG which generates the language L defined by the regular expression (a + b)*bbb(a + b)*

Solution :

Let us assume that the start symbol is S.

$\therefore$ $S \rightarrow XbbbX$,

where X generates a string in (a + b)*

$X \rightarrow aX \mid bX \mid \varepsilon$

The CFG, G = (V, T, P, S)

where, V = (S, X)

T = {a, b}

$$P = \left\{ \begin{array}{l} S \rightarrow XbbbX \\ x \rightarrow aX|bX|\varepsilon \end{array} \right\}$$

S = Start symbol

Example 4.2.40

Write CFG for the language

Σ = {a, b} number of a's is a multiple of 3.

Solution :

We can design a DFA for strings in which number of a's is multiple of 3.

The set of productions can be written from the DFA :

$S \rightarrow bS \mid aX \mid \varepsilon$

$X \rightarrow bX \mid aY$

$Y \rightarrow aS \mid bY$

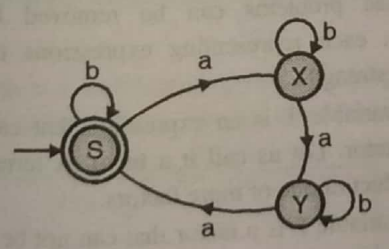


Fig. Ex. 4.2.38

Example 4.2.41 MU - May 2017, 5 Marks

Define context free grammar obtain the CFG for the following regular expression : (110 + 11)* (10)*

Solution :

$S \rightarrow 110S \mid 11S \mid SX \mid X$

$X \rightarrow 10X \mid \in$

Example 4.2.42

Consider the following grammar :

$S \rightarrow aSa \mid bSb \mid aa \mid bb$

Show that

1. L(G) has no strings of odd length.
2. Any string in L(G) is of length 2n, n > 0
3. The number of strings of length 2n is 2^n.

Solution :

1. Each production from ($S \rightarrow aSa \mid bSb \mid aa \mid bb$) has two terminals on the right side. Terminals are always generated in pair. Therefore, L(G) has no strings of odd length.
2. Smallest string generated by S is either aa or bb with length 2. The language of the given grammar is a set of even length palindromes. Therefore, any string in L(G) is of length 2n, n > 0.
3. Let us take a string of length 2n (say S_{2n})

$$S_{2n} = aS_{2n-2}a \mid bS_{2n-2}b$$

Thus, we can write a recurrence relation for S_{2n}

$$S_{2n} = 2 \times S_{2n-2} = 2^2 S_{2n-y} = 2^{n-1} \times S_2 = 2^n$$

Example 4.2.43

Construct a context free grammar G generating all integers (with sign). Derive an example integer.

Solution :

The set of production P is given by

P = {

$S \rightarrow +I \mid -I \mid I$

$I \rightarrow 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \mid 7X \mid 8X \mid 9X$

$X \rightarrow 0X \mid 1X \mid 2X \mid 3X \mid 4X \mid 5X \mid 6X \mid 7X \mid 8X \mid 9X \mid \varepsilon$

}

The grammar G is given by

G = ({S, I, X}, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, P, S)

Example considered for derivation is the string + 95.

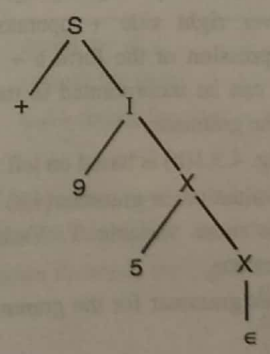


Fig. Ex. 4.2.43

Syllabus Topic : Ambiguity

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.

Scanned by CamScanner

Example : Let us consider the grammar given below :

$E \rightarrow E + E \mid a \mid b$ ...(Grammar 4.8)

A string a + b + a is generated by the given grammar.

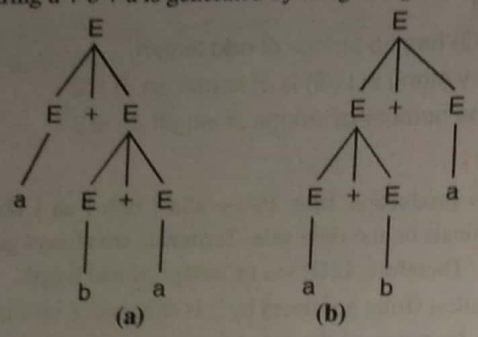


Fig. 4.3.1 : Parse trees considered for ambiguity

- The grammar 4.8, generates a + b + a in two different ways. The two derivations are shown in Fig. 4.3.1(a) and 4.3.1(b).
- The first derivation (Fig. 4.3.1(a)) says that (b + a) is evaluated first and then the evaluated value is added to a. Thus the right side + operator gets a precedence over the left side + operator. The expression a + b + a is treated as (a + (b + a)).
- The second derivation (Fig. 4.3.1(b)) says that a + b is evaluated first and then the evaluated value is added to a. Thus the left side + operator will get precedence over right side + operator. The expression a + b + a is treated as ((a + b) + a).

Removing ambiguity : There is no general rule for removing ambiguity from CFG. Removing ambiguity from a grammar involves rewriting of grammar so that there is only one derivation tree for every string belonging to L(G) i.e., language generated by grammar G.

Ambiguity from the grammar

$$E \rightarrow E + E \mid a \mid b$$

can be removed by strictly assigning higher precedence to left side + operator over right side + operator. This will mean evaluation of an expression of the form a + b + a,, from left to right. This property can be incorporated in the grammar itself by suitably modifying the grammar.

- Parse tree of Fig. 4.3.1(b) is based on left to right evaluation.
- Left to right evaluation in grammar(4.8) can be enforced by introducing one more variable T. Variable T can not be broken by + operator.
- An unambiguous grammar for the grammar of (4.8) is given in (4.9)

$$E \rightarrow E + T \mid T$$
$$T \rightarrow a \mid b \quad \text{...(Grammar 4.9)}$$

- A production of the form $E \rightarrow E + T$ provides a binding. E + T, implies that E must be evaluated first before an atomic T can be added to it. E can be broken down in E + T but T cannot be broken further. This ensures higher precedence to left side + operator over right side + operator.
- Parse tree in Fig. 4.3.2 is based on unambiguous grammar (4.9), for the string a + b + a.

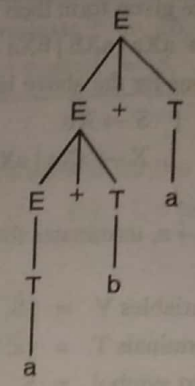


Fig. 4.3.2 : A parse tree for a + b + a using an unambiguous grammar

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."

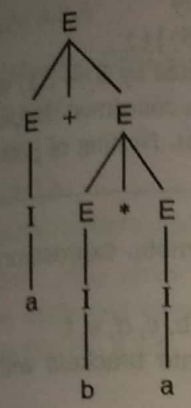
- A variable T is an expression that cannot be broken by + operator. Let us call it a **term**. A term can be expressed as product of one or more factors.
- A variable F is a factor that can not be broken by + operator or * operator. A factor is
  (a) Identifiers
  (b) Literals
  (c) Parenthesized expressions

An unambiguous grammar is given below :

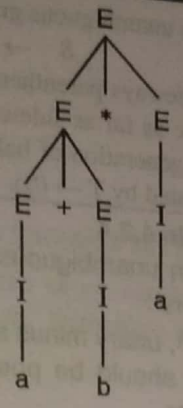
$$\left.\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid a \mid b \end{aligned}\right\} \quad \text{(Grammar 4.10)}$$

A parse tree for a + b * a using both ambiguous and unambiguous grammar is shown in Fig. Ex. 4.3.1.

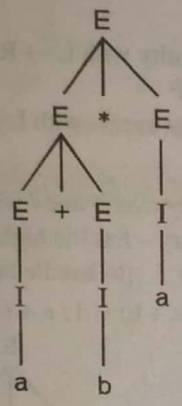
Scanned by CamScanner



(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 Ex. 4.3.1(b) are using the ambiguous grammar.

Fig. Ex. 4.3.1(c) is using an unambiguous grammar.

The grammar given in the example is an ambiguous grammar as there are two different parse trees for the expression a + b * a (shown in Fig. Ex. 4.3.1(a) and 4.3.1(b)).

Example 4.3.2

Give an ambiguous grammar for if-then-else statement and then re-write an equivalent unambiguous grammar.

Solution :

The dangling else is a well known problem in programming language. If statement in C-language can be written in any of the two forms :

```
Form A :   if (x > y)
           a = a + 1 ;
Form B :   if (x > y)
           a = a + 1
           else
           a = a – 1;
```

We can substitute the condition x > y with C and S for a = a + 1. With these substitutions, if statement in two forms can be written as :

```
if   C then S
if   C then S1 else S2
```

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.

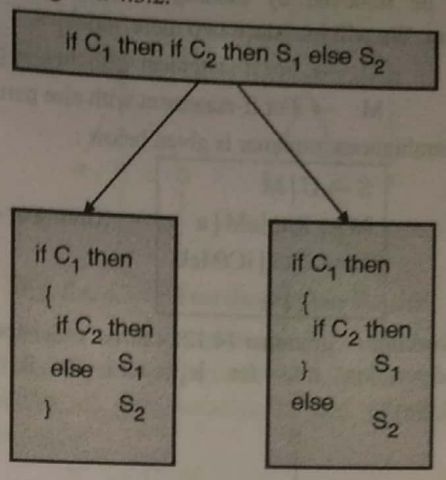


Fig. Ex. 4.3.2 : Two interpretations of a nested if-statement

– A nested if statement can have two interpretations.

Problem

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 → if C then S
    | if C then S else S
    | a  [termination of a statement by 'a']
C → c1 | c2
```

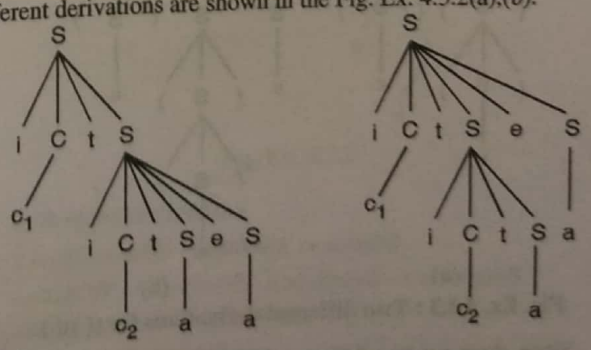
The above grammar can be written in short as :

```
S → iCtS|iCtSeS|a
C → c1 | c2
```
– (Grammar 4.11)

Where, i stands for if,
t stands for then,
and e stands for else

Example :

Grammar (4-11) can be shown to be ambiguous by drawing two different derivation trees for the string ic_1tic_2taea. Two different derivations are shown in the Fig. Ex. 4.3.2(a),(b).



(a) (b)

Fig. Ex. 4.3.2 : Two derivations for ic_1tic_2taea

Scanned by CamScanner

Removing ambiguity : The ambiguity from the grammar (4.11) can be removed by binding dangling else with inner if – statement. We will introduce two more variables.

U → For if-statement without else part.

M → For if-statement with else part.

An unambiguous grammar is given below :

$$\boxed{\begin{array}{l} S \to U \mid M \\ M \to iCtMeM \mid a \\ U \to iCtS \mid iCtMeU \\ C \to c_1 \mid c_2 \end{array}} \quad - \text{ (Grammar 4.12)}$$

The working of grammar (4.12) can be understood with the help of derivation tree for ic_1tic_2taea. It is shown in Fig. Ex. 4.3.2(c).

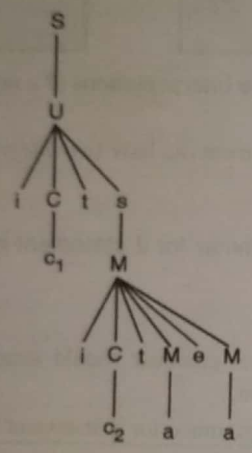


Fig. Ex. 4.3.2(c) : Derivation tree for ic_1tic_2taea

- Outer if-statement is without else part.
- Inner if-statement is with else part.

Example 4.3.3

Show that the CFG given below. Which generates all strings of balanced parentheses is ambiguous. Give an equivalent unambiguous grammar.

S → SS | (S) | ε

Solution : Let us consider derivation of the string (( ))( ). Two derivations are shown in Fig. Ex. 4.3.3(a) and 4.3.3(b).

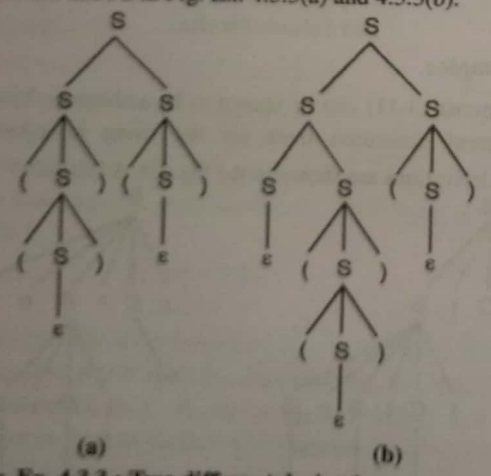


Fig. Ex. 4.3.3 : Two different derivations for (( ))( ).

Since, there are two different derivation for the string (( ))( ) the grammar is ambiguous.

An unambiguous grammar is given by

S → ST | T ; T → (S) | ( )

Sideways parentheses can be generated by S → 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 → (S).

Example 4.3.4

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 → E + T | T

[+ has lowest priority with L → R associativity]

T → T * F | T / F | F

[* and / has higher priority over + with L → R associativity]

F → F ^ G | G

[^ has higher priority over * and / with L → R associativity]

G → – H | H [unary – has the highest priority]

H → a | b | c | d | e | f | (E) [to handle brackets and identifiers]

Derivation tree for (a + b) ^ d / e + (– f)

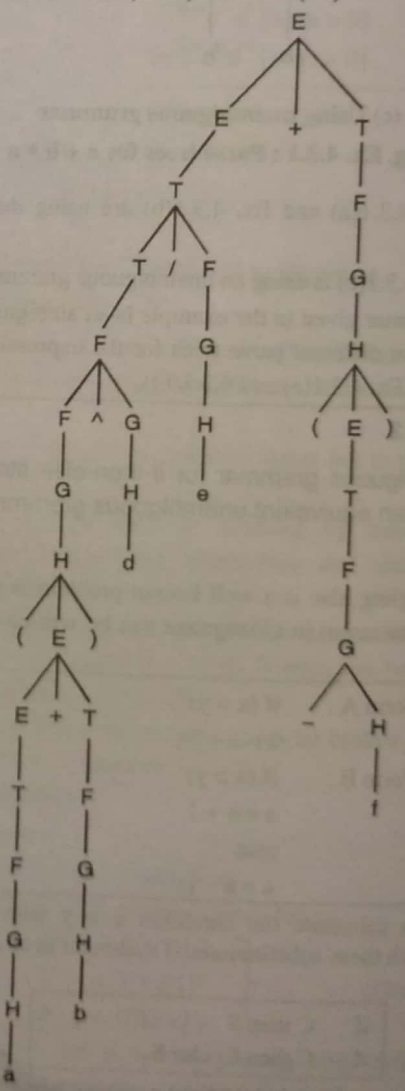


Fig. Ex. 4.3.4 : Derivation tree for (a + b) ^ d / e + (– f)

Scanned by CamScanner

Example 4.3.5

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. 4.3.5.

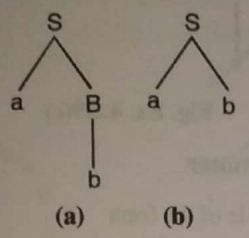


Fig. Ex. 4.3.5 : Two derivations for ab

Example 4.3.6 MU - Dec. 2016, 10 Marks

Is the following CFG ambiguous ?

G = ({S, A}, {a, b}, P, S)

where P consists of

$S \rightarrow aAS \mid a$

$A \rightarrow SbA \mid SS \mid ba$

Solution : The above grammar can be proved to be an ambiguous grammar by deriving a string a^7 in two different ways. Derivation of a^7 is shown in Fig. Ex. 4.3.6.

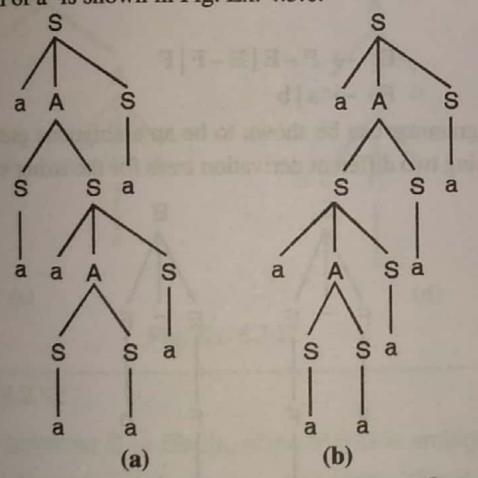


Fig. Ex. 4.3.6 : Two derivations for a^7

Example 4.3.7

Consider the grammar having productions :

$S \rightarrow aS \mid \varepsilon$

$S \rightarrow aSbS$

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. 4.3.7.

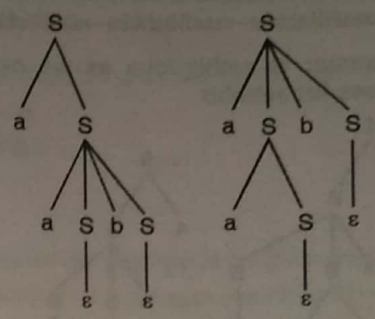


Fig. Ex. 4.3.7 : Two derivations for aab

(ii) The set of productions resemble productions for if-statement. Thus the ambiguity can be removed in a similar way through introduction of two variables U and M with resulting productions is :

$$S \rightarrow U \mid M$$
$$U \rightarrow aS \mid aMbU$$
$$M \rightarrow aMbM \mid \varepsilon$$

Example 4.3.8

Let G be the grammar

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

For string aaabbabbba find

(i) Left most derivation (ii) Right most derivation

(iii) Parse Tree

(iv) Is the grammar unambiguous ?

Solution : It will be worth while to draw the parse tree and from the parse tree, one can easily write left most and right most derivation.

(iii) Parse tree

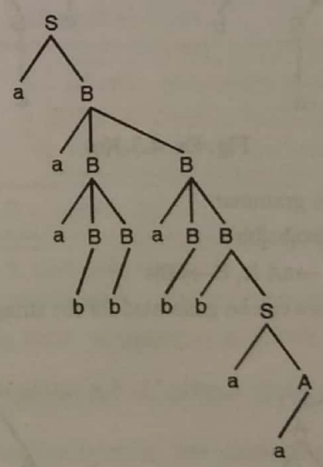


Fig. Ex. 4.3.8

(i) Left most derivation

$S \rightarrow aB \rightarrow aaBB \rightarrow aaaBBB \rightarrow aaabBB$
$\rightarrow aaabbB \rightarrow aaabbaBB \rightarrow aaabbabB \rightarrow aaabbabbS$
$\rightarrow aaabbabbbA \rightarrow aaabbabbba$

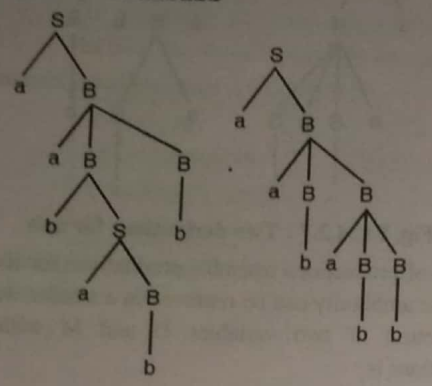
(ii) Right most derivation

$S \rightarrow aB \rightarrow aaBB \rightarrow aaBaBB \rightarrow aaBaBbS \rightarrow aaBaBbbA$
$\rightarrow aaBaBbba$

Scanned by CamScanner

→aaBabbba→aaaBBabbba→aaaBbabbba→aaabbabbba

(iv) The grammar is ambiguous as we can draw two parse trees for aababb



(a) (b)

Fig. Ex. 4.3.8

Example 4.3.9

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.

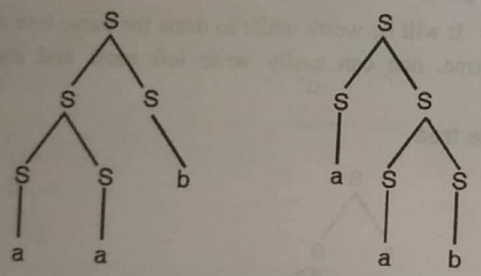


Fig. Ex. 4.3.9(a)

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

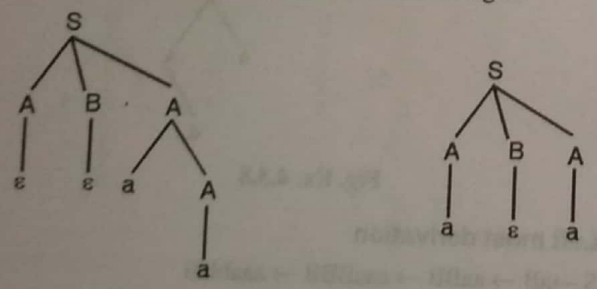


Fig. Ex. 4.3.9(b)

Unambiguous grammar :

S → as|bXε

[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

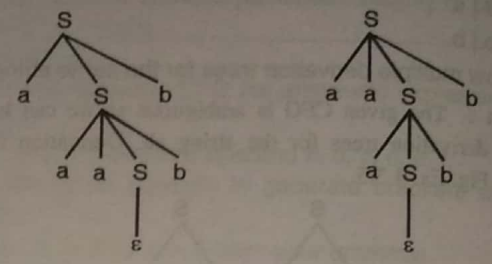


Fig. Ex. 4.3.9(c)

Unambiguous grammar

Given language is of the form

$$L = \{a^i b^j \mid i \geq j \text{ and } i, j \geq 0\}$$

$$S \rightarrow aSb \mid X$$

$$X \rightarrow aX \mid \varepsilon$$

Example 4.3.10

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 :

$$E \rightarrow F - E \mid E - F \mid F$$

$$F \rightarrow a \mid b$$

(a) The grammar can be shown to be an ambiguous grammar by drawing two different derivation trees for the string a – b.

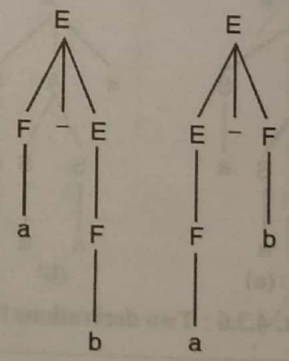


Fig. Ex. 4.3.10 : Two derivations for a – b

(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 production E → F – E. The resulting unambiguous grammar is :

$$E \rightarrow E - F \mid F$$

$$F \rightarrow a \mid b$$

Scanned by CamScanner

Example 4.3.11

Test whether the following grammars are ambiguous.

1. $S \to 0S1S \mid 1S0S \mid \varepsilon$
2. $S \to AA$
   $A \to aAb \mid bAa \mid \varepsilon$

Solution :

1. $S \to 0S1S \mid 1S0S \mid \varepsilon$

The given grammar is ambiguous as we can have two different parse trees for the string 0101.

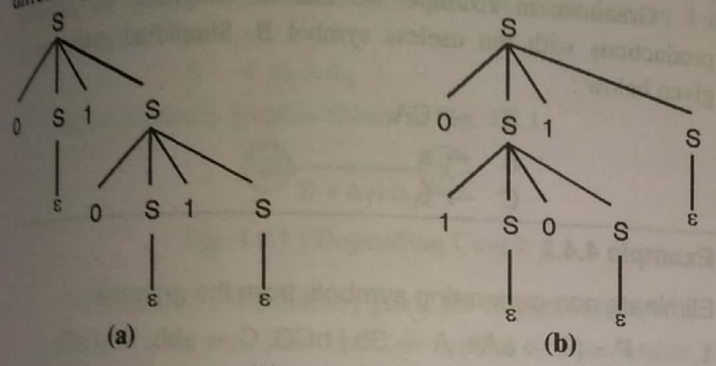


Fig. Ex. 4.3.11

2. $S \to AA$
   $A \to aAb \mid bAa \mid \varepsilon$

The given grammar is ambiguous. We can derive two different parse trees for the string abab.

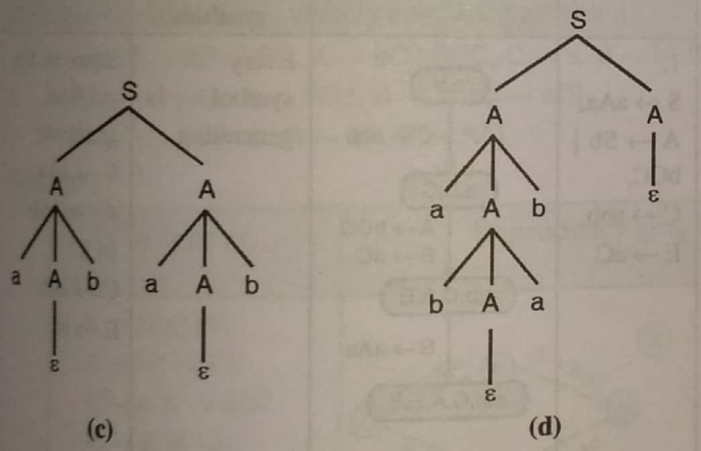


Fig. Ex. 4.3.11

Example 4.3.12

If G is the grammar $S \to SbS|a$, show that G is ambiguous.

Solution : The string ababa has two derivations. This shows that the grammar is ambiguous. The two derivations for the string ababa are given below.

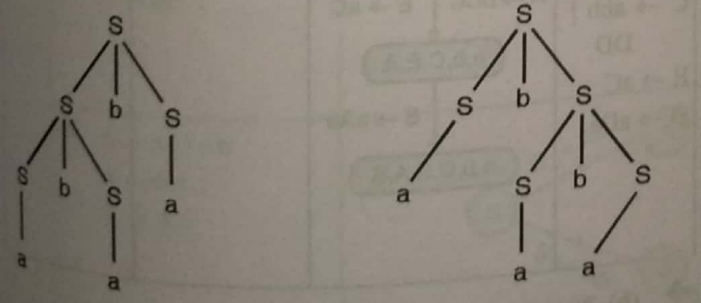


Fig. Ex. 4.3.12

Example 4.3.13

Obtain the unambiguous grammar for the following grammar

$E \to E + E \mid E - E$
$E \to E * E \mid E / E$
$E \to (E) \mid I$
$I \to a \mid b \mid c$

Solution : There are two problems in the given grammar :

1. All the operators are at the same precedence level.
2. An expression is both left and right associative.

These problems can be removed by introducing more variables, each representing expressions that share a level of "binding strength".

An unambiguous grammar is given below :

$$E \to E + T \mid E - T \mid T$$
$$T \to T * F \mid T/F \mid F$$
$$F \to (E) \mid I$$
$$I \to a \mid b \mid c$$

Syllabus Topic : Simplification

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 :

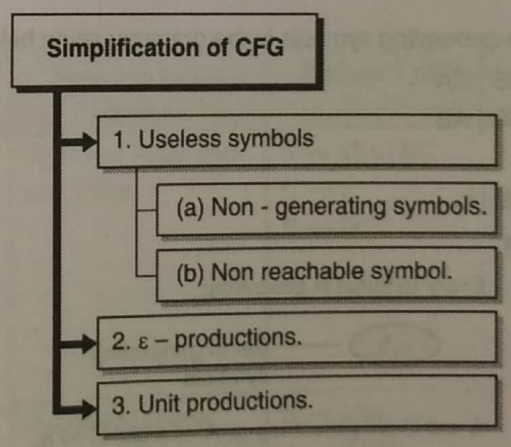


Fig. C4.2 : Simplification of CFG

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 :

(a) Non - generating symbols. (b) Non reachable symbol.

➜ **(a) Non-generating Symbols**

A symbol $X \in V$ (Set of variables) is a generating symbol if

$X \overset{*}{\underset{G}{\Rightarrow}} w$,

Scanned by CamScanner

where, $w \in T^*$ i.e. every variable must generate a string of terminals.

Example : Grammar

$$\left.\begin{array}{l} S \to Aa \mid Bb \mid a \mid b \\ A \to Aa \mid a \\ B \to bB \end{array}\right| \quad \text{...Grammar (4.14)}$$

Requires simplification as the symbol B is non-generating. Only production for B is

$$B \to bB$$

and it can not generate a string of terminals. The Grammar (4.14) can be simplified by deleting every production containing the useless symbol B. A simplified grammar is given in (4.15).

$$\left.\begin{array}{l} S \to Aa \mid a \mid b \\ A \to Aa \mid a \end{array}\right| \quad \text{...Grammar (4.15)}$$

A grammar containing a non-generating symbol V_i should be simplified by deleting every production containing the non-generating symbol V_i.

☞ Finding non-generating symbols

There are two rules for finding a set of generating symbols for the given grammar.

1. Every symbol in T (terminal) is generating.
2. If there is a production $A \to \alpha$ 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

$S \to AB \mid CA$

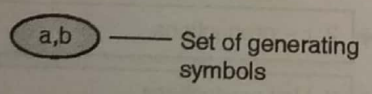
$B \to BC \mid AB$

$A \to a$

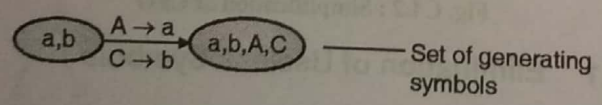
$C \to aB \mid b.$

Solution :

Step 1 : Every terminal is generating

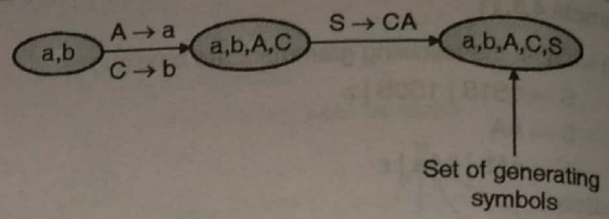


Step 2 : A and C are generating as $A \to a$ and $C \to b$.



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 $A \to \alpha$ and every symbol in α is generating then A is generating.

Step 3 : Symbol **S** is generating as there is a production $S \to CA$ 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 :

$S \to CA$

$A \to a$

$C \to b$

Example 4.4.2

Eliminate non-generating symbols from the grammar.

1. $P = \{S \to aAa, A \to Sb \mid bCC, C \to abb, E \to aC\}$
2. $P = \{S \to aAa, A \to Sb \mid bCC \mid DaA, C \to abb \mid DD, E \to aC, D \to aDA\}$

Solution :

Grammar	Generating symbols	Non-generating symbols	Simplified grammar
1. $S \to aAa$, $A \to Sb \mid bCC$, $C \to abb$, $E \to aC$	a,b → (C → abb) a,b,C → (A → bCC, E → aC) a,b,C,A,E → (S → aAa) a,b,C,A,E,S	Every symbol is generating	Same as the original grammar $S \to aAa$, $A \to Sb \mid bCC$ $C \to abb$ $E \to aC$
2. $S \to aAa$ $A \to Sb \mid bCC \mid DaA$ $C \to abb \mid DD$ $E \to aC$ $D \to aDa$	a,b → (C → abb) a,b,C → (A → bCC, E → aC) a,b,C,E,A → (S → aAa) a,b,C,E,A,S	Symbol D is non-generating	$S \to aAa$ $A \to Sb \mid bCC$ $C \to abb$ $E \to aC$

➜ (b) Non-reachable Symbols

A symbol X is reachable if it can be reached from the start symbol S.

Scanned by CamScanner

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 shown in Fig. 4.4.1.

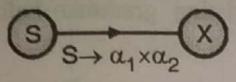


Fig. 4.4.1 : Depending Graph

- We must draw a dependency graph for all productions.
- If there is no path from the start symbol S to a variable X, then X is non-reachable.

Example 4.4.3

Eliminate non-reachable symbols from the given grammar.

$P = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb, E \rightarrow aC\}$

Solution :

Step 1 : Drawing of dependency graph :

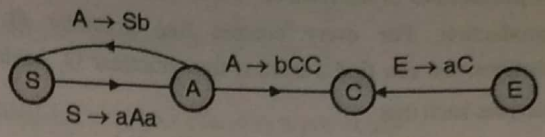


Fig. Ex. 4.4.3 : Dependency Graph

- From the production $S \rightarrow aAa$, A is dependent on S. It is shown using a directed edge from S to A.
- From the production $A \rightarrow bCC$, C is dependent on A. From the production $A \rightarrow Sb$, S is dependent of A.
- From the production $E \rightarrow aC$, C is dependent on E.

There is no path from S to E, E is non-reachable.

Step 2 : Simplification of grammar :

Grammar can be simplified by deleting every production containing the non-reachable symbol E. Simplified grammar is given below :

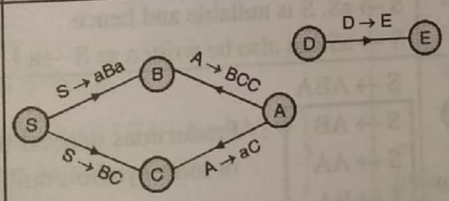
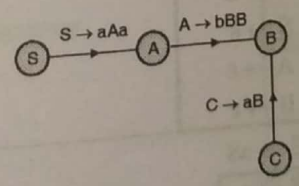
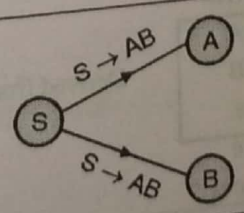
$$P_1 = \{S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb\}$$

Example 4.4.4

Eliminate non-reachable symbols from the grammar :

1. $P = \{S \rightarrow aBa \mid BC, A \rightarrow aC \mid BCC, C \rightarrow a, B \rightarrow bCC, D \rightarrow E, E \rightarrow d\}$
2. $P = \{S \rightarrow aAa, A \rightarrow bBB, B \rightarrow ab, C \rightarrow aB\}$
3. $P = \{S \rightarrow aS \mid AB, A \rightarrow bA, B \rightarrow AA\}$

Solution :

Grammar	Dependency graph	Non-reachable symbols	Reduced grammar
1. $S \rightarrow aBa \mid BC$ $A \rightarrow aC \mid BCC$ $C \rightarrow a, B \rightarrow bCC$ $D \rightarrow E, E \rightarrow d$		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

Scanned by CamScanner

4.4.2 Elimination of ε-productions

A productions of the form $A \to \varepsilon$, 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 $X \to \varepsilon$.
2) If there is a production of the form $X \to \alpha$ and all symbols of α are nullable.

Example : In the grammar with productions $S \to ABA$, $A \to aA \mid \varepsilon$, $B \to bB \mid \varepsilon$

A is nullable as there is a production $A \to \varepsilon$

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.

∴ The set of nullable symbols = {S, A, B}

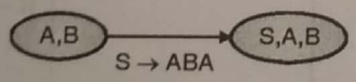
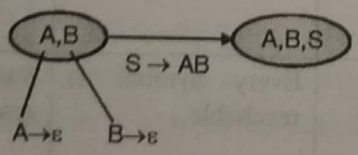
Step 2 : Addition of productions with nullable variables removed.

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. ∴ 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}	$S \to aS$ $S \to a$ $S \to AB$ Derived from $S \to As$ $S \to A$ $S \to B$ $S \to \varepsilon$ Derived from $S \to AB$ $A \to \varepsilon$ $B \to \varepsilon$

Scanned by CamScanner

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

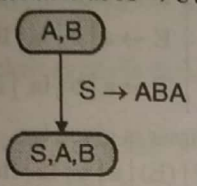
Remove ε-productions from the given grammar.

$S \to ABA,\ A \to aA \mid \varepsilon,\ B \to bB \mid \varepsilon$

Solution

Step 1 : Find nullable variables

A and B are nullable as $A \to \varepsilon$ and $B \to \varepsilon$.



S is nullable as A and B are nullable and there is a production $S \to ABA$, with every symbol on the right as nullable.

∴ Nullable set of symbols = {A, B, S}

Step 2 : Addition of productions with nullable variables removed.

$S \to ABA$ can be written as

$A \to aA$ can be written as $\left\{\begin{array}{l} A \to aA \\ A \to a \end{array}\right\}$

$B \to bB$ can be written as $\left\{\begin{array}{l} B \to bB \\ B \to b \end{array}\right\}$

∴ Equivalent set of productions.

$$P_1 = \left\{\begin{array}{l} S \to ABA \mid AB \mid BA \mid AA \mid A \mid B \mid \varepsilon \\ A \to aA \mid a \mid \varepsilon \\ B \to bB \mid b \mid \varepsilon \end{array}\right\}$$

Step 3 : Remove ε – productions

The final set of productions is given by

$$P_2 = \left\{\begin{array}{l} S \to ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A \to aA \mid a \\ B \to bB \mid b \end{array}\right\}$$

Example 4.4.6

Remove ε-productions from the given grammar.

$S \to AB$

$A \to aAA \mid \varepsilon$

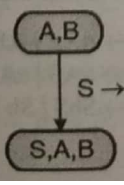
$B \to bBB \mid \varepsilon$

Solution :

Step 1 : Find nullable variables

A and B are nullable as $A \to \varepsilon$ and $B \to \varepsilon$.

S is nullable as A and B are nullable and $S \to AB$.



∴ Nullable set of symbols = {S, A, B}

Step 2 : Addition of production with nullable variable removed.

$S \to AB$ can be written as $\left\{\begin{array}{l} S \to AB \\ S \to A \\ S \to B \\ S \to \varepsilon \end{array}\right\}$

By making one or more nullable symbols as null.

$A \to aAA$ can be written as $\left\{\begin{array}{l} A \to aAA \\ A \to aA \\ A \to a \end{array}\right\}$

By making one or more nullable symbols as null.

$B \to bBB$ can be written as $\left\{\begin{array}{l} B \to bBB \\ B \to bB \\ B \to b \end{array}\right\}$

By making one or more nullable symbols as null.

Step 3 : Remove ε-productions

∴ Equivalent set of productions.

$$P_1 = \left\{\begin{array}{l} S \to AB \mid A \mid B \\ A \to aAA \mid aA \mid a \\ B \to bBB \mid bB \mid b \end{array}\right\}$$

4.4.3 Elimination of Unit Productions

A production of the form $A \to B$

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

  $A_1 \to A_2,\ A_2 \to A_3,\ A_3 \to A_4$ and $A_4 \to A_1$

- 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

$A_1 \to A_2 \ldots A_j \to \alpha$, where $A_j \to \alpha$ is a non unit production.

Each variable A_i to A_j will derive α.

Example 4.4.7

Eliminate unit productions from

$$P = \left\{\begin{array}{l} S \to ABA \mid BA \mid AA \mid AB \mid A \mid B \\ A \to aA \mid a \\ B \to bB \mid b \end{array}\right\}$$

Scanned by CamScanner

Solution : Let G_1 with productions P_1 is an equivalent grammar without unit productions.

Step 1 : Add all non-unit productions of the given grammar to P_1.

$$P_1 = \left\{ \begin{array}{l} S \to ABA \mid BA \mid AA \mid AB \\ A \to aA \mid a \\ B \to bB \mid b \end{array} \right\}$$

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 production $S \to A$]
2. (S, B) [Due to unit production $S \to B$]

Step 3 : Unit production $S \to A$ can be removed by expanding A.

A derives aA | a.

Unit production $S \to B$ can be removed by expanding $S \to B$. B derives bB | b.

Set of final productions for G_1 are given below :

	Productions
From step 1	$S \to ABA \mid BA \mid AA \mid AB$
	$A \to aA \mid a$
	$B \to bB \mid b$
Pair (S, A)	$S \to aA \mid a$
Pair (S, B)	$S \to bB \mid b$

Thus,

$$P_1 = \left\{ \begin{array}{l} S \to ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bB \mid b \\ A \to aA \mid a \\ B \to bB \mid b \end{array} \right\}$$

Example 4.4.8

Eliminate unit productions from the grammar

$$P = \left\{ \begin{array}{l} E \to E+T \mid T,\ T \to T*F \mid F, \\ F \to (E) \mid I \\ I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array} \right\}$$

Solution : Let the grammar G_1, with productions P_1 is an equivalent grammar without unit productions.

Step 1 : Add all non-unit productions of the given grammar to P_1

$$P_1 = \left\{ \begin{array}{l} E \to E+T \\ T \to T*F \\ F \to (E) \\ I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array} \right\}$$

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 $E \to T$]
2. (T, F) [Due to unit production $T \to F$]
3. (F, I) [Due to unit production $F \to I$]
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 :

	Productions
From step 1	$E \to E+T$
	$T \to T*F$
	$F \to (E)$
	$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
Pair (E, T)	$E \to T*F$
Pair (T, F)	$T \to (E)$
Pair (F, I)	$F \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
Pair (E, F)	$E \to (E)$
Pair (E, I)	$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
Pair (T, I)	$T \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Thus, the final grammar is given by

$$P_1 = \left\{ \begin{array}{l} E \to E+T \mid T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ T \to T*F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F \to (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{array} \right\}$$

Example 4.4.9 **MU - May 2017, 10 Marks**

Simplify the following grammar

$S \to ASB \mid \varepsilon$

$A \to aAS \mid a$

$B \to SbS \mid A \mid bb$

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 $S \to \varepsilon$

Grammar with ε production (G)	Grammar after taking effect of nullable symbols (G_1)
$S \to ASB$	$S \to ASB$
	$S \to AB$] By making symbol S as null.
$S \to \varepsilon$	$S \to \varepsilon$
$A \to aAS$	$A \to aAS$
	$A \to aA$] By making symbol S as null
$A \to a$	$A \to a$
$B \to SbS$	$B \to SbS$
	$B \to Sb$] By making one or more S null
	$B \to bS$
	$B \to b$
$B \to A$	$B \to A$
$B \to bb$	$B \to bb$

Final G_1, after deletion of ε-productions is given below :

$$G_1 = \left\{ \begin{array}{l} S \to ASB \mid AB \\ A \to aAS \mid aA \mid a \\ B \to SbS \mid Sb \mid bS \mid b \mid A \mid bb \end{array} \right\}$$

Scanned by CamScanner

Step 2 : Eliminate unit productions from G_1 and obtain G_2.

The grammar G_1 contains a unit production $B \to A$.

A derives $aAS \mid aA \mid b$

$\therefore$ $\boxed{B \to A}$ $\to$ can be written as $\to$ $\boxed{B \to aAS \mid aA \mid b}$

G_2, after elimination of unit productions from G_1 is given by :

$$G_2 = \left\{ \begin{array}{l} S \to ASB \mid AB \\ A \to aAS \mid aA \mid a \\ B \to SbS \mid Sb \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \end{array} \right\}$$

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.

$$\therefore G_3 = G_2 = \left\{ \begin{array}{l} S \to ASB \mid AB \\ A \to aAS \mid aA \mid a \\ B \to SbS \mid Sb \mid bS \mid b \mid bb \mid aAS \mid aA \mid b \mid a \end{array} \right\}$$

Example 4.4.10

Simplify the following grammar :

$S \to 0A0 \mid 1B1 \mid BB$

$A \to C$

$B \to S \mid A$

$C \to S \mid \varepsilon$

Solution :

Step 1 : Eliminate ε-production from G (given grammar) and obtain G_1. Set of nullable symbols are as shown in Fig. Ex. 4.4.10 :

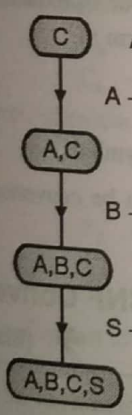


Fig. Ex. 4.4.10

The grammar after removal of null production is given below :

$$\left\{ \begin{array}{l} S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid B \mid BB \\ A \to C,\ B \to S \mid A,\ C \to S \end{array} \right\} \text{ – grammar } G_1$$

Step 2 : Following unit productions are there in G_1.

$S \to B,\ A \to C,\ B \to S,\ B \to A,\ C \to S$

There is a chain.

$B \to A \to C \to S \to B$

and $S \to B \to S$

These chains are cyclic in nature of each is terminating in S.

The grammar without unit productions is given below :

$$\left\{ \begin{array}{l} S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ A \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ B \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \\ C \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \end{array} \right\}$$

Step 3 : The symbol C is not reachable and hence it can be deleted. The set of final productions is given below :

$S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

$A \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

$B \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$

Example 4.4.11

Simplify the following grammar

$S \to Ab$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

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.

Step2.1 : Add all non unit productions of the given grammar G_1 to productions P_2 of G_2.

$$P_2 = \left\{ \begin{array}{l} S \to Ab,\ A \to a \\ E \to a \end{array} \right\}$$

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 → C]
2.	(C, D)	[Due to unit production C → D]
3.	(D, E)	[Due to unit production D → 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 $S \to Ab,\ A \to a,\ E \to a$

Pair (B, C)	B → a	[there is a chain B → C → D → E → a]
Pair (C, D)	C → a	[there is a chain B → C → D → E → a]
Pair (D, E)	D → a	[there is a chain B → C → D → E → a]
Pair (B, D)	B → a	[there is a chain B → C → D → E → a]

Scanned by CamScanner

Pair (B, E)	$B \to a$	[there is a chain $B \to C \to D \to E \to a$]
Pair (C, E)	$C \to a$	[there is a chain $B \to C \to D \to E \to a$]

Thus the productions after elimination of unit productions is

$$P_2 = \left\{ \begin{array}{l} S \to Ab \\ A \to a \\ E \to a \\ B \to a \\ C \to a \\ D \to a \end{array} \right\}$$

Step 3 : Elimination of useless symbols :

- Every symbol, in the given grammar is generating.
- Reachable symbols can be located by drawing a dependency graph.

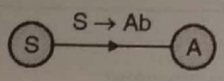


Fig. Ex. 4.4.11 : Dependency graph

Two symbols {S, A} are reachable. Other symbols {B, C, D, E} are non-reachable.

Final grammar G_3 with productions P_3 is obtained by deleting useless symbols.

$$P_3 = \left\{ \begin{array}{l} S \to Ab \\ A \to a \end{array} \right\}$$

Example 4.4.12

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 shown in Fig. Ex. 4.4.12 :

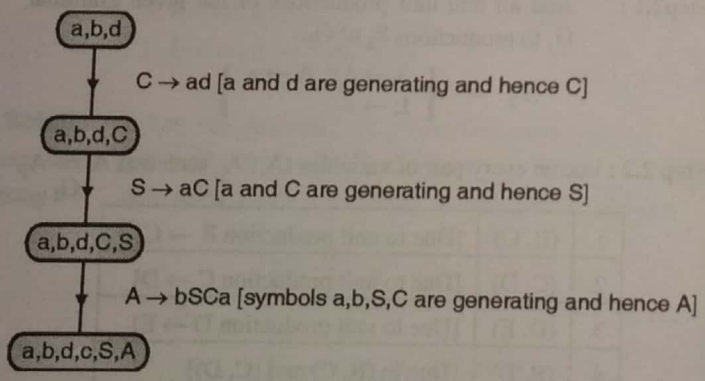


Fig. Ex. 4.4.12

$\therefore$ Non generating symbols = {B}

Thus the productions after elimination of non-generating symbols is :

$$P_1 = \left\{ \begin{array}{l} S \to aC \\ A \to bSCa \\ C \to ad \end{array} \right\}$$

The set of reachable symbols can be found by drawing a dependency graph.

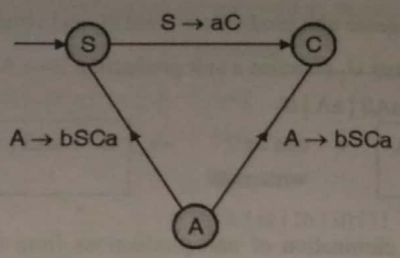


Fig. Ex. 4.4.12(a) : Dependency graph

The symbol {A} is non-reachable.

Thus the final set of productions after elimination of non-reachable symbols is :

$$P_2 = \left\{ \begin{array}{l} S \to aC \\ C \to ad \end{array} \right\}$$

Syllabus Topic : Normalization

4.5 Normal Forms for CFG

➔ **(Dec. 2016)**

Q. Define Normal form and its types.
MU - Dec. 2016, 5 Marks

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 : CNF

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(a) Algorithm for CFG to CNF Conversion

➔ **(May 2014)**

Q. Explain algorithm for the conversion of a Context Free Grammar (CFG) to Chomsky Normal Form (CNF).
MU - May 2014, 5 Marks

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

Scanned by CamScanner

$A \rightarrow V_1V_2aV_3bV_4$ as

$A \rightarrow V_1V_2C_aV_3C_bV_4$

And adding two productions

$C_a \rightarrow a$

and $C_b \rightarrow b$

3. Every production deriving 3 or more variables ($A \rightarrow \alpha$ with $|\alpha| \geq 3$) can be broken down into a cascade of productions with each deriving a string of two variables.

Example : Consider a production $A \rightarrow X_1X_2...X_n$ where $n \geq 3$ and as X_i's are variables.

The production $A \rightarrow X_1X_2... X_n$ should be broken down as given below :

$$A \rightarrow X_1C_1$$
$$C_1 \rightarrow X_2C_2$$
$$C_2 \rightarrow X_3C_3$$
$$\vdots$$
$$C_{n-2} \rightarrow X_{n-1}X_n$$

Each with two variables on the right.

Example 4.5.1 **MU - Dec. 2014, 5 Marks**

Find the CNF equivalent to

$S \rightarrow aAbB, A \rightarrow aA \mid a, B \rightarrow bB \mid b.$

Solution :

Step 1 : Simplification of grammar.

The grammar is already in a simple form without

1. ε-productions.
2. Unit-productions.
3. Useless symbols.

Step 2 : Every symbol in α, in a production of the form $A \rightarrow \alpha$ where $|\alpha| \geq 2$ should be a variable :

This can be done by adding two productions.

$C_a \rightarrow a$

and $C_b \rightarrow b$

The set of productions after the above changes is :

$$P_2 = \left\{ \begin{array}{l} S \rightarrow C_aAC_bB \\ A \rightarrow C_aA \mid a \\ B \rightarrow C_bB \mid b \\ C_a \rightarrow a \\ C_b \rightarrow b \end{array} \right\}$$

Step 3 : Convert to CNF

$S \rightarrow C_aC_1$
$C_1 \rightarrow AC_2$
$C_2 \rightarrow C_bB$
$A \rightarrow C_aA \mid a$
$B \rightarrow C_bB \mid b$
$C_a \rightarrow a$
$C_b \rightarrow b$

Corresponding to $S \rightarrow C_aAC_bB$

$$\therefore P_3 = \left\{ \begin{array}{l} S \rightarrow C_aC_1, C_1 \rightarrow AC_2, C_2 \rightarrow C_bB \\ A \rightarrow C_aA \mid a, B \rightarrow C_bB \mid b, C_a \rightarrow a \\ C_b \rightarrow b \end{array} \right\}$$

Example 4.5.2

Convert the grammar given below to its equivalent CNF :

$S \rightarrow PQP$
$P \rightarrow 0P \mid \varepsilon$
$Q \rightarrow 1Q \mid \varepsilon$

Solution :

Step 1 : Simplify the grammar

Step 1.1 : Eliminate ε productions Nullable symbols = {P, Q, S}

P and Q are nullable as $P \rightarrow \varepsilon$ and $Q \rightarrow \varepsilon$. S is also nullable as $S \rightarrow PQP$ with every symbol on the right as nullable.

Eliminating ε-productions, we get a set of productions P_1

$$P_1 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid P \mid Q \\ P \rightarrow 0P \mid 0 \\ Q \rightarrow 1Q \mid 1 \end{array} \right\}$$

Step 1.2 : Eliminate unit productions :

There are two unit productions $S \rightarrow P$ and $S \rightarrow Q$.

Eliminating unit productions from P_1, we get a set of productions P_2

$$P_2 = \left\{ \begin{array}{l} S \rightarrow PQP \mid PQ \mid QP \mid 0P \mid 0 \mid 1Q \mid 1 \mid PP \\ P \rightarrow 0P \mid 0 \\ Q \rightarrow 1Q \mid 1 \end{array} \right\}$$

Step 2 : Every symbol in α, in a production of the form $A \rightarrow \alpha$ with $|\alpha| \geq 2$ should be a variable.

This can be done by adding two productions

$C_0 \rightarrow 0$
$C_1 \rightarrow 1$

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
$S \rightarrow PQP$	$S \rightarrow PC_2$ $C_2 \rightarrow QP$
$S \rightarrow PQ$	$S \rightarrow PQ$
$S \rightarrow QP$	$S \rightarrow QP$
$S \rightarrow PP$	$S \rightarrow PP$
$S \rightarrow C_0P$	$S \rightarrow C_0P$
$S \rightarrow 0$	$S \rightarrow 0$
$S \rightarrow C_1Q$	$S \rightarrow C_1Q$
$S \rightarrow 1$	$S \rightarrow 1$
$P \rightarrow C_0P \mid 0$	$P \rightarrow C_0P \mid 0$
$Q \rightarrow C_1Q \mid 1$	$Q \rightarrow C_1Q \mid 1$
$C_0 \rightarrow 0$	$C_0 \rightarrow 0$
$C_1 \rightarrow 1$	$C_1 \rightarrow 1$

Scanned by CamScanner

∴ The set of final productions in CNF is :

$$P_3 = \left\{ \begin{array}{l} S \to PC_2 \mid PQ \mid QP \mid PP \mid C_0P \mid 0 \mid C_1Q \mid 1 \\ C_2 \to QP \\ P \to C_0P \mid 0 \\ Q \to C_1Q \mid 1 \\ C_0 \to 0 \\ C_1 \to 1 \end{array} \right\}$$

Example 4.5.3 **MU - May 2014, 5 Marks**

Check whether the given grammar is in CNF

$S \to bA \mid aB$

$A \to bAA \mid aS \mid a$

$B \to aBB \mid bS \mid b$

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.

$$S \to bA \mid aB$$
$$A \to bAA \mid aS$$
$$B \to aBB \mid bS$$

Finding an equivalent CNF

Step 1 : Simplification of grammar – already simplified

Step 2 : Every symbol in α, in production of the form $A \to \alpha$ 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

$$\left\{ \begin{array}{l} S \to C_bA \mid C_aB \\ A \to C_bAA \mid C_aS \mid a \\ B \to C_bBB \mid C_bS \mid b \\ C_a \to a, C_b \to b \end{array} \right\}$$

Step 3 : Finding an equivalent CNF :

Original production	Equivalent productions in CNF
$S \to C_bA \mid C_aB$	$S \to C_bA \mid C_aB$
$A \to C_bAA$	$A \to C_bC_1$ $C_1 \to AA$
$A \to C_aS \mid a$	$A \to C_aS \mid a$
$B \to C_bBB$	$B \to C_bC_2$ $C_2 \to BB$
$B \to C_bS \mid b$	$B \to C_bS \mid b$
$C_a \to a$	$C_a \to a$
$C_b \to b$	$C_b \to b$

∴ The set of final productions in CNF is :

$$\left\{ \begin{array}{l} S \to C_bA \mid C_aB \\ A \to C_bC_1 \mid C_aS \mid a \\ C_1 \to AA \\ B \to C_bC_2 \mid C_bS \mid b \\ C_2 \to BB \\ C_a \to a \\ C_b \to b \end{array} \right\}$$

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

$$V = \{\{S\}, \{(,)\}, \{S \to (S), S \to SS, S \to \varepsilon\}, S\}$$

where the set of productions

$$P = \left\{ \begin{array}{l} S \to (S) \\ S \to SS \\ S \to \varepsilon \end{array} \right\}$$

An equivalent set of productions without ε-productions can be written as

$$P_1 = \left\{ \begin{array}{l} S \to (S) \mid (\,) \\ S \to SS \end{array} \right\}$$

Conversion of P_1 in CNF can be done in two steps.

Step 1 : Addition of two productions

$\boxed{C_1 \to (}$ and $\boxed{C_2 \to)}$ will change the set of productions to :

$$P_2 = \left\{ \begin{array}{l} S \to C_1SC_2 \mid C_1C_2 \\ S \to SS, C_1 \to (, C_2 \to) \end{array} \right\}$$

Step 2 : Finding an equivalent CNF :

Production in CFG	Equivalent productions in CNF
$S \to C_1SC_2$	$S \to C_1C_3$ $C_3 \to SC_2$
$S \to C_1C_2$	$S \to C_1C_2$
$S \to SS$	$S \to SS$
$C_1 \to ($	$C_1 \to ($
$C_2 \to)$	$C_2 \to)$

∴ The set of final productions in CNF is

$$\left\{ \begin{array}{l} S \to C_1C_3 \mid C_1C_2 \mid SS \\ S \to SC_2 \\ C_1 \to (\\ C_2 \to) \end{array} \right\}$$

Example 4.5.5

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 $A \to \alpha$ with $|\alpha| \geq 2$ should be a variable :

This can be done by adding two productions

$$Ca \to a$$

and $$Cb \to b$$

The set of productions after above changes is :

$$S \to AC_bC_a \mid C_aC_aC_b$$

Step 3 : Convert to CNF

$$S \to AC_1$$

Scanned by CamScanner

$C_1 \rightarrow C_bC_a$
$S \rightarrow C_aC_2$
$C_2 \rightarrow C_aC_b$

Example 4.5.6

Convert the following grammar to CNF

$S \rightarrow AACD$
$A \rightarrow aAb \mid \varepsilon$
$C \rightarrow aC \mid a$
$A \rightarrow aDa \mid bDb \mid \varepsilon$

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 :

$S \rightarrow AACD|ACD|CD$
$A \rightarrow aAb|ab$
$C \rightarrow aC|a$
$A \rightarrow aDa|bDb$

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**.

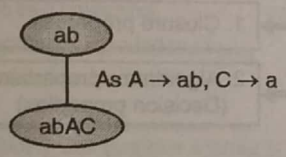


Fig. Ex. 4.5.6

Example 4.5.7

Convert the following grammar to Chomsky normal form (CNF).

G = ({S}, {a}, P, {S})
P = {S → a a a a a S, S → a a a}

Solution :

Step 1 : Re-writing of productions after addition of the new production $C_1 \rightarrow a$

$S \rightarrow C_1C_1C_1C_1C_1S$
$S \rightarrow C_1C_1C_1$
$C_1 \rightarrow a$

	Original production	Equivalent productions in CNF
1.	$S \rightarrow C_1C_1C_1C_1C_1S$	$S \rightarrow C_1C_2$
		$C_2 \rightarrow C_1C_3$
		$C_3 \rightarrow C_1C_4$
		$C_4 \rightarrow C_1C_5$
		$C_5 \rightarrow C_1S$
2.	$S \rightarrow C_1C_1C_1$	$S \rightarrow C_1C_6$
		$C_6 \rightarrow C_1C_1$
3.	$C_1 \rightarrow a$	$C_1 \rightarrow a$

Example 4.5.8

Convert the following CFG to CNF

S → aSa | bSb | a | b | aa | bb

Solution :

Step 1 : Substituting 'A' for 'a' and 'B' for 'b', we get,

$S \rightarrow ASA$
$S \rightarrow BSB$
$S \rightarrow a$
$S \rightarrow b$
$S \rightarrow AA$
$S \rightarrow BB$
$A \rightarrow a$
$B \rightarrow b$

Step 2 : Converting each production in CNF.

	Production (original)	Productions in CNF
1.	$S \rightarrow ASA$	$S \rightarrow AA_1, A_1 \rightarrow SA$
2.	$S \rightarrow BSB$	$S \rightarrow BA_2, A_2 \rightarrow SB$

Other productions are already in CNF.

Example 4.5.9 MU - May 2017, 5 Marks

Convert the CFG to CNF

S → aSa | bSb | a | b

Solution :

Step 1 : Substituting 'A' for 'a' and 'B' for 'b', we get

$S \rightarrow ASA$
$S \rightarrow BSB$
$S \rightarrow a$
$S \rightarrow b$
$A \rightarrow a$
$B \rightarrow b$

Step 2 : converting each production for CNF

	Production (original)	Productions in CNF
1.	$S \rightarrow ASA$	$S \rightarrow AA_1, A_1 \rightarrow SA$
2.	$S \rightarrow BSB$	$S \rightarrow BA_2, A_2 \rightarrow SB$

Other production are already in CNF.

Syllabus Topic : Chomsky Hierarchy (Types 0 to 3)

4.6 Chomsky Classification for Grammar

➔ **(May 2014, Dec. 2014, May 2017)**

Q. Write short note on : Chomsky Hierarchy.
MU- May 2014, 5 Marks

Q. Explain Chomsky hierarchy.
MU - Dec. 2014, May 2017, 10 Marks

Scanned by CamScanner

– A grammar can be classified on the basis of production rules. Chomsky classified grammars into the following types :

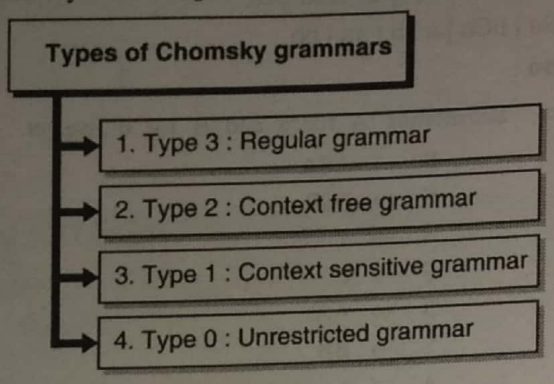


Fig. C4.3 : Types of Chomsky grammars

➜ 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 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.

$S \overset{*}{\Rightarrow} a^{n-1} S(BC)^{n-1}$ [by applying $S \rightarrow aSBC$, (n – 1) times]

$\Rightarrow a^{n-1} aBC(BC)^{n-1}$ [by applying $S \rightarrow aBC$]

$\Rightarrow a^{n-1} abC(BC)^{n-1}$ [by applying $aB \rightarrow ab$]

$\Rightarrow a^{n-1} abB^{n-1} C^n$ [by applying $CB \rightarrow BC$, n – 1 times]

$\Rightarrow a^n b^{n-1} bC^n$ [by applying $bB \rightarrow bb$ several times]

$\Rightarrow a^n b^{n-1} bcC^{n-1}$ [by applying $bC \rightarrow bc$]

$\Rightarrow a^n b^n c^n$ [by applying $cC \rightarrow cc$ several times]

➜ 4.6.4 Type 0 or Unrestricted Grammar

Productions can be written without any restriction in a unrestricted grammar. If there is production of the $\alpha \rightarrow \beta$, then length of α could be more than length of β.

– Every grammar also is a type 0 grammar.
– A type 2 grammar is also a type 1 grammar
– A type 3 grammar is also a type 2 grammar.

4.7 Properties of Context-Free Languages

In this section, we will consider some general properties of CFL. These properties can be classified in two groups

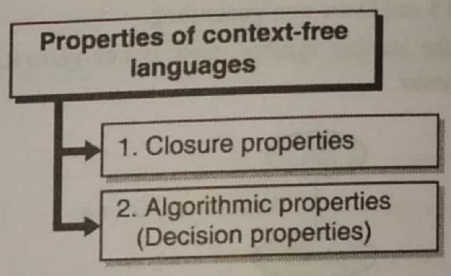


Fig. C4.4 : Properties of Context-Free Languages

➜ 4.7.1 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.

4.7.1(a)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 $G_2 = (V_2, T_2, P_2, S_2)$

Scanned by CamScanner

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.

$$S \to S_1$$
$$S \to S_2$$

The grammar G can be written as :

$$G = (V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup \{ S \to S_1 \mid S_2 \}, S)$$

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.

$$\therefore \quad L(G) = L_1 \cup L_2$$

4.7.1(b) CFL is Closed under Concatenation

Theorem

If L1 and L2 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.

$$S \to S_1S_2$$

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 :

$$G = (V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup \{ S \to S_1S_2 \}, S)$$

4.7.1(c) 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.

$$S \to SS_1$$
$$S \to \varepsilon$$

The production $S \to SS_1 \mid \varepsilon$ will generate a string w* where $w \in L_1$.

The grammar G can be written as :

$$G = (V_1, T_1, P_1 \cup \{ S \to SS_1 \mid \varepsilon \}, S)$$

4.7.1(d) 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 :

$$S \to AB$$
$$A \to aAb \mid \varepsilon$$
$$B \to cB \mid \varepsilon$$

The language L_2 is a CFL with set of productions given below :

$$S \to AB$$
$$A \to aA \mid \varepsilon$$
$$B \to bBc \mid \varepsilon$$

- 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, $L_1 \cap L_2 = \{ a^nb^nc^n \mid n \geq 0 \}$. From pumping lemma for CFL, a string of the form $a^nb^nc^n$ can not be generated by a CFG.

Therefore, the class of context-free languages is not closed under intersection.

4.7.1(e) 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.

4.7.1(f) Intersection of CFL and RL

Theorem

If L is a CFL and R is a regular language, then $R \cap L$ is a CFL.

Scanned by CamScanner

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 $(q_2, u) \overset{*}{\underset{M_2}{\Rightarrow}} (p_2, \varepsilon)$

- 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 $|u|$ steps to reach the state p_2 from q_2.

Thus M is a PDA for intersection of $L(M_1)$ and $L(M_2)$.

$$L(M) = L(M_1) \cap L(M_2)$$

4.7.1(g) 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.

4.7.2 Algorithmic Properties (Decision Properties)

There are algorithms for testing :

1. Whether the language generated by a CFG is empty i.e. is $L(G) = \phi$?
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.

4.8 Exam Pack (University Questions)

Syllabus Topic : CFG

Q. Define CFG. *(Refer section 4.2)* (2/5 Marks) **(Dec. 2016, May 2017)**

Syllabus Topic : Leftmost and Rightmost Derivations

Example 4.2.5 (10 Marks) **(Dec. 2014)**
Example 4.2.10 (5 Marks) **(May 2014)**
Example 4.2.28(5 Marks) **(May 2015)**
Example 4.2.41(5 Marks) **(May 2017)**

Syllabus Topic : Ambiguity

Example 4.3.6 (10 Marks) **(Dec. 2016)**
Example 4.4.9 (10 Marks) **(May 2017)**

Syllabus Topic : Normalization

Q. Define Normal form and its types. *(Refer section 4.5)* (5 Marks) **(Dec. 2016)**

Syllabus Topic : CNF

Q. Explain algorithm for the conversion of a Context Free Grammar (CFG) to Chomsky Normal Form (CNF). *(Refer section 4.5.1(a))* (5 Marks) **(May 2014)**

Example 4.5.1(5 Marks) **(Dec. 2014)**
Example 4.5.3(5 Marks) **(May 2014)**
Example 4.5.9(5 Marks) **(May 2017)**

Syllabus Topic : Chomsky Hierarchy (Types 0 to 3)

Q. Write short note on : Chomsky Hierarchy. *(Refer section 4.6)* (5 Marks) **(May 2014)**

Q. Explain Chomsky hierarchy. *(Refer section 4.6)* (10 Marks) **(Dec. 2014, May 2017)**

Scanned by CamScanner

CHAPTER 5

Regular Languages

Syllabus Topics

Regular Languages : Regular Grammars, RL and LL grammar.

Syllabus Topics : Regular Grammars, RL and LL Grammar

5.1 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 :

$$A \rightarrow a$$
$$A \rightarrow aB$$
$$A \rightarrow Ba$$
$$A \rightarrow \varepsilon$$

Where, $a \in T$ and $A, B \in V$.

A language generated by a regular grammar is known as regular language.

A regular grammar could be written in two forms :

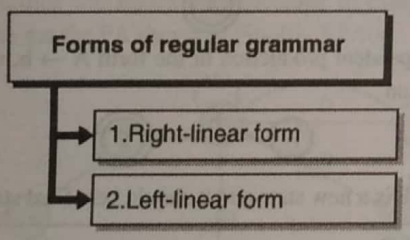


Fig. C5.1 : Form of regular grammar

➜ **1. Right-linear form :** A right linear regular grammar will have production of the given form.

$A \rightarrow a$
$A \rightarrow aB$ | Variable B in $A \rightarrow aB$ is the second symbol on the right.
$A \rightarrow \varepsilon$

➜ **2. Left-linear form :** A left linear regular grammar will have productions of the following form :

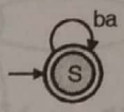
$A \rightarrow a$
$A \rightarrow Ba$ | Variable B in $A \rightarrow Ba$ is the first symbol on the right.
$A \rightarrow \varepsilon$

Example 5.1.1

Construct right linear grammar and left linear grammar for the language (ba)*.

Solution :

Transition system for (ba)* is given by :



We can write left linear grammar and the right linear grammar form the transition systems.

Right linear grammar : $S \rightarrow baS \mid \varepsilon$

Left linear grammar : $S \rightarrow Sba \mid \varepsilon$

5.2 DFA to Right-Linear Regular Grammar

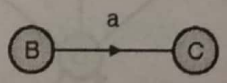
Every DFA can be described using a set of production using the following steps :

Let the DFA, $M = (Q, \Sigma, \delta, q_0, F)$

Let the corresponding right linear grammar

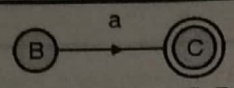
$$G = (V, T, P, S)$$

1. Rename $q_0 \in Q$ as $S \in V$, relating start state of M with starting symbol of G.
2. Rename states of Q as A, B, C, D,.....
   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
   $S \rightarrow \varepsilon$ to P.
   3.2 For every transition of the form,



   add a production $B \rightarrow aC$, where C is a non-accepting state.
   3.3 For every transition of the form

Scanned by CamScanner



add two productions $B \rightarrow aC$, $B \rightarrow a$, where C is an accepting state.

Example 5.2.1

Give a right linear grammar for the DFA shown in Fig. Ex. 5.2.1.

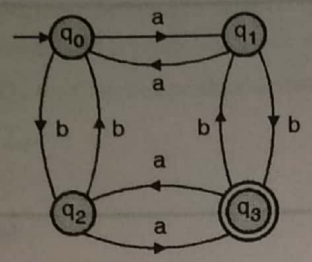


Fig. Ex. 5.2.1

Solution :

Step 1 : Renaming of states, we get the DFA shown in Fig. Ex. 5.2.1(a).

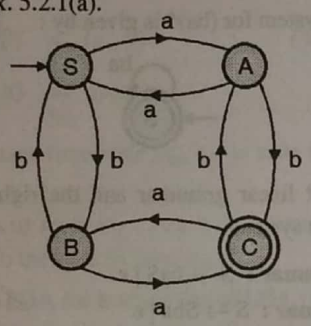


Fig. Ex. 5.2.1(a)

Step 2 : Set of productions are given by :

$$P = \begin{Bmatrix} S \rightarrow aA \mid bB \\ A \rightarrow aS \mid bC \mid b \\ B \rightarrow bs \mid aC \mid a \\ C \rightarrow aB \mid bA \end{Bmatrix}$$

where,

The set of variables $V = \{S, A, B, C\}$

The set of terminals $T = \{a, b\}$

The start symbol $= S$

Example 5.2.2

Give a right linear grammar for the DFA shown in Fig. Ex. 5.2.2.

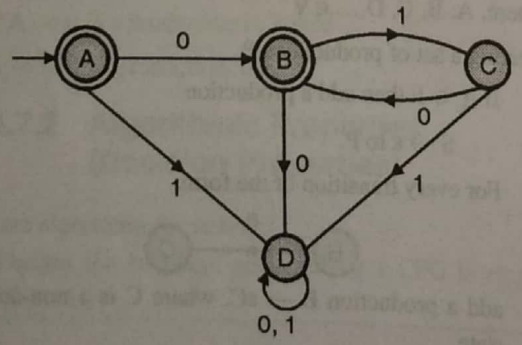


Fig. Ex. 5.2.2

Solution :

Step 1 : State D is a dead state and it can be removed. The FA after deletion of dead state D is shown in Fig. Ex. 5.2.2(a).

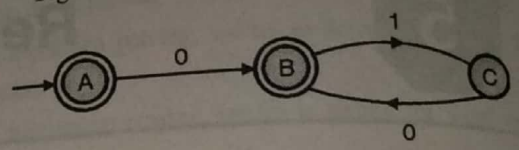


Fig. Ex. 5.2.2(a)

Step 2 : The set of productions are given below.

$$P = \begin{cases} A \rightarrow \varepsilon, & \text{- Start state is a final state} \\ A \rightarrow 0B \mid 0 & \text{- Transition from A to B on 0} \\ B \rightarrow 1C & \text{- Transition from B to C on 1} \\ C \rightarrow 0B \mid 0 & \text{- Transition from C to B on 0} \end{cases}$$

where, Set of variables $V = \{A, B, C\}$

Set of terminals $T = \{0, 1\}$

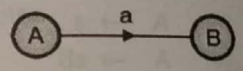
Start symbol $= A$.

5.2.1 Right Linear Grammar to DFA

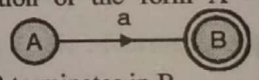
Every right linear grammar can be represented using a DFA

- A production of the form

  $A \rightarrow aB$

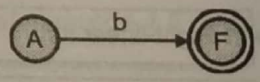
  Will generate a transition 

, 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 5.2.3

Convert the following right-linear grammar to an equivalent DFA.

$S \rightarrow bB$

$B \rightarrow bC$

$B \rightarrow aB$

$C \rightarrow a$

$B \rightarrow b$

Solution :

Re-writing the production we get

$S \rightarrow bB$

$B \rightarrow bC \mid b$

$B \rightarrow aB$

$C \rightarrow a$

Scanned by CamScanner

Step 1 : Adding transitions corresponding to every production, we get

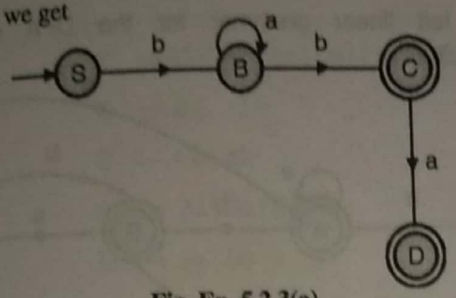


Fig. Ex. 5.2.3(a)

Step 2 : Adding a state E to handle ϕ-transitions, we get the final DFA

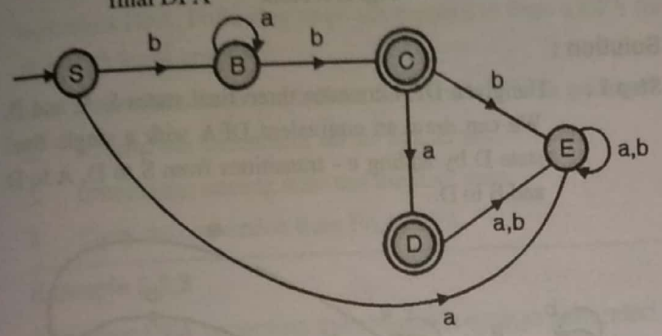


Fig. Ex. 5.2.3(b) : Final DFA

Example 5.2.4

Convert following RG to DFA

$S \rightarrow 0A \mid 1B,$ $\quad A \rightarrow 0C \mid 1A \mid 0,$

$B \rightarrow 1B \mid 1A \mid 1,$ $\quad C \rightarrow 0 \mid 0A.$

Solution : A new final state F is being introduced to handle productions like,

$A \rightarrow 0, B \rightarrow 1, C \rightarrow 0.$

Step 1 : Adding transitions corresponding to every production, we get the FA shown in Fig. Ex. 5.2.4(a).

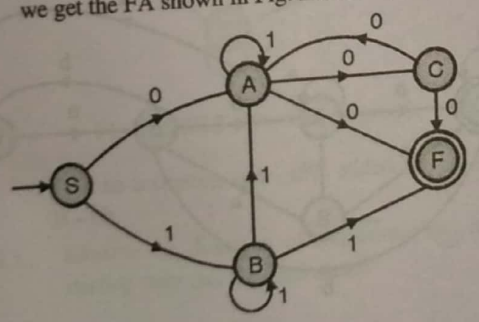


Fig. Ex. 5.2.4(a)

Step 2 : Drawing an equivalent DFA, we get

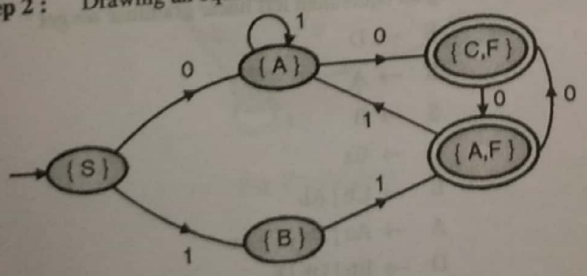


Fig. Ex. 5.2.4(b)

Step 3 : States {S}, {A}, {B}, {C,F}, and {A, F} are renamed as q_0, q_1, q_2, q_3,q_4 and a dead state q_ϕ is introduced to handle ϕ – transitions. The resulting DFA is shown in Fig. Ex.5.2.4(c) :

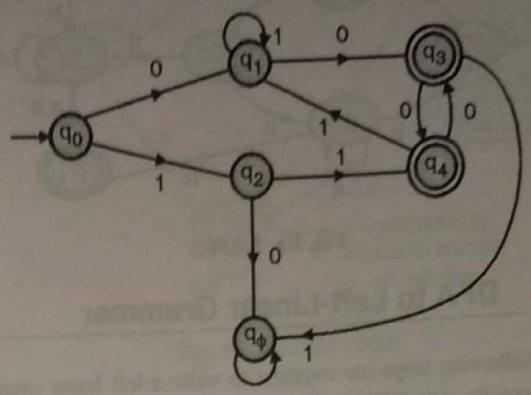


Fig. Ex. 5.2.4(c) : Final DFA

Example 5.2.5

Find the equivalent DFA accepting the regular language defined by the right linear grammar given as :

$S \rightarrow aA \mid bB,$

$A \rightarrow aA \mid bc \mid a$

$B \rightarrow aB \mid b$

$C \rightarrow bB$

Solution : A new final state F is being introduced to handle productions like,

$A \rightarrow a, \quad B \rightarrow b$

Step 1 : Adding transitions corresponding to every production, we get the FA shown in Fig. Ex. 5.2.5(a).

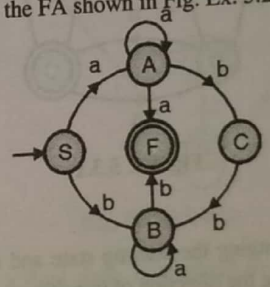


Fig. Ex. 5.2.5(a)

Step 2 : Drawing an equivalent DFA, we get :

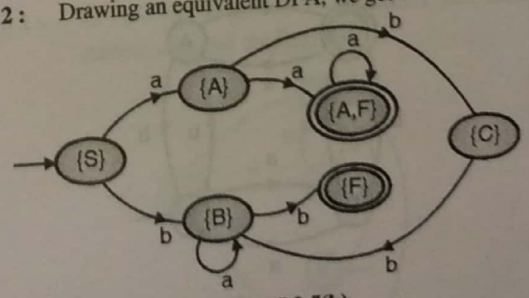


Fig. Ex. 5.2.5(b)

Step 3 : A dead state is added to handle ϕ–transition. The resulting DFA is shown in Fig. Ex. 5.2.5(c).

Scanned by CamScanner

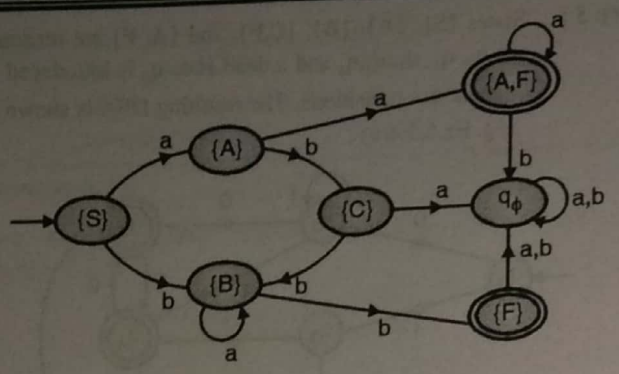


Fig. Ex. 5.2.5(c)

5.3 DFA to Left-Linear Grammar

Following steps are required to write a left linear grammar corresponding to a DFA.

1. Interchange starting state and the final state.
2. Reverse the direction of all the transitions.
3. Write the grammar from the transition graph in left-linear form.

Example 5.3.1

Give a left linear grammar for the DFA shown in Fig. Ex. 5.3.1.

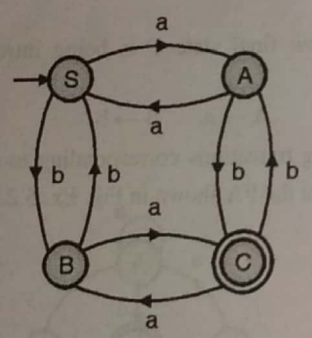


Fig. Ex. 5.3.1

Solution :

Step 1 : Interchanging the starting state and the final state and reversing the direction of transitions, we get a transition graph as shown in Fig. Ex. 5.3.1(a).

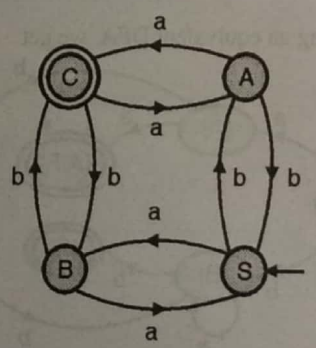


Fig. Ex. 5.3.1(a)

Step 2 : Writing an equivalent left linear grammar, we get :

S → Ba | Ab, A → Sb | Ca | a

B → Sa | Cb | b, C → Bb | Aa.

Example 5.3.2

Give a left linear grammar for the DFA shown in Fig. Ex. 5.3.2.

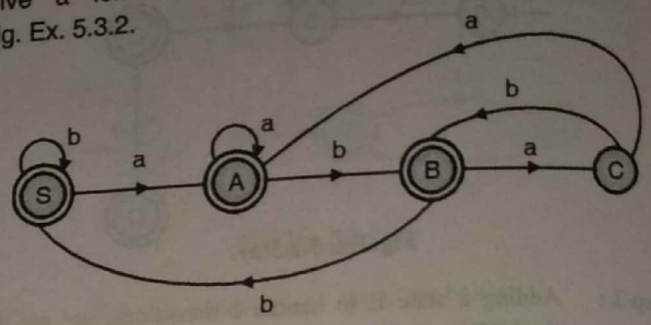


Fig. Ex. 5.3.2

Solution :

Step 1 : The given DFA contains three final states S A, and B. We can draw an equivalent DFA with a single final state D by adding ε - transitions from S to D, A to D and B to D.

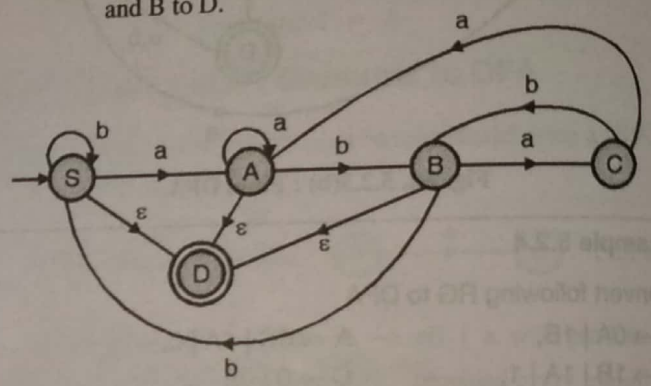


Fig. Ex. 5.3.2(a)

Step 2 : Interchanging the starting state and the final state and reversing direction of transitions, we get

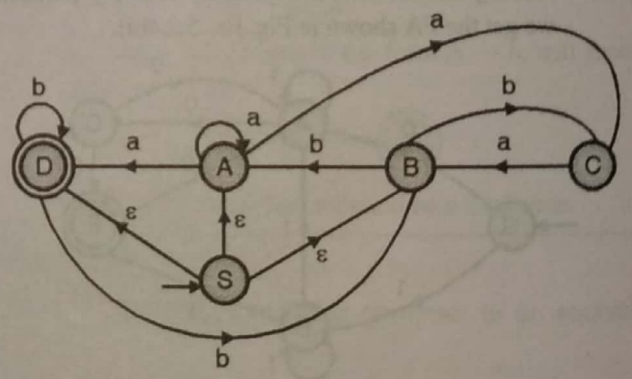


Fig. Ex. 5.3.2(b)

Step 3 : Writing an equivalent left linear grammar we get :

S → D

S → A

S → B

C → Ba

B → Cb | Ab

A → Aa | Da | Ca | a

D → Bb | Db | b

Scanned by CamScanner

Step 4 : Removing unit productions, the resulting productions are :

$$S \rightarrow Cb \mid Ab \mid Aa \mid Da \mid Ca \mid a \mid Bb \mid Db \mid b$$
$$C \rightarrow Ba$$
$$B \rightarrow Cb \mid Ab$$
$$A \rightarrow Aa \mid Da \mid Ca \mid a$$
$$D \rightarrow Ba \mid Db \mid b$$

5.3.1 Left Linear Grammar to DFA

Every left linear grammar can be represented using an equivalent DFA. Following steps are required to draw a DFA for a given left linear grammar.

1. Draw a transition graph from the given left linear grammar.
2. Reverse the direction of all the transitions.
3. Interchange starting state and the final state.
4. Carry out conversion from FA to DFA.

Example 5.3.3

Construct DFA accepting the regular language generated by the left linear grammar given below.

$S \rightarrow Ca \mid Bb$
$C \rightarrow Bb$
$B \rightarrow Ba \mid b$

Solution :

Step 1 : Draw a transition graph from the given left linear grammar.

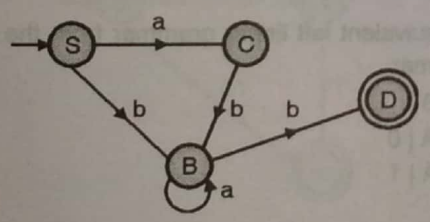


Fig. Ex. 5.3.3 : Transition graph

D is an accepting state. It is added for the production $B \rightarrow b$.

Step 2 : Reverse the direction of transitions and interchange starting state and the final state.

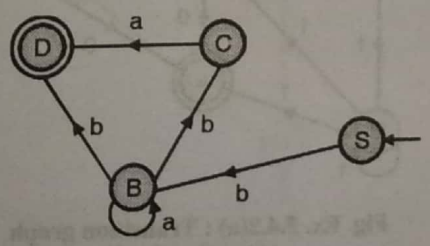


Fig. Ex. 5.3.3(a)

Step 3 : Conversion from FA to DFA

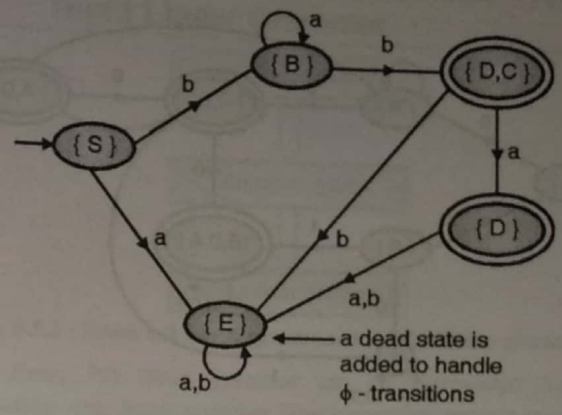


Fig. Ex. 5.3.3(b) : DFA

Example 5.3.4

Construct DFA accepting the language generated by the left linear grammar given below.

$S \rightarrow B1 \mid A0 \mid C0$
$B \rightarrow B1 \mid 1$
$A \rightarrow A1 \mid B1 \mid C0 \mid 0$
$C \rightarrow A0$

Solution :

Step 1 : Draw a transition graph from the given left linear grammar

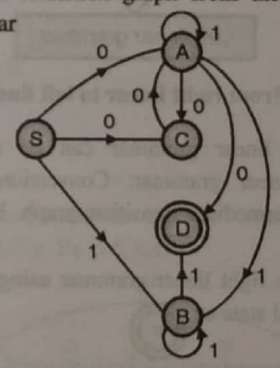


Fig. Ex. 5.3.4(a)

State D is a final state. It is added to handle the following transitions :

$B \rightarrow 1, A \rightarrow 0.$

Step 2 : Reverse the direction of transitions and interchange starting state and the final state.

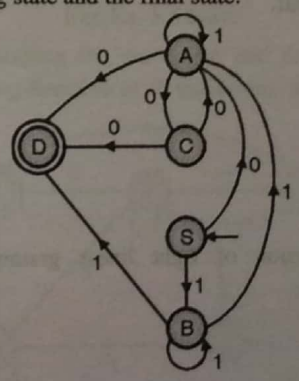


Fig. Ex. 5.3.4(b)

Scanned by CamScanner

Step 3 : Conversion from FA to DFA.

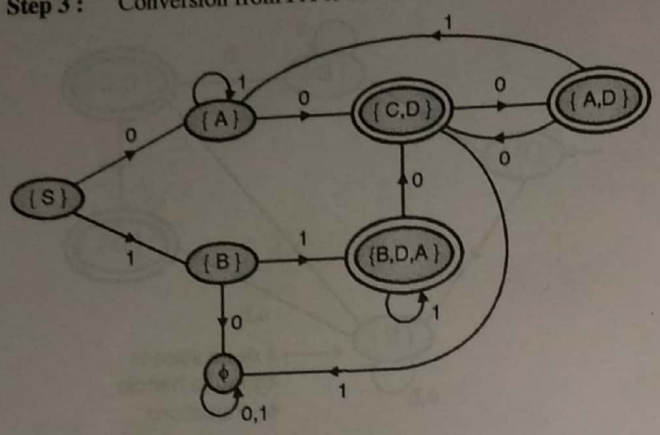


Fig. Ex. 5.3.4(c) : DFA

5.4 Right Linear Grammar to Left Linear Grammar

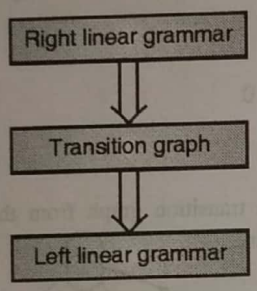


Fig. 5.4.1 : From right linear to left linear grammar

Every right linear grammar can be represented by an equivalent left linear grammar. Conversion process involves drawing of an intermediate transition graph. Following steps are required :

1. Represent the right linear grammar using a transition graph. Mark the final state as ε
2. Interchange the start state and the final state.
3. Reverse the direction of all transitions.
4. Write left linear grammar from the transitions graph.

Example 5.4.1

Convert the following right linear grammar to an equivalent left-linear grammar.

$S \rightarrow bB \mid b$

$B \rightarrow bC$

$B \rightarrow aB$

$C \rightarrow a$

$B \rightarrow b$

Solution :

Step 1 : Conversion of right linear grammar to transition system.

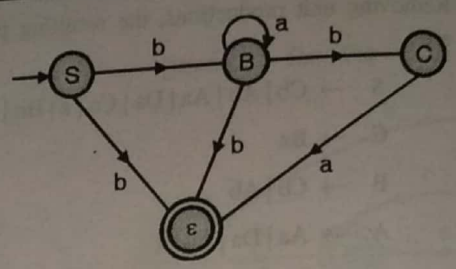


Fig. Ex. 5.4.1(a)

Step 2 : Interchanging the start state with the final state and receiving direction of transitions, we get :

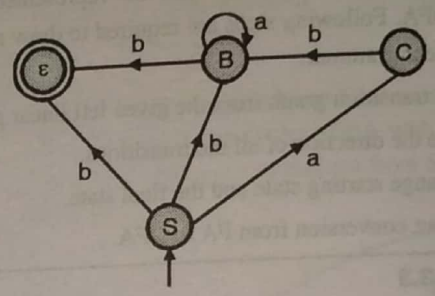


Fig. Ex. 5.4.1(b)

Step 3 : Writing of left linear grammar from the transition system, we get :

$S \rightarrow b \mid Bb \mid Ca$

$B \rightarrow Ba \mid b$

$C \rightarrow Bb$

Example 5.4.2

Write an equivalent left linear grammar from the given right linear grammar.

$S \rightarrow 0A \mid 1B$

$A \rightarrow 0C \mid 1A \mid 0$

$B \rightarrow 1B \mid 1A \mid 1$

$C \rightarrow 0 \mid 0A$

Solution :

Step 1 : Transition system for the given right linear grammar is as shown in Fig. Ex. 5.4.2(a).

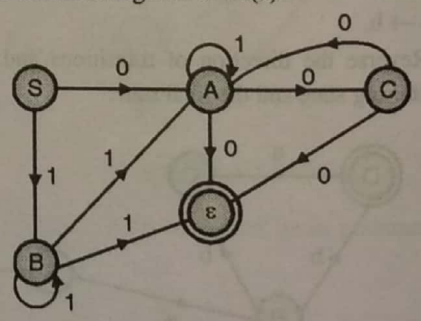


Fig. Ex. 5.4.2(a) : Transition graph

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get

Scanned by CamScanner

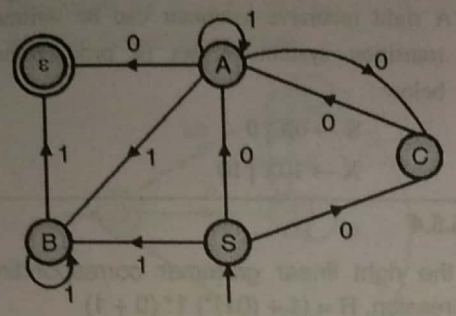


Fig. Ex. 5.4.2(b)

Step 3 : Writing of left linear grammar from the transition system, we get :

$S \rightarrow C0 \mid A0 \mid B1$

$A \rightarrow A1 \mid C0 \mid B1 \mid 0$

$B \rightarrow B1 \mid 1$

$C \rightarrow A0.$

Example 5.4.3

For right linear grammar given below, obtain an equivalent left linear grammar.

$S \rightarrow 10A \mid 01$

$A \rightarrow 00A \mid 1$

Solution :

Step 1 : Transition system from the given right linear grammar is as shown in Fig. Ex. 5.4.3(a).

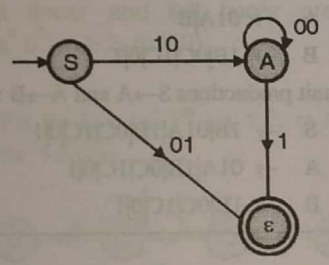


Fig. Ex. 5.4.3(a)

Step 2 : Interchanging the start state with the final state and reversing direction of transitions, we get :

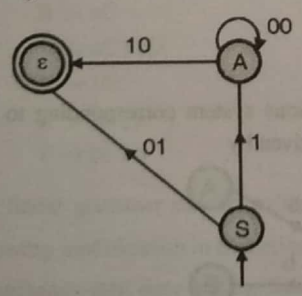


Fig. Ex. 5.4.3(b)

Step 3 : Writing of left linear grammar from the transition system, we get

$S \rightarrow A1 \mid 01$

$A \rightarrow A00 \mid 10$

5.5 Left Linear Grammar to Right Linear Grammar

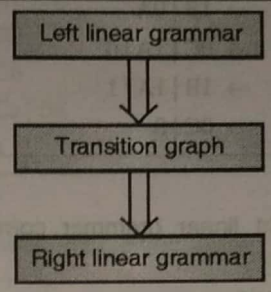


Fig. 5.5.1 : From left linear grammar to right linear grammar

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 5.5.1

Write an equivalent right linear grammar from the given left-linear grammar.

$S \rightarrow C0 \mid A0 \mid B1$

$A \rightarrow A1 \mid C0 \mid B1 \mid 0$

$B \rightarrow B1 \mid 1$

$C \rightarrow A0$

Solution :

Step 1 : Transition system for the left - linear grammar is as shown in Fig. Ex. 5.5.1(a) :

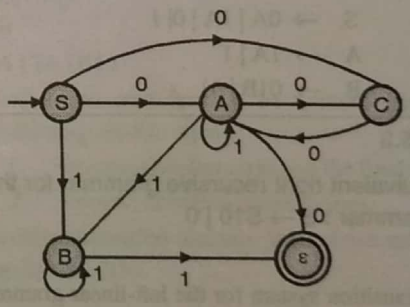


Fig. Ex. 5.5.1(a)

Step 2 : Interchanging the start state and the final state and changing direction of all transitions, we get :

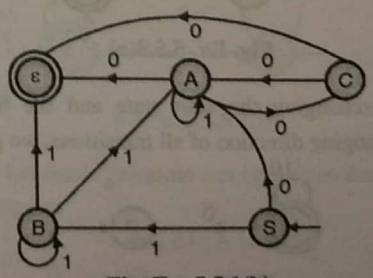


Fig. Ex. 5.5.1(b)

Scanned by CamScanner

Step 3 : A right linear grammar can be written from the transition system. A set of productions is given below :

$$S \rightarrow 1B \mid 0A$$
$$A \rightarrow 0C \mid 1A \mid 0$$
$$B \rightarrow 1B \mid 1A \mid 1$$
$$C \rightarrow 0C \mid 0$$

Example 5.5.2

Construct the right linear grammar corresponding to the regular expression

$R = (0+1)1^*(1+(01)^*)$

Solution :

Step 1 : The given expression can be represented using a transition system as shown in Fig. Ex. 5.5.2.

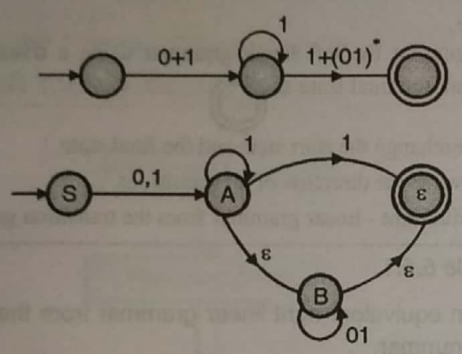


Fig. Ex. 5.5.2

Step 2 : Writing of right linear grammar from the transition system, we get :

$$S \rightarrow 0A \mid 1A,\ A \rightarrow 1A \mid 1 \mid B,\ B \rightarrow 01B \mid \varepsilon$$

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 5.5.3

Write an equivalent right recursive grammar for the given left recursive grammar : $S \rightarrow S10 \mid 0$

Solution :

Step 1 : Transition system for the left-linear grammar is shown in Fig. Ex. 5.5.3(a) :

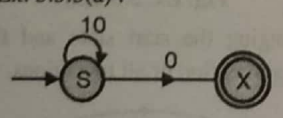


Fig. Ex. 5.5.3(a)

Step 2 : Interchanging the start state and the final state and changing direction of all transitions, we get :

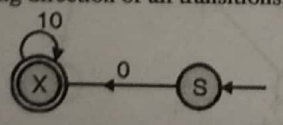


Fig. Ex. 5.5.3(b)

Step 3 : A right recursive grammar can be written from the transition system. A set of productions is given below :

$$S \rightarrow 0X \mid 0$$
$$X \rightarrow 10X \mid 10$$

Example 5.5.4

Construct the right linear grammar corresponding to the regular expression. $R = (1 + (01)^*)\ 1^*\ (0 + 1)$

Solution :

The R.E. $(1 + (01)^*)\ 1^*(0 + 1)$ can be represented using a transition system as shown in Fig. Ex. 5.5.4.

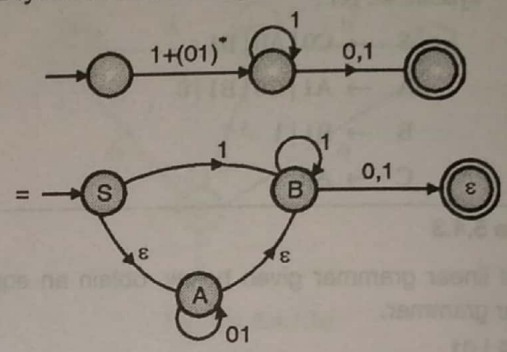


Fig. Ex. 5.5.4

Writing of right linear grammar from the transition system, we get :

$$S \rightarrow 1B|A$$
$$A \rightarrow 01A|B$$
$$B \rightarrow 1B|0C|1C|0|1$$

Removing unit productions $S \rightarrow A$ and $A \rightarrow B$:

$$S \rightarrow 1B|01A|1B|0C|1C|0|1$$
$$A \rightarrow 01A|1B|0C|1C|0|1$$
$$B \rightarrow 1B|0C|1C|0|1$$

Example 5.5.5

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

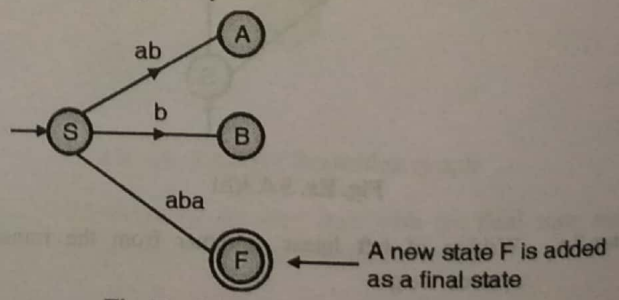


Fig. Ex. 5.5.5(a) : Transition graph

Scanned by CamScanner

Step 2 : Transitions for $A \to b \mid aB \mid bA$ and $B \to aB \mid aA$ are added to the transition graph

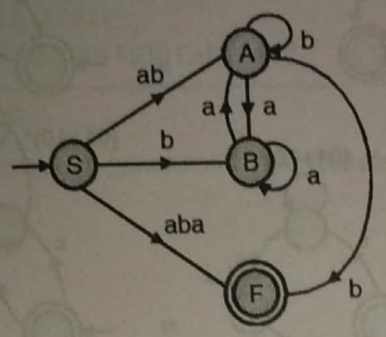


Fig. Ex. 5.5.5(b)

Step 3 : Transitions on ab and aba are expanded with resulting NFA as shown in Fig. Ex. 5.5.5(c).

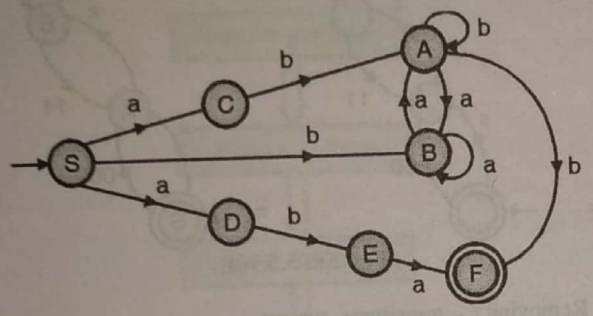


Fig. Ex. 5.5.5(c)

Example 5.5.6

Construct right linear and left linear grammar for the language $L = \{a^n b^m, n \geq 2, m \geq 3\}$

Solution :

Step 1 : A DFA for $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$ is shown in Fig. Ex. 5.5.6(a) :

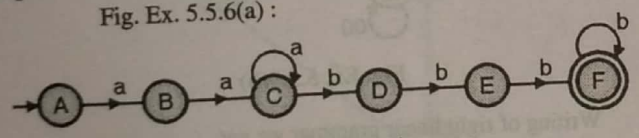


Fig. Ex. 5.5.6(a)

Step 2 : Right linear grammar can be written from the DFA.

$A \to aB$
$B \to aC$
$C \to aC \mid bD$
$D \to bE$
$E \to bF \mid b$
$F \to bF \mid b$

Step 3 : Left linear grammar can be written after making the following modification in transition graph.

1. Interchange start state and the final state
2. Reverse direction of transitions.

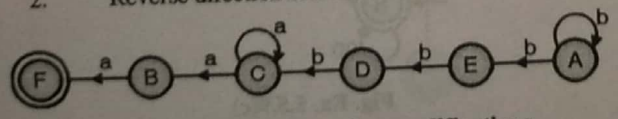


Fig. Ex. 5.5.6(b) : DFA after modifications

Step 4 : Left linear grammar can be written from the transition graph.

$A \to Ab \mid Eb$
$E \to Db$
$D \to Cb$
$C \to Ca \mid Ba$
$B \to a$

Example 5.5.7

Construct left linear and right linear grammar for the language $0^* (1 (0 + 1))^*$

Solution :

Step1 : We will first draw a transition diagram from the given regular expression.

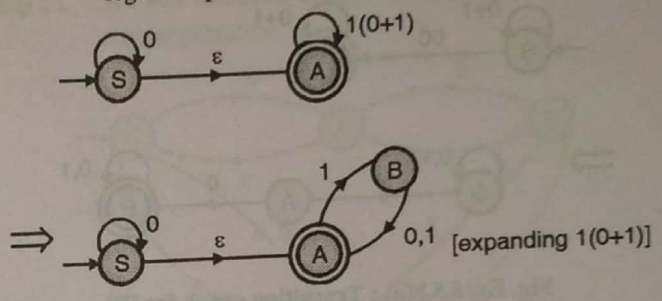


Fig. Ex. 5.5.7(a) : Transition graph for given RE

Step 2 : A right linear grammar can be written from the transition graph.

$S \to 0S \mid A \mid \varepsilon$
$A \to 1B$
$B \to 0A \mid 1A \mid 0 \mid 1$

Removing the unit production, $S \to A$, the resulting grammar is given below :

$S \to 0S \mid 1B \mid \varepsilon$
$A \to 1B$
$B \to 0A \mid 1A \mid 0 \mid 1$

Step 3 : Left linear grammar can be written after making the following modifications :

1. Interchange start state and the final state
2. Reverse direction of transitions.

The resulting transition diagram with above modifications is shown in Fig. Ex. 5.5.7(b).

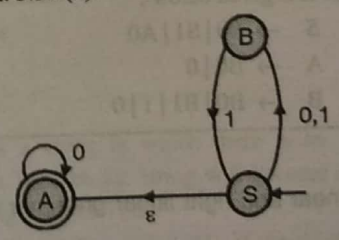


Fig. Ex. 5.5.7(b)

Step 4 : A left linear grammar can be written from the transition diagram

$S \to B0 \mid B1 \mid A \mid \varepsilon$
$B \to S1$

Scanned by CamScanner

$A \rightarrow A0 \mid 0$

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 5.5.8

Construct left linear and right linear grammar for the language $(0 + 1)^*00(0 + 1)^*$

Solution :

Step1 : We will first draw a transition diagram from the given regular expression.

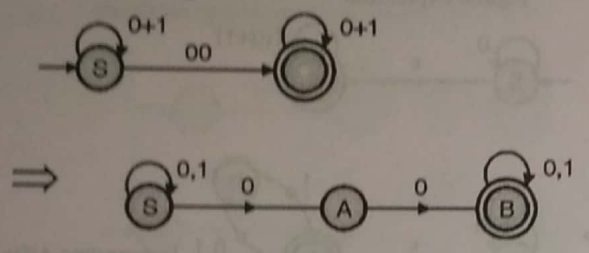


Fig. Ex. 5.5.8(a) : Transition graph for RE

Step 2 : Right linear grammar, from the transition graph can be written as given below :

$S \rightarrow 0S \mid 1S \mid 0A$

$A \rightarrow 0B \mid 0$

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

Right linear grammar

Step 3 : Left linear grammar can be written after making the following modifications :

1. Interchange start state and the final state
2. Reverse direction of transitions.

Resulting transition diagram with above modifications is shown in Fig. Ex. 5.5.8(b) :

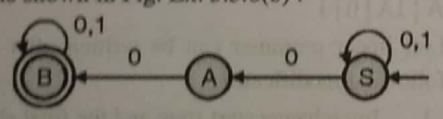


Fig. Ex. 5.5.8(b)

Step 4 : A left linear grammar can be written from the transition graph. It is given below :

$S \rightarrow S0 \mid S1 \mid A0$

$A \rightarrow B0 \mid 0$

$B \rightarrow B0 \mid B1 \mid 1 \mid 0$

Example 5.5.9

Construct left linear and right linear grammar for the regular expression.

$((01 + 10)^*11)^*00)^*$

Solution :

The given expression can be represented using a transition system as shown below :

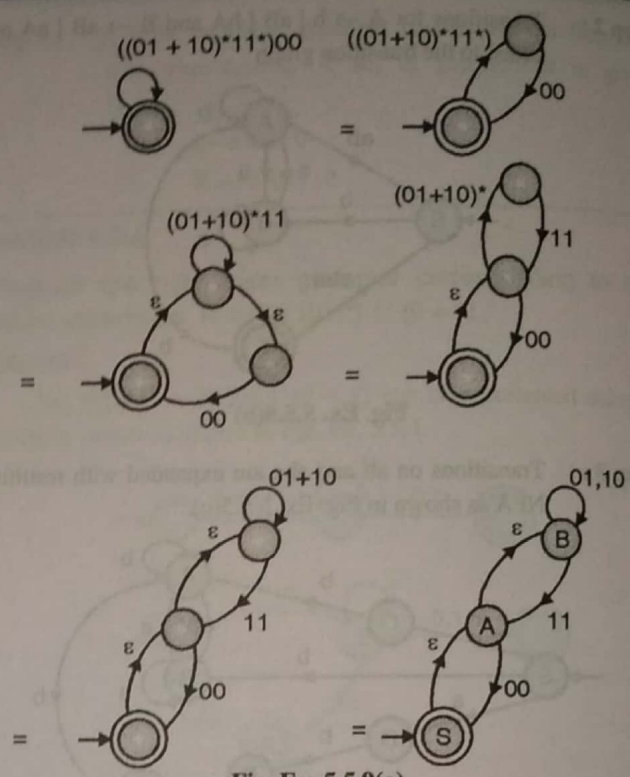


Fig. Ex. 5.5.9(a)

Removing ε – transitions, we get :

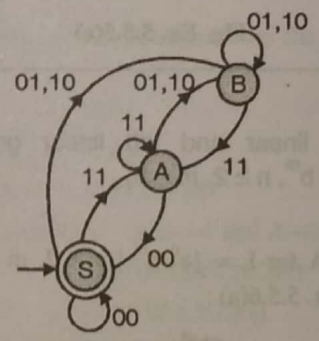


Fig. Ex. 5.5.9(b)

Writing of right linear grammar we get,

$S \rightarrow 00S \mid 11A \mid 01B \mid 10B \mid \varepsilon$

$A \rightarrow 11A \mid 01B \mid 10B \mid 00S$

$B \rightarrow 01B \mid 10B \mid 11A$

For writing of left linear grammar, we interchange the start state and the final state and change direction of all transitions. The resulting transition system is given by :

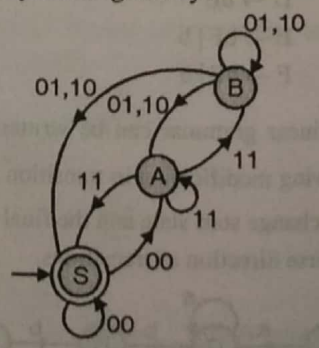


Fig. Ex. 5.5.9(c)

Scanned by CamScanner

Writing of left linear grammar we get,

$S \rightarrow S00 \mid A00 \mid \varepsilon$

$A \rightarrow A11 \mid B11 \mid S11$

$B \rightarrow B01 \mid 10B \mid S01 \mid S10 \mid A01 \mid A10$

Example 5.5.10

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 shown in Fig. Ex. 5.5.10(a) :

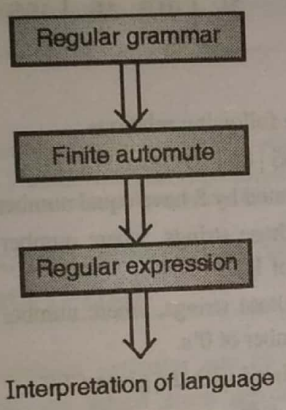


Fig. Ex. 5.5.10(a)

Step 1 : We will first draw a transition diagram from the given regular grammar.

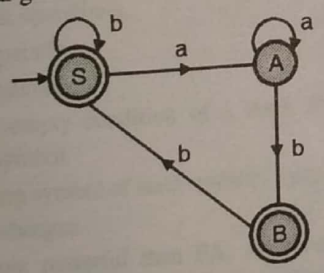


Fig. Ex. 5.5.10(b) : Transition graph

S is a final state, as $S \rightarrow \varepsilon$

B is a final state, as $A \rightarrow bB \mid b$

Step 2 : A regular expression can be written easily by moving the effect of loop $S \rightarrow A \rightarrow B \rightarrow S$ on S. The resulting transition diagram is shown in Fig. Ex. 5.5.10(c).

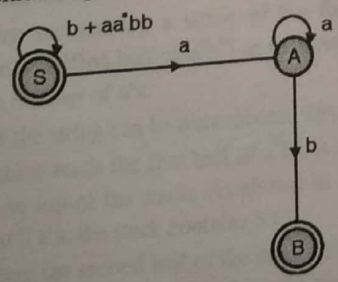


Fig. Ex. 5.5.10(c)

$\therefore$ 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 5.5.11

Describe the language generated by the following grammar.

$S \rightarrow aA \mid bC \mid b$

$A \rightarrow aS \mid bB$

$B \rightarrow aC \mid bA \mid a$

$C \rightarrow aB \mid bS$

Solution :

A language can be understood easily from a regular expression or finite automata.

Step 1 : We will first draw a transition diagram from the given regular grammar.

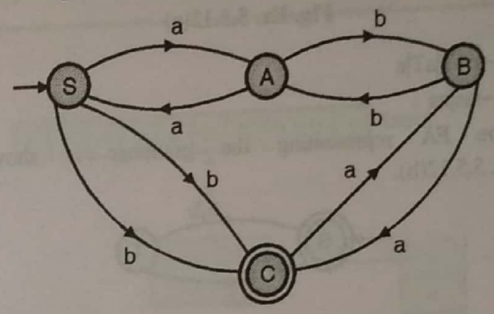


Fig. Ex. 5.5.11 : Transition graph for RE

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 is,

$L = \{\omega \in \{a, b\}^* \mid \omega$ contains even number of a's and odd umber of b's$\}$.

Example 5.5.12

Describe the language generated by each of these grammar and justify your answer with the example string derive from the grammar of the productions given below.

(1) $S \rightarrow aSa|bsb|aAb|bAa$

$A \rightarrow aAa|bAb|a|b|\varepsilon$

(2) $S \rightarrow bT|aT|\varepsilon$

$T \rightarrow aS|bS$

Solution :

(1) It generates a string in which there is an 'a' and if it is rewritten as 'b' then the string will become a palindrome as shown in Fig. Ex. 5.5.12(a).

The string abaabbabba is in L(G). If the third character 'a' is made b then the modified string abbabbabba is a palindrome.

Scanned by CamScanner

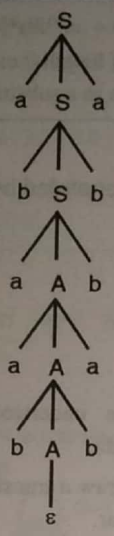


Fig. Ex. 5.5.12(a)

(2) $S \rightarrow bT|aT|\varepsilon$

$A \rightarrow as|bs$

The FA representing the grammar is shown in Fig. Ex. 5.5.12(b).

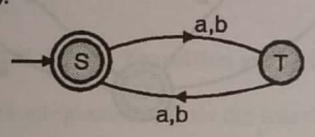


Fig. Ex. 5.5.12(b)

Thus the grammar generates the following language :

$$L(G) = \{\omega \in \{a,b\}^* \mid |\omega| \text{ is even}\}$$

Example 5.5.13

Find the CFL associated with CFG

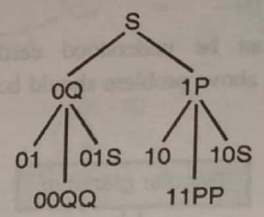
$S \rightarrow 0Q \mid 1P$

$P \rightarrow 0 \mid 0S \mid 1PP$

$Q \rightarrow 1 \mid 1S \mid 0QQ$

Solution :

This is a case of indirect recursive. S is related back to itself through P or Q.



There are the following relations :

$S \rightarrow 01S \mid 10S \mid 01 \mid 10 \mid 00QQ \mid 11PP$

- Strings generated by S have equal number of 0's and 1's.
- P generates those strings, where number of 0's is one more than number of 1's.
- Q generates those strings, where number of 1's is one more than them number of 0's.

$\therefore$ CFL associated with the given CFG

$= \{w \in \{0, 1\}^* \mid w \text{ contain equal number of 0's and 1's}\}$.

❑❑❑

Scanned by CamScanner

CHAPTER 6

Push Down Automata

Syllabus Topics

Push Down Automata : Deterministic (single stack) PDA, Equivalence between PDA and CFG.

Syllabus Topic : Deterministic(Single Stack) PDA

6.1 Introduction to Push Down Automata (PDA)

Informally, pushdown automata can be viewed as finite automata with stack. An added stack provides memory and increases the capability of the machine.

A pushdown automata can do the followings :

1. Read input symbol [as in case of FA]
2. Perform stack operations.
   - 2.1 Push operation
   - 2.2 Pop operation
   - 2.3 Check empty condition of a stack through an initial stack symbol.
   - 2.4 Read top symbol of stack without a pop.
3. Make state changes.

PDA is more powerful than FA. A context-free language (CFL) can be recognized by a PDA. Only a subset of CFL that are regular can be recognized by finite automata.

- A context free language can be recognized by PDA.
- For every context-free language, there exists a PDA.
- The language of PDA is a context-free language.

Example : A string of the form $a^n b^n$ can not be handled by a finite automata. But the same can be handled by a PDA.

- Any machine recognizing a string of the form $a^n b^n$, must keep track of a's [first half of $a^n b^n$] as number of b's must be equal to the number of a's.
- First half of the string can be remembered through a stack.
- As the machine reads the first half of $a^n b^n$, it remembers it by pushing it on top of the stack. As shown in Fig. 6.1.1, after reading first 5 a's, the stack contains 5 a's.
- While reading the second half of the input string consisting of b's, the machine pops out an 'a' from the stack for every 'b' as input.

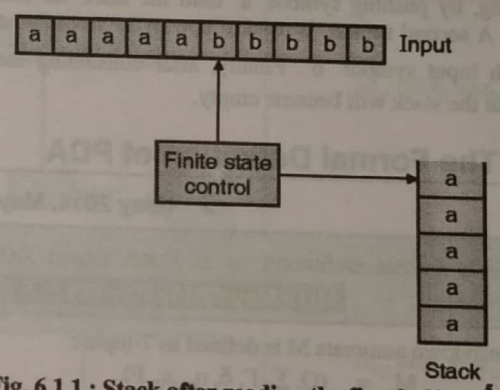


Fig. 6.1.1 : Stack after reading the first half of $a^5 b^5$

- After reading 5 b's, input will finish and the stack will become empty. This will indicate that the input string is of the form $a^n b^n$.
- The machine will have two states q_0 and q_1.

  State q_0 – while the machine is reading a's

  State q_1 – While the machine is reading b's.

  While in state q_1; an input 'a' is not allowed and hence there is a need for two states.
- A transition in PDA depends on :
  1. Current state
  2. Current input
  3. Top symbol of the stack.

A transition in PDA can be shown as a directed edge from the state q_i to q_j. While moving to state q_j, the machine can also perform stack operation. A transition edge from q_i to q_j should be marked with current input, current stack symbol (topmost symbol of the stack) and the stack operation. It is shown in Fig. 6.1.2.

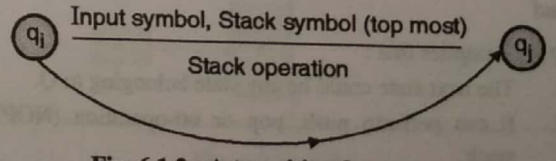


Fig. 6.1.2 : A transition from q_i to q_j

Scanned by CamScanner

- A PDA uses three stack operations :
  (1) **Pop** operation, it removes the top symbol from the stack.
  (2) **Push** operation, it inserts a symbol onto the top of the stack.
  (3) **Nop** operation, it does nothing to stack.
- The language $\{a^n b^n \mid n \geq 1\}$ can be accepted by the PDA of Fig. 6.1.3.

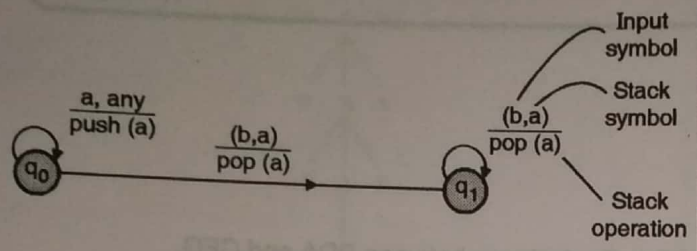


Fig. 6.1.3 : PDA for $a^n b^n$

- The state q_0 will keep track of the number of a's in an input string, by pushing symbol 'a' onto the stack for each input 'a'. A second state q_1 is used to pop an 'a' from the stack for each input symbol 'b'. Finally, after consuming the entire input the stack will become empty.

6.2 The Formal Definition of PDA

➔ **(May 2014, May 2015)**

Q. Give formal definition of a Push Down Automata (PDA). **MU - May 2014, May 2015, 5 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

$$Q \times (\Sigma \cup \varepsilon) \times \Gamma \text{ to } Q \times \Gamma^*$$

Where

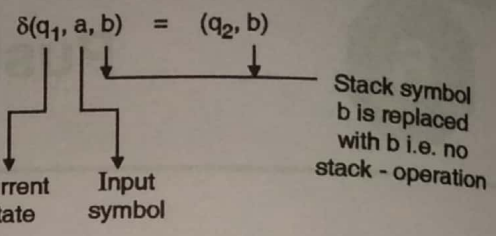
$Q \times (\Sigma \cup \varepsilon) \times \Gamma$ implies that a transition is based on :
1. Current state
2. Next input (including ε)
3. Stack symbol (topmost)

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

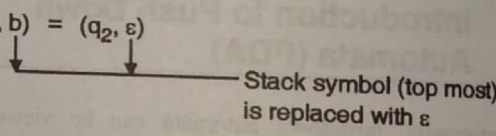


- Left hand side of the transition, $\delta(q_1, a, b)$ implies :
  q_1 is the current state,
  a is the current input,
  b is the stack symbol (topmost).
- Right hand side of the transition, (q_2, b) implies that

[The machine enters a new state q_2. The top symbol of the stack, which is b is replaced with b.]

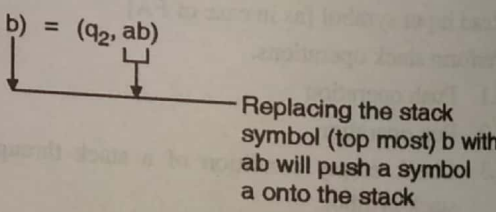
Thus, the transition $\delta(q_1, a, b) = (q_2, b)$ does not modify a stack.

2. Pop operation



The above transition will erase the stack symbol (topmost). Replacing, b with ε amounts to erasing b from the stack top.

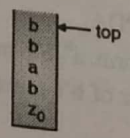
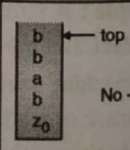
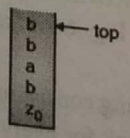
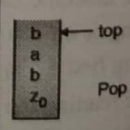
3. Push operation



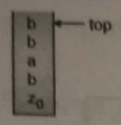
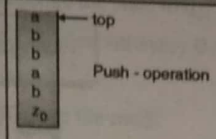
The above transition will perform a push operation. It will push 'a' onto the stack. Replacing 'b' with 'ab' amounts to push 'a' onto the stack.

The three operations for stack are shown in Table 6.2.1.

Table 6.2.1 : Transitions rules for stack operations

Stack before transition (assumed)	Transition rule	Stack after transition
1. 	$\delta(q_1, a, b) = (q_2, b)$	 No - operation
2. 	$\delta(q_1, a, b) = (q_2, \varepsilon)$	 Pop - operation

Scanned by CamScanner

Stack before transition (assumed)	Transition rule	Stack after transition
3. 	$\delta(q_1, a, b) = (q_2, ab)$	

Example : Transition function for a PDA accepting a language

$L = \{\omega \in \{a, b\}^* \mid \omega \text{ is of the } a^n b^n \text{ with } n \geq 1\}$,

can be givgen 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)

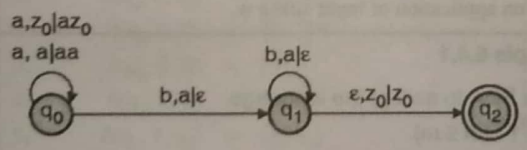


Fig. 6.2.1 : Transition diagram

6.3 Instantaneous Description of a PDA

Instantaneous behaviour is useful in showing the processing of a string by PDA. Processing of the input string a^3b^3 by the PDA of Fig. 6.2.1 is shown in Fig. 6.3.1.

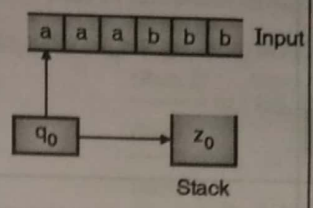
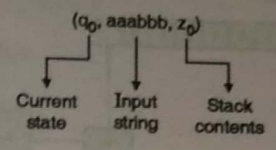
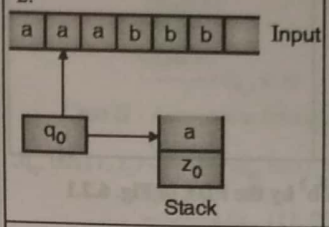
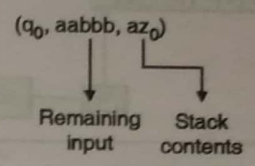
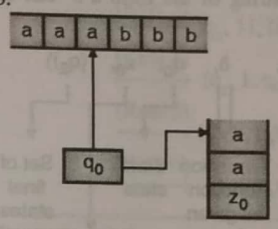
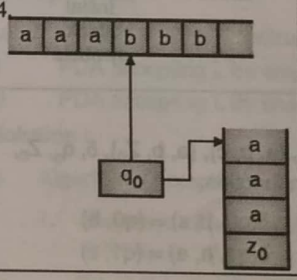
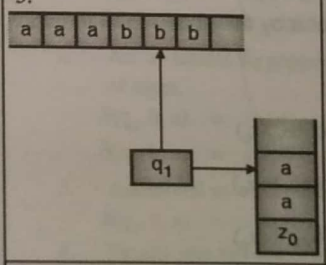
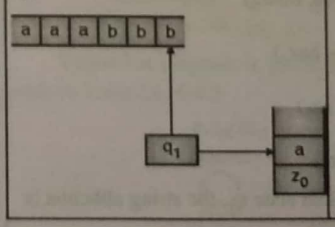
Graphical representation (instantaneous behaviour)	Tuple format (instantaneous behaviour)
1. 	
2. 	
3. 	$(q_0, abbb, aaz_0)$
4. 	$(q_0, bbb, aaaz_0)$
5. 	(q_1, bb, aaz_0)
6. 	(q_1, b, az_0)

Fig. 6.3.1 contd....

Scanned by CamScanner

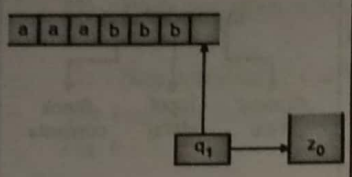
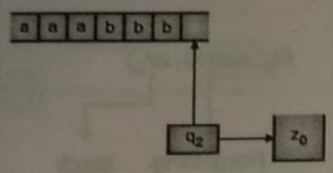
Graphical representation (instantaneous behaviour)	Tuple format (instantaneous behaviour)
7. 	(q_1, ε, z_0)
8. 	(q_2, ε, z_0)

Fig. 6.3.1 : Processing of a^3b^3 by the PDA of Fig. 6.2.1

The PDA for recognizing a string of the form a^nb^n can be described as:

$M = (\{q_0, q_1, q_2\}, \{a,b\}, \{a,b,z_0\}, \delta, q_0, z_0, \{q_2\})$

- $\{q_0, q_1, q_2\}$ → Q, the set of states
- $\{a,b\}$ → Σ, input symbols
- $\{a,b,z_0\}$ → Γ, Stack symbols
- δ → Transition function as given above
- q_0 → Start state
- z_0 → Initial stack symbols
- $\{q_2\}$ → Set of final states

Example 6.3.1

Suppose the PDA $M = (\{q_0, q_1\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_1\})$ has the following transition function.

1. $\delta(q0, a, \varepsilon) = (q0, a)$
2. $\delta(q0, b, \varepsilon) = (q0, b)$
3. $\delta(q0, c, \varepsilon) = (q1, \varepsilon)$
4. $\delta(q1, a, a) = (q1, \varepsilon)$
5. $\delta(q1, b, b) = (q1, \varepsilon)$

Show the acceptance of abbcbba by the above PDA through an instantaneous description.

Solution :

$$(q_0, abbcbba, z_0) \xrightarrow{\text{(Rule 1)}} (q_0, bbcbba, az_0)$$
$$\xrightarrow{\text{(Rule 2)}} (q_0, bcbba, baz_0)$$
$$\xrightarrow{\text{(Rule 2)}} (q_0, cbba, bbaz_0)$$
$$\xrightarrow{\text{(Rule 3)}} (q_1, bba, bbaz_0)$$
$$\xrightarrow{\text{(Rule 5)}} (q_1, ba, baz_0)$$
$$\xrightarrow{\text{(Rule 5)}} (q_1, a, az_0)$$
$$\xrightarrow{\text{(Rule 4)}} (q_1, \varepsilon, z_0)$$

Since, the machine is in a final state q_1, the string abbcbba is accepted.

6.4 The Language of a PDA

A language L can be accepted by a PDA in two ways :

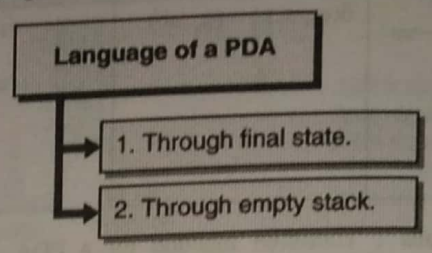


Fig. C6.1 : Language of a PDA

It is possible to convert between the two classes.

1. From final state to empty stack.
2. From empty stack to final state.

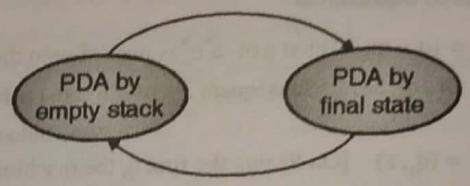


Fig. 6.4.1 : Equivalence of two PDAs

→ 6.4.1 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.

→ 6.4.2 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.

Example 6.4.1

Give a PDA to accept the language

$L = \{0^n1^m \mid n \le m\}$

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]

Scanned by CamScanner

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. 6.4.1(a to d)

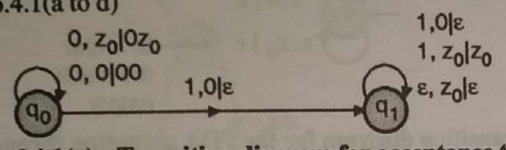


Fig. Ex. 6.4.1(a) : Transition diagram for acceptance through an empty stack

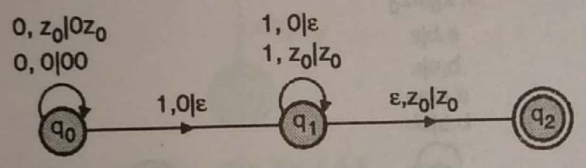


Fig. Ex. 6.4.1(b) : Transition diagram for acceptance through a final state

1	$\delta(q_0, 0, z_0)$	=	$(q_0, 0z_0)$
2.	$\delta(q_0, 0, 0)$	=	$(q_0, 00)$
3.	$\delta(q_0, 1, 0)$	=	(q_1, ε)
4.	$\delta(q_1, 1, 0)$	=	(q_1, ε)
5.	$\delta(q_1, 1, z_0)$	=	(q_1, z_0)
6.	$\delta(q_1, \varepsilon, z_0)$	=	(q_1, ε)

Fig. Ex. 6.4.1(c) : State transition rules for acceptance through an empty stack

1.	$\delta(q_0, 0, z_0)$	=	$(q_0, 0z_0)$
2.	$\delta(q_0, 0, 0)$	=	$(q_0, 00)$
3.	$\delta(q_0, 1, 0)$	=	(q_1, ε)
4.	$\delta(q_1, 1, 0)$	=	(q_1, ε)
5.	$\delta(q_1, 1, z_0)$	=	(q_1, z_0)
6.	$\delta(q_1, \varepsilon, z_0)$	=	(q_2, z_0)

Fig. Ex. 6.4.1(d) : State transition rules for acceptance through a final state

The PDA accepting through an empty stack is given by :

$$M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \phi)$$

Where δ is given in Fig. Ex. 6.4.1(c).

The PDA accepting through final state is given by :

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Where δ is given in Fig. Ex. 6.4.1(d).

Example : Processing of string 00111 by the PDA

Case I : Acceptance through empty stack.

$$(q_0, 00111, z_0) \xrightarrow{\text{(Rule 1)}} (q_0, 0111, 0z_0)$$
$$\xrightarrow{\text{(Rule 2)}} (q_0, 111, 00z_0)$$
$$\xrightarrow{\text{(Rule 3)}} (q_1, 11, 0z_0)$$
$$\xrightarrow{\text{(Rule 4)}} (q_1, 1, z_0)$$
$$\xrightarrow{\text{(Rule 5)}} (q_1, \varepsilon, z_0)$$
$$\xrightarrow{\text{(Rule 6)}} (q_1, \varepsilon, \varepsilon)$$

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 6.4.2

Let $L = \{a^m b^n \mid n < m\}$. Construct,

i) PDA accepting L by empty store.

ii) PDA accepting L by final state.

Solution :

i) **Algorithm (Accepting L by empty store)**

1. 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. An 'a' should be popped for every b as input till the end of input.

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

3. Additional m – n a's should be popped from the stack.

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

4. Finally, the symbol z_0 should be popped out to make the state empty.

$$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$$

Transition diagram is given in Fig. Ex. 6.4.2 and transition table in Table Ex. 6.4.2

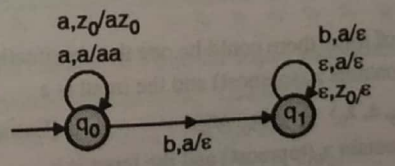


Fig. Ex. 6.4.2

Scanned by CamScanner

Table Ex. 6.4.2

$\delta(q_0,a,z_0) = (q_0,az_0)$
$\delta(q_0,a,a) = (q_0,aa)$
$\delta(q_0,b,a) = (q_1,\varepsilon)$
$\delta(q_1,b,a) = (q_1,\varepsilon)$
$\delta(q_1,\varepsilon,a) = (q_1,\varepsilon)$
$\delta(q_1,\varepsilon,z_0) = (q_1,\varepsilon)$

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$$

ii) Accepting L through final state

Following modifications are required to accept through a final state.

In step3, when the input ends with stack containing m-n a's, the machine should enter the final state.

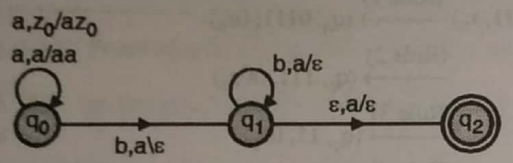


Fig. Ex. 6.4.2(a)

Table Ex. 6.4.2(a)

$\delta(q_0,a,z_0) = (q_0,az_0)$
$\delta(q_0,a,a) = (q_0,aa)$
$\delta(q_0,b,a) = (q_1,\varepsilon)$
$\delta(q_1,b,a) = (q_1,\varepsilon)$
$\delta(q_1,\varepsilon,a) = (q_2,\varepsilon)$

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$$

Example 6.4.3

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. 6.4.3.

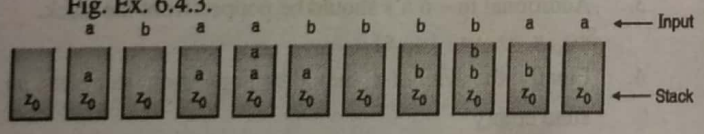


Fig. Ex. 6.4.3 : Stack preserving excess of a's over b's or b's over a's

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]

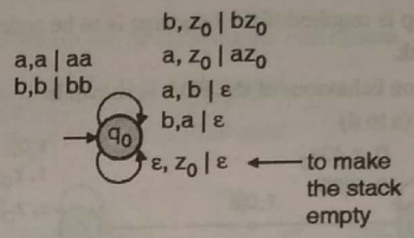
5. Stack contains b(topmost) and the input is a.

   $\delta(q_0, a, b) = (q_0, \varepsilon)$ [Excess b's will decrease by 1]

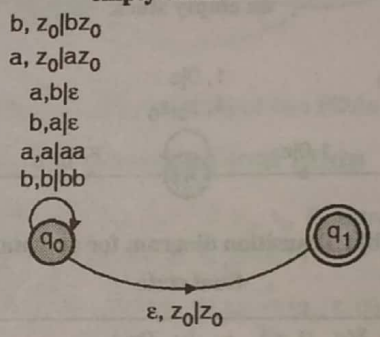
6. Stack contains b(topmost) and the input is b.

   $\delta(q_0, b, b) = (q_0, bb)$ [Excess b's will increase by 1]

The PDA is shown in Fig. Ex. 6.4.3(a,b).



(a) Transition diagram for the PDA accepting through an empty stack



(b) Transition diagram for the PDA accepting through a final state

Fig. Ex. 6.4.3

$\delta(q_0, a, z_0) = (q_0, az_0)$	$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$	$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, b) = (q_0, \varepsilon)$	$\delta(q_0, a, b) = (q_0, \varepsilon)$
$\delta(q_0, b, a) = (q_0, \varepsilon)$	$\delta(q_0, b, a) = (q_0, \varepsilon)$
$\delta(q_0, a, a) = (q_0, aa)$	$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$	$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$	$\delta(q_0, \varepsilon, z_0) = (q_1, z_0)$

Fig. Ex. 6.4.3(c) : Transition rules for the PDA accepting through empty stack | **Fig. Ex. 6.4.3(d) : Transition rules for the PDA accepting through final state**

Fig. Ex. 6.4.3 (a to d) shows state transition behaviour of PDA of Example 6.4.3.

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. 6.4.3(c).

The PDA accepting through final state is given by :

$$M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\}),$$

Where, δ is given in Fig. Ex. 6.4.3(d).

Scanned by CamScanner

Example 6.4.4

Design a PDA to accept $(ab)^n(cd)^n$.

Solution : To solve this problem, we can take a stack symbol x. For every 'ab', one x will be pushed on top of the stack. After reading $(ab)^n$, the stack should contain n number of x's. These x's will be matched with $(cd)^n$. For every 'cd' one x will be popped.

The transitions for the PDA accepting through an empty stack are given in Fig. Ex. 6.4.4.

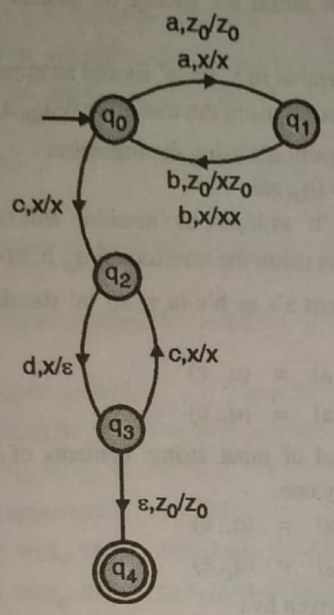


Fig. Ex. 6.4.4

– PDA accepts through the final state q_4.

The PDA M = $\{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

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

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

$\Gamma = \{x, z_0\}$

The transition function δ is given by,

$\delta(q_0, a, z_0) = (q_1, z_0)$

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

$\delta(q_1, b, z_0) = (q_0, x z_0)$

$\delta(q_1, b, x) = (q_0, xx)$

$\delta(q_0, c, x) = (q_2, x)$

$\delta(q_2, d, x) = (q_3, \varepsilon)$

$\delta(q_3, c, x) = (q_2, x)$

$\delta(q_2, \varepsilon, z_0) = (q_4, z_0)$

q_0 is initial state,

z_0 is initial stack symbol.

Set of final states $F = \{q_4\}$

Example 6.4.5

Design a PDA to accept $(bdb)^n c^n$

Solution : To solve this problem, we can take a stack symbol x. For every 'bdb', one x will be pushed on top of the stack. After reading $(bdb)^n$, the stack should contain n number of x's. These x's will be matched with c's.

The transitions for the PDA accepting through an empty stack are given in Fig. Ex.6.4.5.

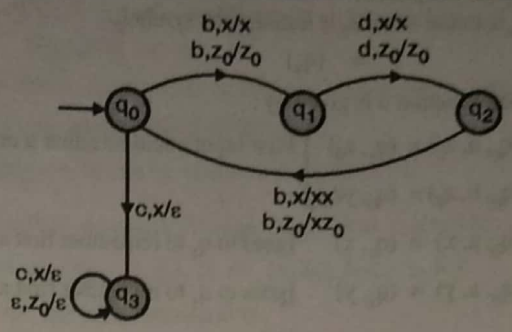


Fig. Ex. 6.4.5

– A cycle through $q_0 \to q_1 \to q_2 \to q_0$ traces a group of bdb.

The PDA M = $\{Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi\}$

Where, $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{b, d, c\}$, $\Gamma = \{x, z_0\}$

q_0 is the initial state, z_0 is initial stack symbol.

The transition function δ is given by,

$\delta(q_0, b, z_0) = (q_1, z_0)$

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

$\delta(q_1, d, z_0) = (q_2, z_0)$

$\delta(q_1, d, x) = (q_2, x)$

$\delta(q_2, b, z_0) = (q_0, xz_0)$

$\delta(q_2, b, x) = (q_0, xx)$

$\delta(q_0, c, x) = (q_3, \varepsilon)$

$\delta(q_3, c, x) = (q_3, \varepsilon)$

$\delta(q_3, \varepsilon, z_0) = (q_3, \varepsilon)$ Accept through empty stack.

Example 6.4.6

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. 6.4.6. Input applied to the machine is abaabbaaaaabaab.

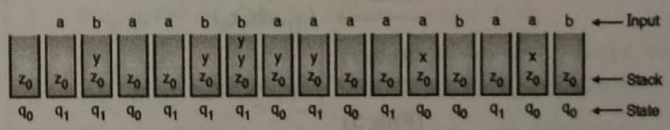


Fig. Ex. 6.4.6 : Status of stack for input abaabbaaaaabaab

Implementation of PDA using final state :

Let the PDA, $M = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$

Where, $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{x, y, z_0\}$

Scanned by CamScanner

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 6.4.7

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]
$\delta(q_0, b, a) = (q_1, \varepsilon)$]
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$ erase, everything from the stack
$\delta(q_1, \varepsilon, a) = (q_1, \varepsilon)$]
$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$

Example 6.4.8

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]

Scanned by CamScanner

$\delta(q_1, c, z_0) = (q_2, cz_0)$ [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 6.4.9

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 6.4.10 MU - May 2014, 10 Marks

Construct a PDA accepting $\{ a^n b^m a^n \mid m, n \geq 1 \}$ by null store.

Solution :

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]

Example 6.4.11

Design a PDA for accepting a language

$L = \{ WcW^T \mid W \in \{a, b\}^* \}$

Solution : W^T stands for reverse of W. A string of the form WcW^T is an odd length palindrome with the middle character as c.

Algorithm

If the length of the string is 2n + 1, then the first n symbols should be matched with the last n symbols in the reverse order. A stack can be used to reverse the first n input symbols.

– Status of the stack and state of the machine is shown in Fig. Ex. 6.4.11. Input applied is abbcbba.

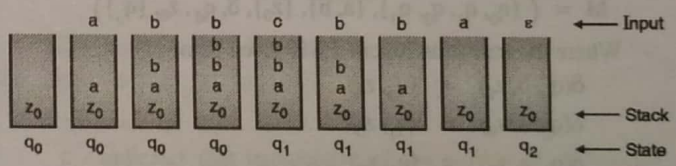


Fig. Ex. 6.4.11 : A PDA on input abbcbba

The PDA accepting through final state is given by

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_2\})$$

here the transition function δ is given below :

1. $\delta(q_0, a, \varepsilon) = (q_0, a)$
2. $\delta(q_0, b, \varepsilon) = (q_0, b)$

First n symbols are pushed onto the stack

3. $\delta(q_0, c, \varepsilon) = (q_1, \varepsilon)$ [State changes on c]
4. $\delta(q_1, a, a) = (q_1, \varepsilon)$
5. $\delta(q_1, b, b) = (q_1, \varepsilon)$

Last n symbols are matched with first n symbols in reverse order

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.

Scanned by CamScanner

Example 6.4.12

Design a PDA to check whether a given string over {a, b} ends in abb.

Solution : Strings ending in abb form a regular language. A regular language is accepted by DFA. A PDA for a regular language can be constructed in two steps :

1. Design a DFA.
2. Convert DFA to PDA by appending no-stack-operation to every transition in DFA.

Step 1 : DFA for the given language is :

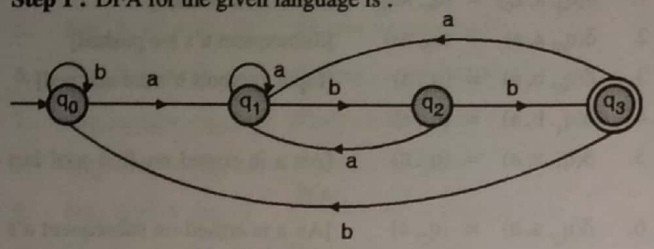


Fig. Ex. 6.4.12 : A DFA for string ending in abb

Step 2 : From DFA to PDA

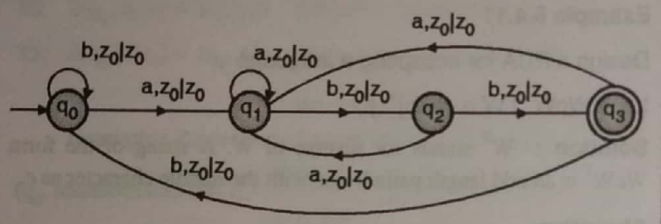


Fig. Ex. 6.4.12(a) : A PDA constructed from DFA

– In every transition of DFA, a stack operation z_0/z_0 is added. z_0/z_0 implies no-stack operation.

The PDA accepting strings ending in abb through final state is given by :

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{z_0\}, \delta, q_0, z_0, \{q_3\})$$

Where the transition function δ is given below,

$$\delta(q_0, b, z_0) = (q_0, z_0)$$
$$\delta(q_0, a, z_0) = (q_1, z_0)$$
$$\delta(q_1, a, z_0) = (q_1, z_0)$$
$$\delta(q_1, b, z_0) = (q_2, z_0)$$
$$\delta(q_2, a, z_0) = (q_1, z_0)$$
$$\delta(q_2, b, z_0) = (q_3, z_0)$$
$$\delta(q_3, a, z_0) = (q_1, z_0)$$
$$\delta(q_3, b, z_0) = (q_0, z_0)$$

Example 6.4.13 MU - Dec. 2016, 10 Marks

Construct deterministic PDA to recognize $a^n abb^n$, n > 0 over {a, b}

Ans. :

The smallest string of the above language will be a^2b^2. The PDA can be constructed by pushing leading is on the stack and then matching than with the trailing b's.

The transitions rules for the PDA :

$\delta(q_0, a, z_0) = (q_1, az_0)$ [First 'a' is pushed]

$\delta(q_1, a, a) = (q_2, aa)$ [Second 'a' is pushed]

$\delta(q_2, a, a) = (q_2, aa)$ [Subsequent a's are pushed]

$\delta(q_2, b, a) = (q_3, \varepsilon)$ [First 'b' is matched]

$\delta(q_3, b, a) = (q_3, \varepsilon)$ [Subsequent b's are matched]

$\delta(q_3, \varepsilon, z_0) = (q_3, \varepsilon)$

↓

Accept through as empty stack

The PDA is given by,

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{a_1, z_0\}, \$, q_0, z_0, \phi)$$

Example 6.4.14 MU - May 2017, 10 Marks

Design a PDA to accept the language

$\{L = a^m b^m c^n \mid m, n, > = 1\}$

Solution :

The transitions for PDA are :

1. $\delta(q_0, a, z_0) = (q_1, az_0)$
2. $\delta(q_1, a, a) = (q_1, aa)$
3. $\delta(q_1, b, a) = (q_2, \in)$
4. $\delta(q_2, b, a) = (q_2, \in)$
5. $\delta(q_2, c, z_0) = \delta(q_3, z_0)$
6. $\delta(q_3, c, z_0) = \delta(q_3, z_0)$
7. $\delta(q_3, \in, z_0) = \delta(q_3, \in)$

Accept through the empty stack.

Example 6.4.15

Design a PDA for CFL that checks the well formedness of parenthesis i.e. the language L of all "balanced" string of two types of parenthesis say "( )" and "[ ]". Trace the sequence of moves made corresponding to input string (([ ])[ ]).

Solution :

The transition function of the PDA is given below :

1. $\delta(q_0, (, z_0) = (q_0, (z_0)$
2. $\delta(q_0, (, ( ) = (q_0, (( )$
3. $\delta(q_0, (, [ ) = (q_0, ([ )$

Push the opening bracket '('

4. $\delta(q_0, [, z_0) = (q_0, [z_0)$
5. $\delta(q_0, [, ( ) = (q_0, [( )$
6. $\delta(q_0, [, [ ) = (q_0, [[ )$

Push the opening bracket '['

7. $\delta(q_0,),)) = (q_0, \varepsilon)$
8. $\delta(q_0,],]) = (q_0, \varepsilon)$

POP an opening bracket for a closing bracket.

9. $\delta(q_0, \varepsilon, z_0) = (q_f, z_0)$] Accept through a final state.

Simulation of PDA for the input string (([])[])

$$(q_0, (([])[]), z_0) \xrightarrow{\text{Rule 1}} (q_0, ([])[]), (z_0)$$
$$\xrightarrow{\text{Rule 2}} (q_0, [])[]), ((z_0)$$
$$\xrightarrow{\text{Rule 5}} (q_0,])[]), [((z_0)$$
$$\xrightarrow{\text{Rule 8}} (q_0,)[]), ((z_0)$$

Scanned by CamScanner

$\xrightarrow{\text{Rule 7}} (q_0, []), (z_0)$

$\xrightarrow{\text{Rule 5}} (q_0,]), [(z_0)$

$\xrightarrow{\text{Rule 8}} (q_0,), (z_0)$

$\xrightarrow{\text{Rule 7}} (q_0, \varepsilon, z_0)$

$\xrightarrow{\text{Rule 9}} (q_f, \varepsilon, z_0)$

6.5 Non-deterministic PDA (NPDA)

There are two types of push down automata :

1. DPDA (Deterministic PDA)
2. NPDA (Non-deterministic PDA)

A NPDA provides non-determinism to PDA.

In a DPDA there is only one move in every situation. Where as, in case of NPDA there could be multiple moves under a situation.

- DPDA is less powerful than NPDA. Every context free language can not be recognized by a DPDA but it can be recognized by NPDA. The class of language a DPDA can accept lies in between a regular language and CFL. A palindrome can be accepted by NPDA but it can not be accepted by a DPDA.

Example 6.5.1

Design a PDA for detection of odd palindrome over {a, b}.

Solution : An odd palindrome will be of the form

1. waw^R

$$\underbrace{ab}_{w}\ a\ \underbrace{ba}_{w^R},\ \underbrace{a\,b\,a}_{w}\ a\ \underbrace{a\,b\,a}_{w^R},\ \underbrace{aa}_{w}\ a\ \underbrace{aa}_{w^R}$$

2. wbw^R

$$\underbrace{ab}_{w}\ b\ \underbrace{ba}_{w^R},\ \underbrace{a\,b\,a}_{w}\ b\ \underbrace{a\,b\,a}_{w^R},\ \underbrace{aa}_{w}\ b\ \underbrace{aa}_{w^R}$$

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. 6.5.1. Input applied is ababa.

- Left child → current input is taken as the middle character
- Right child → current input is not a middle character.

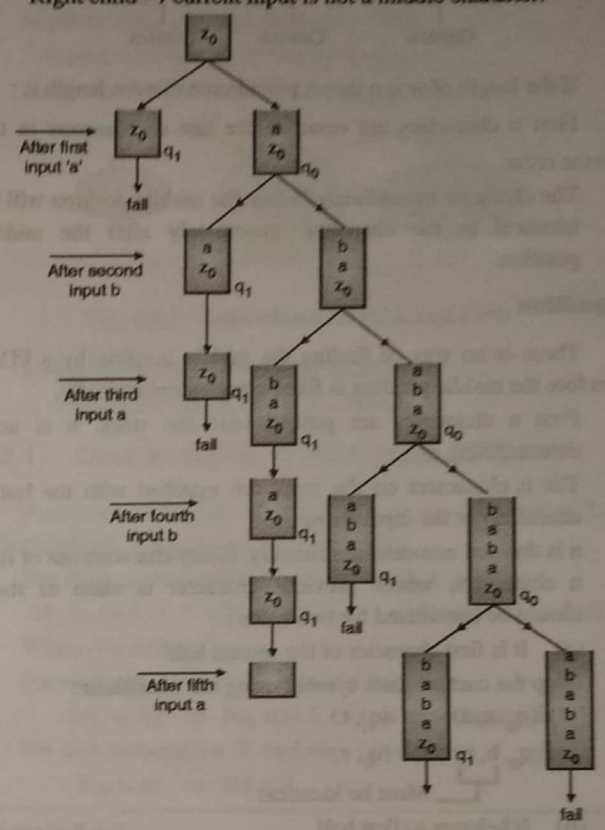


Fig. Ex. 6.5.1 : Processing of string by the PDA. String is taken as "ababa"

The transition table for the PDA is given below,

$\delta(q_0, a, \varepsilon) \Rightarrow \{(q_1, \varepsilon), (q_0, a)\}$

↓

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 Σ = {a, b}

The set of stack symbols Γ = $\{a, b, z_0\}$

Starting state = q_0

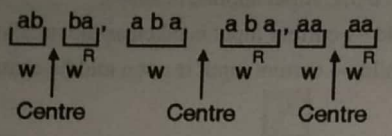
Initial stack symbol = z_0

Scanned by CamScanner

Example 6.5.2 MU- May 2015, 10 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 :

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 6.5.3

Design a PDA for detection of palindromes over {a, b}.

Solution :

A palindrome will be of the form :

1. ww^R – even palindrome
2. waw^R
3. wbw^R – odd palindrome

If the length of w is n then a palindrome is :

First n characters are equal to the last n characters in the reverse order with the middle character as :

(1) a [For odd palindrome]

(2) b [For odd palindrome]

(3) ε [For even palindrome]

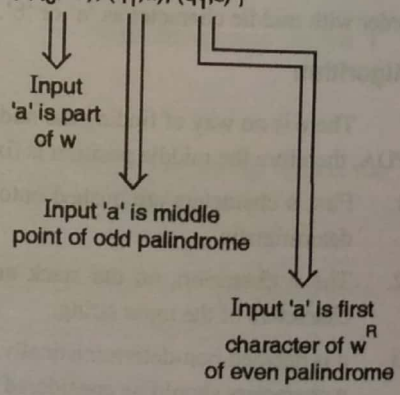
The transition table for the PDA is given below :

$\delta(q_0, a, z_0) \Rightarrow \{(q_1, z_0), (q_0, az_0)\}$
$\delta(q_0, b, z_0) \Rightarrow \{(q_1, z_0), (q_0, bz_0)\}$
$\delta(q_0, a, a) \Rightarrow \{(q_0, aa)\ (q_1, a), (q_1, \varepsilon)\}$
$\delta(q_0, a, b) \Rightarrow \{(q_0, ab), (q_1, b)\}$
$\delta(q_0, b, a) \Rightarrow \{(q_0, ba), (q_1, a)\}$
$\delta(q_0, b, b) \Rightarrow \{(q_0, bb), (q_1, b), (q_1, \varepsilon)\}$
$\delta(q_1, a, a) \Rightarrow \{(q_1, \varepsilon)\}$
$\delta(q_1, b, b) \Rightarrow \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, z_0) \Rightarrow \{(q_1, \varepsilon)\}$

[Accept through an empty stack].

Details of important transitions

The transaction, $\delta(q_0,a,a) \Rightarrow \{ (q_0,aa), (q_1,a), (q_1,\varepsilon) \}$

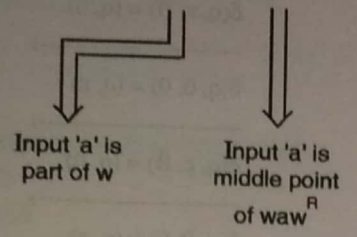


The transition rule for $\delta(q_0, a, a)$, must consider the three cases :

1. Input 'a' is part of w of the palindrome.
2. Input 'a' is middle character of waw^R
3. Input 'a' is the first character of w^R.

Scanned by CamScanner

The transaction, $\delta(q_0, a, b) \Rightarrow \{ (q_0, ab), (q_1, b) \}$



Example 6.5.4

Construct PDA for the language

$L = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

Solution :

Algorithm

Machine has to non-deterministically figure out which of the two conditions will be satisfied by the string :

1. $i \neq j$, by entering the state q_1 on first input.
2. $j \neq k$, by entering the state q_2 on first input.

$$\delta(q_0, a, z_0) = \{(q_1, aa), (q_2, z_0)\}$$

In state q_1

1. Initial a's will be pushed.
2. For every input symbol b, an 'a' should be erased from the stack.
3. If a's and b's are not equal i.e. either some a's are left on the stack or some b's are still in the input then the string should be accepted.

In state q_2

1. Initial a's will be skipped.
2. For every input symbol b, a 'b' should be pushed.
3. For every input symbol c, a 'ab' should be erased from the stack.
4. If b's and c's are not equal i.e. either some b's are left on the stack or some c's are still in the input then the string should be accepted.

Transitions for PDA

$$\delta(q_0, a, z_0) = \{(q_1, aa), (q_2, z_0)\}$$
$$\delta(q_1, a, a) = \{(q_1, aa)\}$$
$$\delta(q_1, b, a) = \{(q_3, \varepsilon)\}$$
$$\delta(q_3, b, z_0) = \{(q_f, z_0)\}$$
$$\delta(q_3, c, a) = \{(q_f, \varepsilon)\}$$
$$\delta(q_3, b, a) = \{(q_3, \varepsilon)\}$$
$$\delta(q_2, a, z_0) = \{(q_2, \varepsilon)\}$$
$$\delta(q_2, b, z_0) = \{(q_4, bz_0)\}$$
$$\delta(q_4, b, b) = \{(q_4, bb)\}$$
$$\delta(q_4, c, b) = \{(q_4, \varepsilon)\}$$
$$\delta(q_4, \varepsilon, b) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, b, \varepsilon) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, c, \varepsilon) = \{(q_f, \varepsilon)\}$$
$$\delta(q_f, \varepsilon, \varepsilon) = \{(q_f, \varepsilon)\}$$

6.6 Push Down Automata and Context Free Language

The class of languages accepted by pushdown automata is exactly the class of context-free languages. The following three classes of languages are same :

1. Context Free Language defined by CFG.
2. Languages accepted by PDA by final state.
3. Languages accepted by PDA by empty stack.

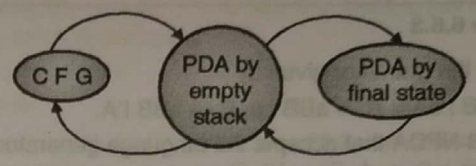


Fig. 6.6.1 : Equivalence of PDA and CFG

- It is possible to find a PDA for a CFG
- It is possible to find a CFG for a PDA.

6.6.1 Construction of PDA from CFG

From a given CFG G = (V, T, P, S), we can construct a PDA, M that simulates the leftmost derivation of G.

The PDA accepting L(G) by empty stack is given by :

$$M = (\{q\}, T, V \cup T, \delta, q, S, \phi) \quad \text{[M is a PDA for L(G)]}$$

Where δ is defined by :

1. For each variable $A \in V$, include a transition,
$$\delta(q, \varepsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in G}\}$$
2. For each terminal $a \in T$, include a transition
$$\delta(q, a, a) \Rightarrow \{(q, \varepsilon)\}$$

Example 6.6.1

Find a PDA for the given grammar $S \rightarrow 0S1 \mid 00 \mid 11$

Solution :

The equivalent PDA, M is given by :

$$M = (\{q\}, \{0, 1\}, \{0, 1, S\}, \delta, q, S, \phi),$$

where δ is given by :

$$\delta(q, \varepsilon, S) = \{(q, 0S1), (q, 00), (q, 11)\}$$
$$\delta(q, 0, 0) = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Example 6.6.2

Convert the grammar

$S \rightarrow 0S1 \mid A$

$A \rightarrow 1A0 \mid S \mid \varepsilon$

to PDA that accepts the same language by empty stack.

Solution :

Step 1 : for each variable $A \in V$, include a transition

$$\delta(q, \varepsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in G }\}$$
$$\therefore \delta(q, \varepsilon, S) \Rightarrow \{(q, 0S1), (q, A)\}$$
$$\delta(q, \varepsilon, A) \Rightarrow \{(q, 1A0), (q, S), (q, \varepsilon)\}$$

Step 2 : For each terminal $a \in T$, include a transition

$$\delta(q, a, a) \Rightarrow (q, \varepsilon)$$

Scanned by CamScanner

$\therefore \quad \delta(q, 0, 0) = \{(q, \varepsilon)\}$

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

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 6.6.3

Let G be the grammar given by

$S \rightarrow aABB \mid aAA,\ A \rightarrow aBB \mid a,\ B \rightarrow bBB \mid A.$

Construct NPDA that accepts the language generated by this grammar.

Solution :

The equivalent PDA, M is given by :

$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$

Where, δ is given by :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, aABB), (q, aAA)\}$

$\delta(q, \varepsilon, A) \Rightarrow \{(q, aBB), (q, a)\}$ — For each production in given grammar

$\delta(q, \varepsilon, B) \Rightarrow \{(q, bBB), (q, A)\}$

$\delta(q, a, a) \Rightarrow (q, \varepsilon)$

$\delta(q, b, b) \Rightarrow (q, \varepsilon)$ — For each terminal in T

Example 6.6.4

Construct a PDA equivalent to the following CFG.

$S \rightarrow 0BB$

$B \rightarrow 0S \mid 1S \mid 0$

Test if 010^4 is in the language

Solution : The equivalent PDA, M is given by

$M = (\{q\}, \{0, 1\}, \{0, 1, S, B\}, \delta, q, S, \phi)$,

where δ is given by

$\delta(q, \varepsilon, S) \Rightarrow \{(q, 0BB)\}$

$\delta(q, \varepsilon, B) \Rightarrow \{(q, 0S), (q, 1S), (q, 0)\}$ — 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 6.6.5

Design a PDA to recognize the language generated by the following grammar :

$S \rightarrow S + S \mid S * S \mid 4 \mid 2$

Show the acceptance of the input string $2 + 2 * 4$ by this PDA.

Solution : The equivalent PDA, M is given by :

$M = (\{q\}, \{+, *, 4, 2\}, \{+, *, 4, 2, S\}, \delta, q, S, \phi)$

Where δ is given by :

$\delta(q, \varepsilon, S) \Rightarrow \{(q, S + S), (q, S * S), (q, 4), (q, 2)\}$ | 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

$\delta(q, 2 + 2 * 4, S) \xrightarrow{\delta(q, \varepsilon S) \Rightarrow (q, S + S)} (q, 2 + 2 * 4, S + S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, 2)} (q, 2 + 2 * 4, 2 + S)$

$\xrightarrow{\delta(q, 2, 2) \Rightarrow (q, \varepsilon)} (q, + 2 * 4, + S)$

$\xrightarrow{\delta(q, +, +) \Rightarrow (q, \varepsilon)} (q, 2 * 4, S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, S * S)} (q, 2 * 4, S * S)$

$\xrightarrow{\delta(q, \varepsilon, S) \Rightarrow (q, 2)} (q, 2 * 4, 2 * S)$

$\xrightarrow{\delta(q, 2, 2) \Rightarrow (q, \varepsilon)} (q, * 4, * S)$

$\xrightarrow{\delta(q, *, *) \Rightarrow (q, \varepsilon)} (q, 4, S)$

$\xrightarrow{\delta(q, \varepsilon, S) = (q, 4)}$

Scanned by CamScanner

$$\xrightarrow{\delta(q, 4, 4) = (q, \varepsilon)} (q, 4, 4)$$

$$\xrightarrow{} (q, \varepsilon, \varepsilon)$$

Example 6.6.6

Convert the following expression grammar to PDA

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$E \rightarrow I \mid E * E \mid E * E \mid (E)$

Solution : The equivalent PDA, M is given by,

$M = (\{q\}, \{0, 1, a, b, *, +, (,)\}, \{0, 1, a, b, *, +, (,), I, E\}, \delta, q, E, \phi)$

where, δ is given by,

$$\begin{aligned}
\delta(q, \varepsilon, E) &= \{(q, I), (q, E * E), (q, E + E), (q, (E))\} \\
\delta(q, \varepsilon, I) &= \{(q, a), (q, b), (q, Ib), (q, Ia), (q, I0), (q, I1)\} \\
\delta(q, 0, 0) &= \{(q, \varepsilon)\} \\
\delta(q, 1, 1) &= \{(q, \varepsilon)\} \\
\delta(q, a, a) &= \{(q, \varepsilon)\} \\
\delta(q, b, b) &= \{(q, \varepsilon)\} \\
\delta(q, +, +) &= \{(q, \varepsilon)\} \\
\delta(q, *, *) &= \{(q, \varepsilon)\} \\
\delta(q, (, () &= \{(q, \varepsilon)\} \\
\delta(q,),)) &= \{(q, \varepsilon)\}
\end{aligned}$$

Example 6.6.7 MU - Dec. 2014, 10 Marks

Design a PDA corresponding to the grammar

$S \rightarrow aSa \mid bSb \mid \in$

Ans. : The equivalent PDA, M is given by :

$$M = (\{q\}, \{a,b\}, \{a,b,S\}, \delta, q, S, \phi)$$

Where δ is given by :

$$\begin{aligned}
\delta(q, \in, S) &= \{(q, aSa), (q, bSb), (q, \in)\} \\
\delta(q, a, a) &= (q, \in) \\
\delta(q, b, b) &= (q, \in)
\end{aligned}$$

Syllabus Topic : Equivalence between PDA and CFG

Example 6.6.8

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. 6.6.8.

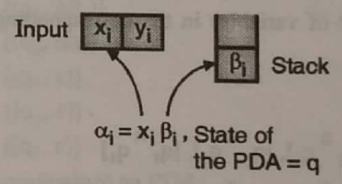


Fig. Ex. 6.6.8 : Unique configuration of PDA for α_i

– x_i portion of the input string w has already been scanned.

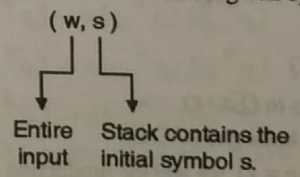
– The y_i is the portion of the input string, yet to be scanned.

– The stack contains β_i

– The theorem can be proved by induction on i of α_i.

Base case : i = 0

$\alpha_0 = S$, the configuration of PDA is given by



Induction step

We have to show that the correspondence is preserved for j = i + 1 also, if the correspondence for α_i is assumed.

Without the loss of generality, we can assume that the given grammar is in GNF.

If $(i + 1)_{th}$ 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

Scanned by CamScanner

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 strings α_i and the configuration of PDA. Thus a string $w \in L(G)$ will be accepted by the PDA M.

6.6.2 Construction of CFG from PDA

We can find the Context Free Grammar G for any PDA, M such that

$$L(G) = L(M)$$

i.e., we can construct an equivalent CFG for a PDA.

The variables of the CFG, so constructed will be of the form :

[pXq], 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.

   $S \to [q_0{}^z q_i]$ 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 :

   $$[q_i{}^B q] \to a\,[q_j{}^C q]$$
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

   $$[q_i{}^B p_1] \to a\,[q_j{}^{C_1} p_2]\,[p_2{}^{C_2} p_1]$$

Example 6.6.9

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.

$$S \to [q^z q]$$
$$S \to [q^z p]$$

Step 2 : Add productions for $\delta(q, 1, z) \Rightarrow \{(q, xz)\}$

$$[q^z q] \to 1\,[q^x q]\,[q^z q]$$
$$[q^z q] \to 1\,[q^x p]\,[p^z q]$$
$$[q^z p] \to 1\,[q^x q]\,[q^z p]$$
$$[q^z p] \to 1\,[q^x p]\,[p^z p]$$

Step 3 : Add productions For $\delta(q, 1, x) \Rightarrow \{(q, xx)\}$

$$[q^x q] \to 1\,[q^x q]\,[q^x q]$$
$$[q^x q] \to 1\,[q^x p]\,[p^x q]$$
$$[q^x p] \to 1\,[q^x q]\,[q^x p]$$
$$[q^x p] \to 1\,[q^x p]\,[p^x p]$$

Step 4 : Add productions for $\delta(q, \varepsilon, x) \Rightarrow \{(q, \varepsilon)\}$

$$[q^x q] \to \varepsilon$$

Step 5 : Add productions for $\delta(q, 0, x) \Rightarrow \{(p, x)\}$

$$[q^x q] \to 0\,[p^x q]$$
$$[q^x p] \to 0\,[p^x p]$$

Step 6 : Add productions for $\delta(p, 1, x) \Rightarrow \{(p, \varepsilon)\}$

$$[p^x p] \to 1$$

Step 7 : Add productions for $\delta(p, 0, z) \Rightarrow \{(q, z)\}$

$$[p^z q] \to 0\,[q^z q]$$
$$[p^z p] \to 0\,[q^z p]$$

Step 8 : Renaming of variables :

Original name	New name
$[q^z q]$	A
$[q^z p]$	B
$[p^z q]$	C
$[p^z p]$	D
$[q^x q]$	E
$[q^x p]$	F

Scanned by CamScanner

Original name	New name
$[p\ {}^{x}\ q]$	G
$[p\ {}^{x}\ p]$	H

The set of productions can be written as :

$S \to A \mid B$

$A \to 1EA \mid 1FC$

$B \to 1EB \mid 1FD$

$E \to IEE \mid 1FG$

$F \to 1EF \mid 1FH$

$E \to \varepsilon$

$E \to 0G$

$F \to 0H$

$H \to 1$

$C \to 0A$

$D \to 0B$

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 :

$S \to A \mid B$

$A \to 1EA \mid 1FC$

$B \to 1EB \mid 1FD$

$E \to 1EE \mid \varepsilon$

$F \to 1EF \mid 1FH \mid 0H$

$H \to 1$

$C \to 0A$

$D \to 0B$

Example 6.6.10

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

$S \to [q_0\ {}^{z_0}\ q_0]$

$S \to [q_0\ {}^{z_0}\ q_1]$

Step 2 : Add productions for $\delta(q_0, 1, z_0) = \{(q_0, xz_0)\}$

$[q_0\ {}^{z_0}\ q_0] \to 1\ [q_0\ {}^{x}\ q_0]\ [q_0\ {}^{z_0}\ q_0]$

$[q_0\ {}^{z_0}\ q_0] \to 1\ [q_0\ {}^{x}\ q_1]\ [q_1\ {}^{z_0}\ q_0]$

$[q_0\ {}^{z_0}\ q_1] \to 1\ [q_0\ {}^{x}\ q_0]\ [q_0\ {}^{z_0}\ q_1]$

$[q_0\ {}^{z_0}\ q_1] \to 1\ [q_0\ {}^{x}\ q_1]\ [q_1\ {}^{z_0}\ q_1]$

Step 3 : Add productions for $\delta(q_0, 1, x) \Rightarrow \{(q_0, xx)\}$

$[q_0\ {}^{x}\ q_0] \to 1\ [q_0\ {}^{x}\ q_0]\ [q_0\ {}^{x}\ q_0]$

$[q_0\ {}^{x}\ q_0] \to 1\ [q_0\ {}^{x}\ q_1]\ [q_1\ {}^{x}\ q_0]$

$[q_0\ {}^{x}\ q_1] \to 1\ [q_0\ {}^{x}\ q_0]\ [q_0\ {}^{x}\ q_1]$

$[q_0\ {}^{x}\ q_1] \to 1\ [q_0\ {}^{x}\ q_1]\ [q_1\ {}^{x}\ q_1]$

Step 4 : Add productions for $\delta(q_0, 0, x) \Rightarrow \{(q_1, x)\}$

$[q_0\ {}^{x}\ q_0] \to 0\ [q_1\ {}^{x}\ q_0]$

$[q_0\ {}^{x}\ q_1] \to 0\ [q_1\ {}^{x}\ q_1]$

Step 5 : Add productions for $\delta(q_0, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

$[q_0\ {}^{z_0}\ q_1] \to \varepsilon$

Step 6 : Add production for $\delta(q_1, 1, x) \Rightarrow \{(q_1, \varepsilon)\}$

$[q_1\ {}^{x}\ q_1] \to 1$

Step 7 : Add productions for $\delta(q_1, 0, z_0) \Rightarrow \{(q_0, z_0)\}$

$[q_1\ {}^{z_0}\ q_0] \Rightarrow 0\ [q_0\ {}^{z_0}\ q_0]$

$[q_1\ {}^{z_0}\ q_1] \Rightarrow 0\ [q_0\ {}^{z_0}\ q_1]$

Example 6.6.11

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.

$S \to [q_0\ {}^{z_0}\ q_0]$

$S \to [q_0\ {}^{z_0}\ q_1]$

Step 2 : Add productions for $(q_0, a, a) = \{(q_0, aa)\}$

$[q_0\ {}^{a}\ q_0] \to a\ [q_0\ {}^{a}\ q_0]\ [q_0\ {}^{a}\ q_0]$

$[q_0\ {}^{a}\ q_0] \to a\ [q_0\ {}^{a}\ q_1]\ [q_1\ {}^{a}\ q_0]$

$[q_0\ {}^{a}\ q_1] \to a\ [q_0\ {}^{a}\ q_0]\ [q_0\ {}^{a}\ q_1]$

$[q_0\ {}^{a}\ q_1] \to a\ [q_0\ {}^{a}\ q_1]\ [q_1\ {}^{a}\ q_1]$

Step 3 : Add productions for $\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$

$[q_0\ {}^{a}\ q_1] \to b$

Step 4 : Add productions for $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$

$[q_1\ {}^{a}\ q_1] \to b$

Step 5 : Add productions for $\delta(q_1, \varepsilon, z_0) \to \{(q_1, \varepsilon)\}$

$[q_1\ {}^{z_0}\ q_1] \to \varepsilon$

Example 6.6.12

For the PDA $(\{q_0, q_1\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \phi)$ where δ is

$\delta(q_0, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

$\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$

Scanned by CamScanner

$\delta(q_0, 0, 0) = \{(q_0, 00)\}$
$\delta(q_0, 1, 0) = \{(q_0, 10)\}$
$\delta(q_0, 1, 1) = \{(q_0, 11)\}$
$\delta(q_0, 0, 1) = \{(q_1, \varepsilon)\}$
$\delta(q_1, 0, 1) = \{(q_1, \varepsilon)\}$
$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, z_0) = \{(q_1, \varepsilon)\}$

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.	$S \to [q_0\ {}^{z_0}\ q_0]$
		$S \to [q_0\ {}^{z_0}\ q_1]$
2.	$\delta(q_0, \varepsilon, z_0) = (q_1, \varepsilon)$	$[q_0\ {}^{z_0}\ q_1] \to \varepsilon$
3.	$\delta(q_0, 0, z_0) = (q_0, 0z_0)$	$[q_0\ {}^{z_0}\ q_0] \to 0\ [q_0\ {}^{0}\ q_0]\ [q_0\ {}^{z_0}\ q_0]$
		$[q_0\ {}^{z_0}\ q_0] \to 0\ [q_0\ {}^{0}\ q_1]\ [q_1\ {}^{z_0}\ q_0]$
		$[q_0\ {}^{z_0}\ q_1] \to 0\ [q_0\ {}^{0}\ q_0]\ [q_0\ {}^{z_0}\ q_1]$
		$[q_0\ {}^{z_0}\ q_1] \to 0\ [q_0\ {}^{0}\ q_1]\ [q_1\ {}^{z_0}\ q_1]$
4.	$\delta(q_0, 0, 0) = (q_0, 00)$	$[q_0\ {}^{0}\ q_0] \to 0\ [q_0\ {}^{0}\ q_0]\ [q_0\ {}^{0}\ q_0]$
		$[q_0\ {}^{0}\ q_0] \to 0\ [q_0\ {}^{0}\ q_1]\ [q_1\ {}^{0}\ q_0]$
		$[q_0\ {}^{0}\ q_1] \to 0\ [q_0\ {}^{0}\ q_0]\ [q_0\ {}^{0}\ q_1]$
		$[q_0\ {}^{0}\ q_1] \to 0\ [q_0\ {}^{0}\ q_1]\ [q_1\ {}^{0}\ q_1]$
5.	$\delta(q_0, 1, 0) = (q_0, 10)$	$[q_0\ {}^{0}\ q_0] \to 1\ [q_0\ {}^{1}\ q_0]\ [q_0\ {}^{0}\ q_0]$
		$[q_0\ {}^{0}\ q_0] \to 1\ [q_0\ {}^{1}\ q_1]\ [q_1\ {}^{0}\ q_0]$
		$[q_0\ {}^{0}\ q_1] \to 1\ [q_0\ {}^{1}\ q_0]\ [q_0\ {}^{0}\ q_1]$
		$[q_0\ {}^{0}\ q_1] \to 1\ [q_0\ {}^{1}\ q_1]\ [q_1\ {}^{0}\ q_1]$
6.	$\delta(q_0, 1, 1) = (q_0, 11)$	$[q_0\ {}^{1}\ q_0] \to 1\ [q_0\ {}^{1}\ q_0]\ [q_0\ {}^{1}\ q_0]$
		$[q_0\ {}^{1}\ q_0] \to 1\ [q_0\ {}^{1}\ q_1]\ [q_1\ {}^{1}\ q_0]$
		$[q_0\ {}^{1}\ q_1] \to 1\ [q_0\ {}^{1}\ q_0]\ [q_0\ {}^{1}\ q_1]$
		$[q_0\ {}^{1}\ q_1] \to 1\ [q_0\ {}^{1}\ q_1]\ [q_1\ {}^{1}\ q_1]$
7.	$\delta(q_0, 0, 1) = (q_1, \varepsilon)$	$[q_0\ {}^{1}\ q_1] \to 0$
8.	$\delta(q_1, 0, 1) = (q_1, \varepsilon)$	$[q_1\ {}^{1}\ q_1] \to 0$
9.	$\delta(q_1, 0, 0) = (q_1, \varepsilon)$	$[q_1\ {}^{0}\ q_1] \to 0$
10.	$\delta(q_1, \varepsilon, z_0) = (q_1, \varepsilon)$	$[q_1\ {}^{z_0}\ q_1] \to \varepsilon$

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 :

$S \to A \mid B$
$B \to \varepsilon$
$A \to 0EA \mid 0FC$
$B \to 0EB \mid 0FD$
$E \to 0EE \mid 0FG$
$F \to 0EF \mid 0FH$
$E \to 1IE \mid 1JG$
$F \to 1IF \mid 1JH$
$I \to 1II \mid 1JK$
$J \to 1IJ \mid 1JL$
$J \to 0$
$L \to 0$
$H \to 0$
$D \to \varepsilon$

1. Removing ε-productions :

Nullable set = {D, B, S}

ε-productions are removed with resulting set of productions as given below :

$S \to A \mid B$
$A \to 0EA \mid 0FC$
$B \to 0EB \mid 0FD \mid 0E \mid 0F$
$E \to 0EE \mid 0FG \mid 1IE \mid 1JG$
$F \to 0EF \mid 0FH \mid 1IF \mid 1JH$
$H \to 0$
$I \to 1II \mid 1JK$
$J \to 1IJ \mid 1JL \mid 0$
$L \to 0$

2. Removing non-generating symbols

Set of productions after elimination of non-generating symbols {A, C, D, E, G, I} is given below :

$S \to B$
$B \to 0F$
$F \to 0FH \mid 1JH$
$H \to 0$
$J \to 1JL \mid 0$
$L \to 0$

Scanned by CamScanner

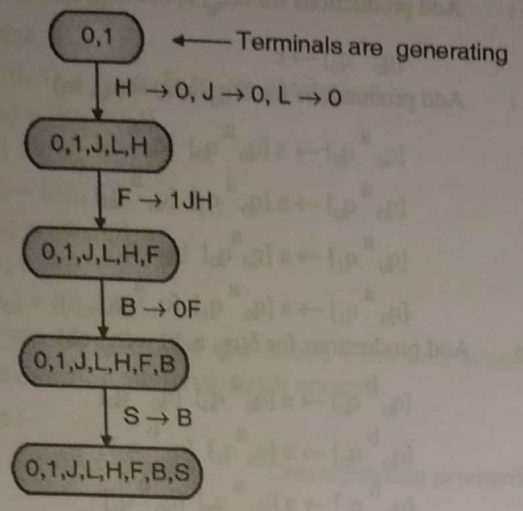


Fig. Ex. 6.6.12

3. The unit production $S \to B$ should be removed. The set of productions after elimination of the unit production $S \to B$ is given below :

$S \to 0F$

$F \to 0FH \mid 1JH$

$H \to 0$

$J \to 1JL \mid 0$

$L \to 0$

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 6.6.13

Construct a PDA accepting $\{a^n b^m a^n \mid m, n > 1\}$ by null store. From the PDA construct the corresponding CFG.

Solution :

Algorithm

1. 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. Sequence of input b's should be skipped. These b's will have no effect on the stack.

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

3. Initial a's which are on the stack should be matched with the trailing a's in the input. An 'a' should be popped for every 'a' as input till the end of input.

$$\delta(q_1,a,a) = (q_2, \varepsilon)$$
$$\delta(q_2,a,a) = (q_2, \varepsilon)$$

4. Finally, the symbol z_0 should be popped but to make the stack empty.

$$\delta(q_2, \varepsilon, z_0) = (q_2, \varepsilon)$$

Transition diagram and the transition table are given below.

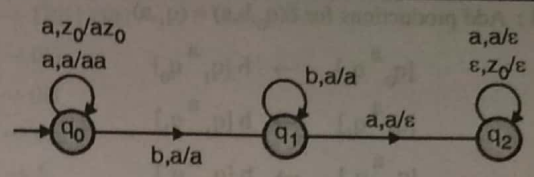


Fig. Ex. 6.6.13 : Transition diagram

Transition table

$$\delta(q_0,a,z_0) = (q_0,a,z_0)$$
$$\delta(q_0,a,a) = (q_0,aa)$$
$$\delta(q_0,b,a) = (q_1,a)$$
$$\delta(q_1,b,a) = (q_1,a)$$
$$\delta(q_1,a,a) = (q_2, \varepsilon)$$
$$\delta(q_2,a,a) = (q_2, \varepsilon)$$
$$\delta(q_2, \varepsilon, z_0) = (q_2, \varepsilon)$$
$$\text{The PDA M} = (\{q_0,q_1,q_2\},\{a,b\},\{a,b,z_0\},\delta,q_0,z_0,\phi)$$

PDA to CFG

Step 1 : Add productions for the start symbol

$$S \to [q_0 \; {}^{z_0} \; q_0]$$
$$S \to [q_0 \; {}^{z_0} \; q_1]$$
$$S \to [q_0 \; {}^{z_0} \; q_2]$$

Step 2 : Add productions for $\delta(q_0,a,z_0) = (q_0,az_0)$

$$[q_0 \; {}^{z_0} \; q_0] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{z_0} \; q_0]$$
$$[q_0 \; {}^{z_0} \; q_0] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{z_0} \; q_0]$$
$$[q_0 \; {}^{z_0} \; q_0] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{z_0} \; q_0]$$
$$[q_0 \; {}^{z_0} \; q_1] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{z_0} \; q_1]$$
$$[q_0 \; {}^{z_0} \; q_1] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{z_0} \; q_1]$$
$$[q_0 \; {}^{z_0} \; q_1] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{z_0} \; q_1]$$
$$[q_0 \; {}^{z_0} \; q_2] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{z_0} \; q_2]$$
$$[q_0 \; {}^{z_0} \; q_2] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{z_0} \; q_2]$$
$$[q_0 \; {}^{z_0} \; q_2] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{z_0} \; q_2]$$

Step 3 : Add productions for $\delta(q_0,a,a) = (q_0,aa)$

$$[q_0 \; {}^{a} \; q_0] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{a} \; q_0]$$
$$[q_0 \; {}^{a} \; q_0] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{a} \; q_0]$$
$$[q_0 \; {}^{a} \; q_0] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{a} \; q_0]$$
$$[q_0 \; {}^{a} \; q_1] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{a} \; q_1]$$
$$[q_0 \; {}^{a} \; q_1] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{a} \; q_1]$$
$$[q_0 \; {}^{a} \; q_1] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{a} \; q_1]$$
$$[q_0 \; {}^{a} \; q_2] \to a\,[q_0 \; {}^{a} \; q_0]\;[q_0 \; {}^{a} \; q_2]$$
$$[q_0 \; {}^{a} \; q_2] \to a\,[q_0 \; {}^{a} \; q_1]\;[q_1 \; {}^{a} \; q_2]$$
$$[q_0 \; {}^{a} \; q_2] \to a\,[q_0 \; {}^{a} \; q_2]\;[q_2 \; {}^{a} \; q_2]$$

Scanned by CamScanner

Step 4 : Add productions for $\delta(q_0,b,a) = (q_1,a)$

$$[q_0\ {}^a\ q_0] \rightarrow b\,[q_1\ {}^a\ q_0]$$
$$[q_0\ {}^a\ q_1] \rightarrow b\,[q_1\ {}^a\ q_1]$$
$$[q_0\ {}^a\ q_2] \rightarrow b\,[q_1\ {}^a\ q_2]$$

Step 5 : Add productions for $\delta(q_1,b,a) = (q_1,a)$

$$[q_1\ {}^a\ q_0] \rightarrow b\,[q_1\ {}^a\ q_0]$$
$$[q_1\ {}^a\ q_1] \rightarrow b\,[q_1\ {}^a\ q_1]$$
$$[q_1\ {}^a\ q_2] \rightarrow b\,[q_1\ {}^a\ q_2]$$

Step 6 : Add productions for $\delta(q_1,a,a) = (q_2,\varepsilon)$

$$[q_1\ {}^a\ q_2] \rightarrow a$$

Step 7 : Add productions for $\delta(q_2,a,a) = (q_2,\varepsilon)$

$$[q_2\ {}^a\ q_2] \rightarrow a$$

Step 8 : Add productions for $\delta(q_2, \varepsilon,z_0) = (q_2,\varepsilon)$

$$[q_2\ {}^{z_0}\ q_2] \rightarrow \varepsilon$$

Example 6.6.14

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 $\delta(q_0,b,z) = (q_0,bz)$

$$[q_0\ {}^z\ q_0] \rightarrow b[q_0\ {}^b\ q_0]\,[q_0\ {}^z\ q_0]$$
$$[q_0\ {}^z\ q_0] \rightarrow b[q_0\ {}^b\ q_1]\,[q_1\ {}^z\ q_0]$$
$$[q_0\ {}^z\ q_1] \rightarrow b[q_0\ {}^b\ q_0]\,[q_0\ {}^z\ q_1]$$
$$[q_0\ {}^z\ q_1] \rightarrow b[q_0\ {}^b\ q_1]\,[q_1\ {}^z\ q_1]$$

Scanned by CamScanner

Example 6.6.15

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.	$S \to [q_0 \; {}^{z_0} \; q_0]$
		$S \to [q_0 \; {}^{z_0} \; q_1]$
2.	$\delta(q_0, 1, z_0) = (q_0, xz_0)$	$[q_0 \; {}^{z_0} \; q_0] \to 1 \, [q_0 \; {}^{x} \; q_0] \, [q_0 \; {}^{z_0} \; q_0]$
		$[q_0 \; {}^{z_0} \; q_0] \to 1 \, [q_0 \; {}^{x} \; q_1] \, [q_1 \; {}^{z_0} \; q_0]$
		$[q_0 \; {}^{z_0} \; q_1] \to 1 \, [q_0 \; {}^{x} \; q_0] \, [q_0 \; {}^{z_0} \; q_1]$
		$[q_0 \; {}^{z_0} \; q_1] \to 1 \, [q_0 \; {}^{x} \; q_1] \, [q_1 \; {}^{z_0} \; q_1]$
3.	$\delta(q_0, 1, x) = (q_0, xx)$	$[q_0 \; {}^{x} \; q_0] \to 1 \, [q_0 \; {}^{x} \; q_0] \, [q_0 \; {}^{x} \; q_0]$
		$[q_0 \; {}^{x} \; q_0] \to 1 \, [q_0 \; {}^{x} \; q_1] \, [q_1 \; {}^{x} \; q_0]$
		$[q_0 \; {}^{x} \; q_1] \to 1 \, [q_0 \; {}^{x} \; q_0] \, [q_0 \; {}^{x} \; q_1]$
		$[q_0 \; {}^{x} \; q_1] \to 1 \, [q_0 \; {}^{x} \; q_1] \, [q_1 \; {}^{x} \; q_1]$
4.	$\delta(q_0, 0, x) = (q_1, x)$	$[q_0 \; {}^{x} \; q_0] \to 0 [q_1 \; {}^{x} \; q_0]$
		$[q_0 \; {}^{x} \; q_1] \to 0 [q_1 \; {}^{x} \; q_1]$
5.	$\delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$	$[q_0 \; {}^{z_0} \; q_0] \to \varepsilon$
6.	$\delta(q_1, 1, x) = (q_1, \varepsilon)$	$[q_1 \; {}^{x} \; q_1] \to 1$
7.	$\delta(q_1, 0, z_0) = (q_1, z_0)$	$[q_1 \; {}^{z_0} \; q_0] \to 0 \, [q_0 \; {}^{z_0} \; q_0]$
		$[q_1 \; {}^{z_0} \; q_1] \to 0 \, [q_0 \; {}^{z_0} \; q_1]$

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 :

$S \to A \mid B$

$A \to 1EA \mid 1FC$

$B \to 1EB \mid 1FD$

$E \to 1EE \mid 1FG$

$F \to 1EF \mid 1FH$

$E \to 0G$

$F \to 0H$

$A \to \varepsilon$

$H \to 1$

$C \to 0A$

$D \to 0B$

1. Removing ε-production.

Nullable set = {S, A}

ε-productions are removed with resulting set of productions as given below :

$S \to A \mid B$

$A \to 1EA \mid 1FC \mid 1E$

$B \to 1EB \mid 1 FD$

$C \to 0A \mid 0$

$D \to 0B$

$E \to 1EE \mid 1 FG \mid 0G$

$F \to 1EH \mid 1FH \mid 0H$

$H \to 0$

2. Removing non-generating symbols

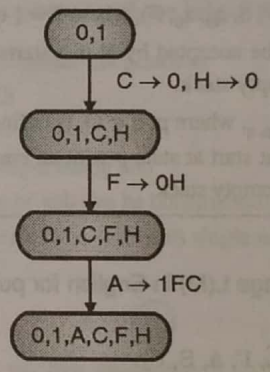


Fig. Ex. 6.6.15

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 :

$$L = \{1^n 0 1^n 0 \ldots 1^n 0 \mid n \geq 1\} \cup \varepsilon$$

Scanned by CamScanner

Example 6.6.16

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,

$$\delta(q_i, x, b) \Rightarrow (q_j, c) \text{ [pop b and push c]}$$

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,

$$N = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F), \text{ where } F = \{ q_f \}$$

- A word w will be accepted by N if it starts in q_0 and ends up in q_f with an empty stack.
- The language $L_{p,q}$, where $p, q \in Q$, is defined as consisting of those strings that start at state p with an empty stack and ends up in q with an empty stack.

Example 6.6.17

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 $(a + b)^m a(a + b)^n \mid n \le m$ and $n, m \ge 0$

- 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.

6.7 Deterministic Push Down Automata (DPDA)

In a DPDA there is only one move in every situation. A DPDA is less powerful than NPDA. Every context free language cannot be accepted by a DPDA. For example, a string of the form ww^R can not be processed by a DPDA. The class of a language a DPDA can accept lies in between a regular language and CFL.

A DPDA is defined as :

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F), \text{ where}$$

$\delta(q, a, x)$ has one move for any $q \in Q$, $X \in \Gamma$ and $a \in \Sigma$.

6.7.1 Regular Language and DPDA

We can always design a DPDA for a regular language. A DPDA can be designed for regular language in two steps :

Step 1 : Construct an equivalent DFA for the given regular language.

Step 2 : For every transition $\delta(q_i, a) = q_j$ in FA (where $q_i, q_j \in Q$ and $a \in \Sigma$), we can write an equivalent transition for DPDA.

$$\delta(q_i, a, z_0) = \{(q_j, z_0)\}.$$

The move for PDA involves neither a **Push** nor a **Pop** operation.

Example 6.7.1

Design a DPDA for a binary number divisible by 3.

Solution :

Step 1 : Construction of DFA.

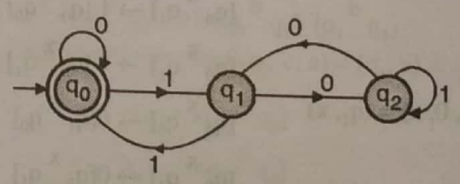


Fig. Ex. 6.7.1 : Construction of DFA

Step 2 : The DPDA is given by :

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, 1, z_0\}, \delta, q_0, z_0, \{q_0\})$$

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 6.7.2

Show that if L is accepted by a PDA in which no symbols are removed from the stack, then L is regular.

Solution : Every regular language is accepted by some FA. Every transition of an FA can be converted into a PDA move without any push or pop operation.

Scanned by CamScanner

Let there be a transition $\delta(q_i, a) = q_j$ in the FA. Where,

$q_i, q_j \in Q$ and $a \in \Sigma$.

The FA move $\delta(q_i, a) = q_j$ can be converted into an equivalent PDA move as given below :

$$\delta(q_i, a, z_0) = \{(q_j, z_0)\}$$

This move for the PDA involves neither a **Push** nor a **Pop** operation.

Example 6.7.3

Prove "Let L be a language accepted by deterministic PDA, then the complement of L, can also be accepted by a DPDA".

Solution : Let the DPDA for the given language L is

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

We can construct M′ from M, such that M′ accepts L′.

$$M' = (Q, \Sigma, \Gamma, \delta, q_0, z_0, Q\text{-}F)$$

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 6.7.4

Enlist the difference between PDM and FSM.

Solution :

1. PDM is more powerful than FSM.
2. PDM has additional memory in the form of a stack.
3. PDM can handle CFL but FSM can not handle CFL.

6.8 Application of PDA

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.

6.9 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.

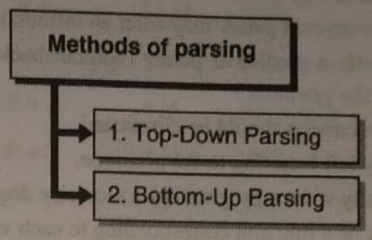


Fig. C6.2 : Methods of parsing

➜ 6.9.1 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

$S \rightarrow aXb$

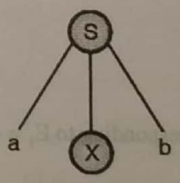
$X \rightarrow ab \mid b$

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



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 $X \rightarrow b$.

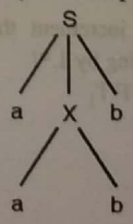


Fig. 6.9.1

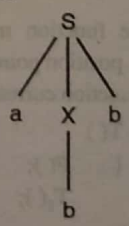


Fig. 6.9.2

Scanned by CamScanner

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. 6.9.2.

- 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 6.9.1

Write a program for recursive descent parser for the following grammar.

E = E + T | T

T = T * F | F

F = (E) | a | b

Solution :

Step 1 : Removing left recursion from the grammar we get :

$$E = TE_1$$
$$E_1 = +TE_1 \mid \varepsilon$$
$$T = FT_1$$
$$T_1 = *FT_1 \mid \varepsilon$$
$$F = (E) \mid a \mid b$$

Step 2 : The program will have a function for each variable.

Function corresponding to $E = TE_1$

```
E( )
{ T( );
  E1( );
}
```

Function corresponding to $E_1 = +TE_1 \mid \varepsilon$

```
E1( )
{     if (input [position] == '+')
      {   match( );
          T( );
          E1( );
      }
}
```

/*The function match ( ) will increment the current position pointer for input string by 1.*/

Function corresponding to $T = FT_1$

```
T( )
{     F( );
      T1( );
}
```

Function corresponding to $T_1 = *FT_1 \mid \varepsilon$

```
T1( )
{ if(input [position] == '*')
  { match( );
    F( );
    T1( );
  }
}
```

Function corresponding to F = (E) | a | 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");
    }
```

6.9.2 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. 6.9.3. 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.

Scanned by CamScanner

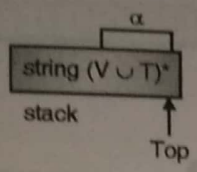


Fig. 6.9.3

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 6.9.2

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 :

	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

6.10 Exam Pack (University Questions)

Q. Give formal definition of a Push Down Automata (PDA). *(Refer section 6.2)* (5 Marks) **(May 2014, May 2015)**

Example 6.4.10 (10 Marks) **(May 2014)**

Example 6.4.13 (10 Marks) **(Dec. 2016)**

Example 6.4.14 (10 Marks) **(May 2017)**

Example 6.5.2 (10 Marks) **(May 2015)**

Example 6.6.7(10 Marks) **(Dec. 2014)**

❑❑❑

Scanned by CamScanner

CHAPTER 7

Turing Machine

Syllabus Topics

Turing Machine : Deterministic TM, Multi-track and Multi-tape TMs, concept of UTM and idea of system program. Issue and concept of Halting Problem

Syllabus Topic : Deterministic TM

7.1 Introduction to Turing Machine

➔ **(Dec. 2014)**

Q. Write short note on : Turing machines.

MU - Dec. 2014, 10 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 cannot 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.

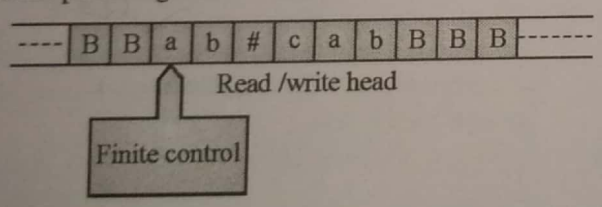


Fig. 7.1.1 : An example of a Turing Machine

An abstract model of a turing machine is shown in Fig. 7.1.1.

- Input to a turing machine is provided through a long tape.
- Turing machine is provided with a read / write head.
- The tape is divided into squares. Each square holds a single symbol.
- Blank squares hold a special character 'B'.
- The head is capable of performing three operations :
  1. Reading a symbol being scanned.
  2. Modifying a symbol being scanned.
  3. Shifting either to previous square (L) or next square (R).

Example 7.1.1

Design a turing machine to perform the following computation :

Initially the tape contains two finite blocks of 1's separated by finite block of blanks. The machine should delete the block of blanks between two blocks of 1's.

B	B	1	1	1	1	B	B	B	1	1	1	B

(a) Input tape

B	B	1	1	1	1	1	1	1	B

(b) Output tape

Fig. Ex. 7.1.1 : Input, Output tapes

Solution :

Let us call the first block of 1's as ω_1 and the second block of 1's as ω_2.

There are three blanks between ω_1 and ω_2. These three blanks will be deleted in three cycle.

Each cycle will consist of following steps :

Step 1 : Leftmost 1 of ω_1 is erased.

Step 2 : Head moves to first 'B' between ω_1 and ω_2 and replaces it by 1.

Step 3 : If some B's are yet to be deleted then head comes back to first 1 of ω_1.

These cycles of computation are shown in Fig. Ex. 7.1.1(c).

Scanned by CamScanner

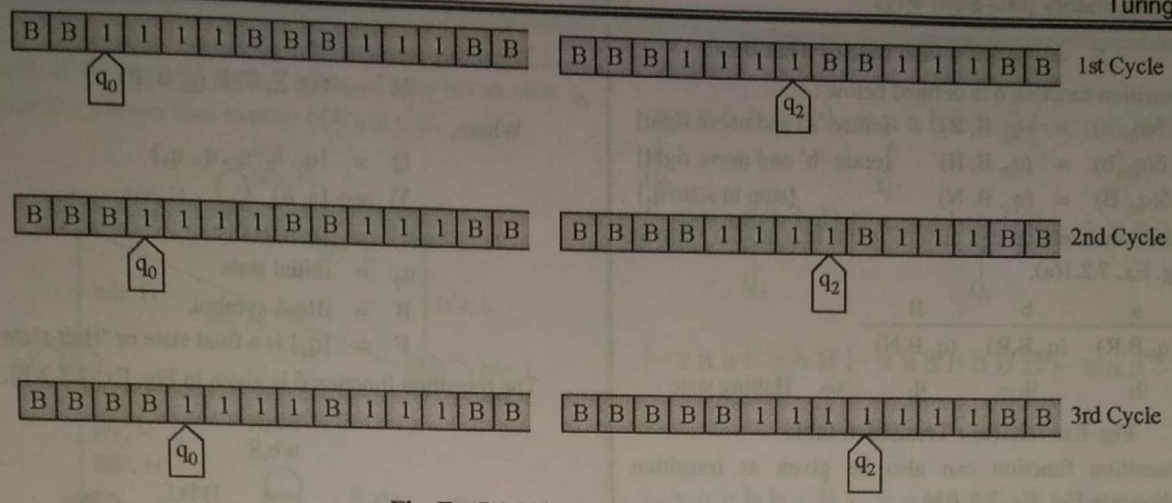


Fig. Ex. 7.1.1(c) : Cycles of computation

The transition behaviour of the turing machine is shown in Fig. Ex. 7.1.1(d).

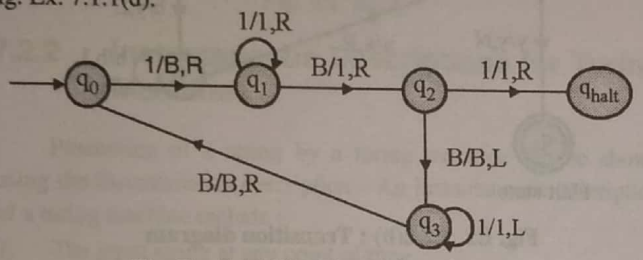
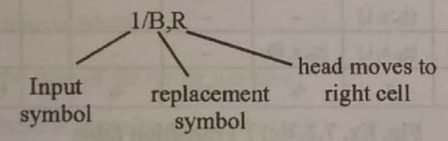


Fig. Ex. 7.1.1(d) : Transition diagram

Arc between q_0 and q_1 is marked as (1/B, R). It implies that the square being scanned contains 1 and it will be replaced by B, then the head will move to right cell.



Meaning of various transitions and their effect on input tape

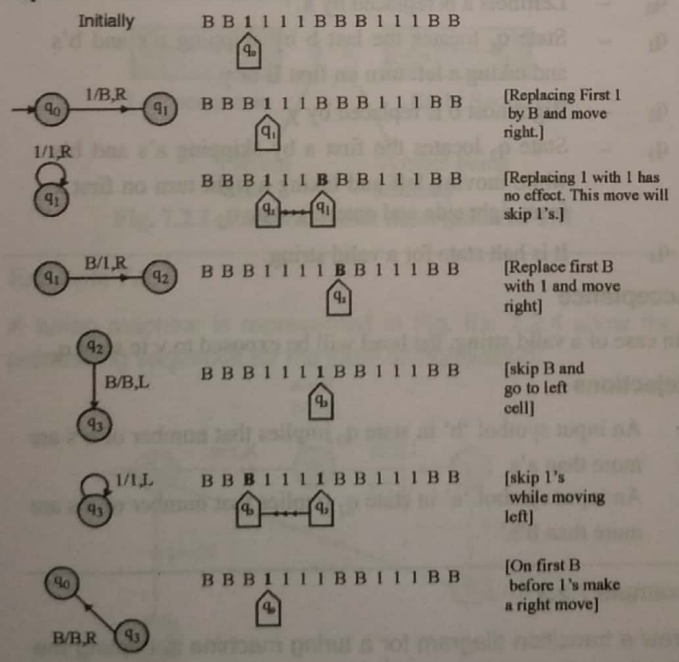


Fig. Ex. 7.1.1(e) : Transitions in detail for first cycle

7.2 The Formal Definition of Turing Machine

A Turing machine M is a 7-tuple given by :

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

where

1. Q is finite set of states
2. Σ is finite set of input alphabet not containing B.
3. Γ is a finite set of tape symbols. Tape symbols include B.
4. $q_0 \in Q$ is the initial symbol.
5. $B \in \Gamma$ is a special symbol representing an empty cell.
6. $F \subseteq Q$ is the set of final states, final states are also known as halting states.
7. The transition function δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times (L,R,N)$

A transition in turing machine is written as,

$\delta(q_0, a) = (q_1, b, R)$, which implies, when in state q_0 and scanning symbol a, the machine will enter state q_1, it will rewrite a as b and move to the right cell.

A transition $\delta(q_0, a) = (q_1, a, R)$, implies that the machine will enter state q_1, it will not change the symbol being scanned and move to the right cell.

Movement of Read / Write head is given L, R or N

L $\rightarrow$ Move to left cell

R $\rightarrow$ Move to right cell

N $\rightarrow$ Remain in the current cell (No movement)

Example 7.2.1

Design a turing machine that erases all non-blank symbols on the tape, where the sequence of non-blank symbols does not contain any blank symbol B in-between.

Solution : The turing machine M is given by :

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

where

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

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

$\Gamma = \{a, b, B\}$,

q_0 = is the initial state,

B = is blank symbol.

Scanned by CamScanner

$F = \{q_1\}$ 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. 7.2.1(a).

	a	b	B	
$\rightarrow q_0$	(q_0,B,R)	(q_0,B,R)	(q_1,B,N)	
q_1^*	q_1	q_1	q_1	← Halting state

Fig. Ex. 7.2.1(a) : Transition table

The transition function can also be given as transition diagram as shown in Fig. Ex. 7.2.1(b).

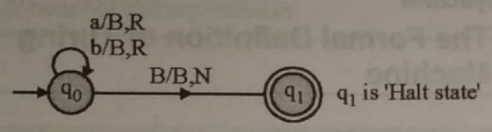


Fig. Ex. 7.2.1(b) : Transition diagram

7.2.1 A String Accepted by TM

A string α over Σ is said to be accepted by a turing machine $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F)$ if when the string α is placed on the tape and head is positioned on leftmost cell containing a non-blank cell, machine is started in state q_0, then after a finite number of moves the machine is in a "Halt-state" ∈ F.

$$(q_0,\omega) \overset{*}{\underset{M}{\vdash}} \text{Halt-state}$$

A string is rejected by a TM if the machine enters a state $q \notin F$ and scans a symbol x for which δ(q,x) is not defined.

Example 7.2.2

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. 7.2.2(a).

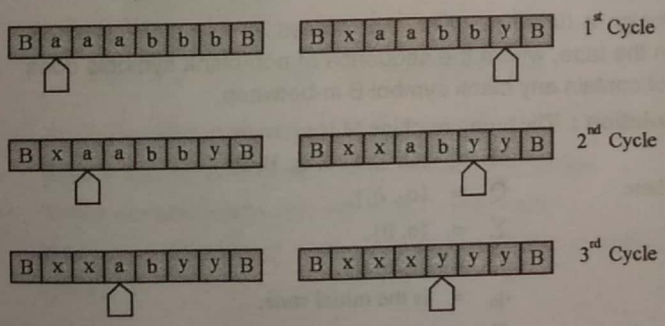


Fig. Ex. 7.2.2(a) : Cycles of computation for a^3b^3

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. 7.2.2(b).

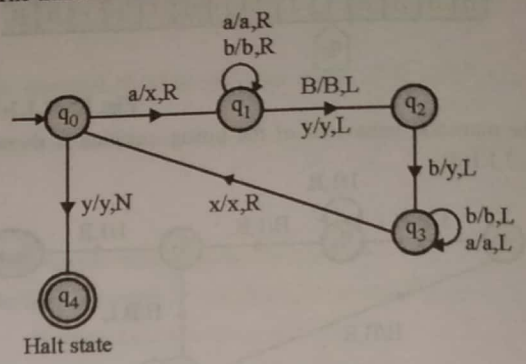


Fig. Ex. 7.2.2(b) : Transition diagram

	a	b	x	y	B		
$\rightarrow q_0$	(q_1, x, R)	–	–	(q_4, y, N)	–		
q_1	(q_1, a, R)	(q_1, b, R)	–	(q_2, y, L)	(q_2, B, L)		
q_2	–	(q_3, y, L)	–	–	–		
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, x, R)	–	–		
q_4^*	q_4	q_4	q_4	q_4	q_4	←	Halting state

Fig. Ex. 7.2.2(c) : Transition table

Meaning of Various States

q_0 – Leftmost a is replaced by x.

q_1 – State q_1 locates the last b by skipping a's and b's and taking a left turn on first B or y.

q_2 – Rightmost b is replaced by y.

q_3 – State q_3 locates the first a by skipping a's and b's while moving left and taking a right turn on first x from right side and entering state q_0.

q_4 – It is halt state for a valid string.

Acceptance

In case of a valid string, the head will be exposed to y in state q_0.

Rejections

1. An input symbol 'b' in state q_0 implies that number of b's are more than a's.
2. An input symbol 'a' in state q_2 implies that number of a's are more than b's.

Example 7.2.3

Draw a transition diagram for a turing machine accepting the following language. $L = \{ a^i b^j \mid i < j\}$

Scanned by CamScanner

Solution : Solution to this example follows from Ex. 7.2.2. In Fig. 7.2.2(b), if the machine finds a string of only b's in state q_0 then number of a's is less than number of b's in $a^i b^j$.

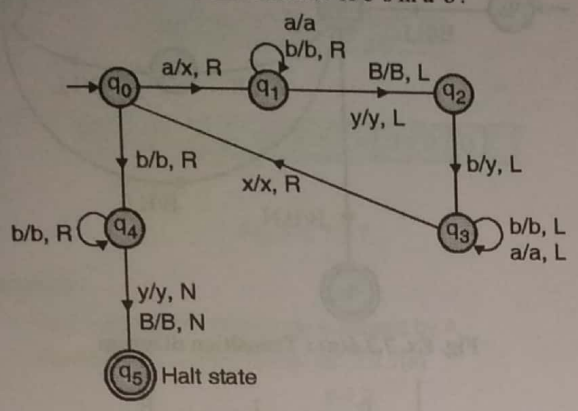


Fig. Ex. 7.2.3

7.2.2 Instantaneous Descriptions for Turing Machines

Processing of a string by a turing machine can be shown using the instantaneous description - An instantaneous description of a turing machine include :

1. The input string at any point of time
2. Position of head
3. State of the machine

The string $a_1 a_2 \ldots a_{i-1} \boxed{q\, a_i}\, a_{i+1} \ldots a_n$ 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. 7.2.1.

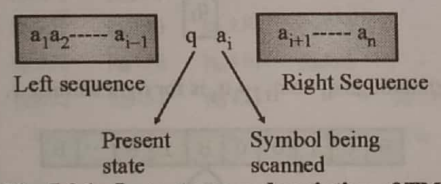


Fig. 7.2.1 : Instantaneous description of TM

Example 7.2.4

A turing machine is represented in Fig. Ex. 7.2.4 show the processing sequence for the input string aaabbb.

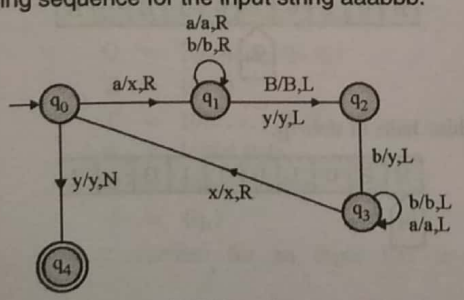


Fig. Ex. 7.2.4 : TM under consideration

Solution :

$\underset{q_0}{a}aabbbB \vdash x\underset{q_1}{a}abbbB \vdash xa\underset{q_1}{a}bbbB$

$\vdash xaa\underset{q_1}{b}bbB \vdash xaab\underset{q_1}{b}bB \vdash xaabb\underset{q_1}{b}B$

$\vdash xaabbb\underset{q_1}{B} \vdash xaabb\underset{q_2}{b}B \vdash xaab\underset{q_3}{b}yB$

$\vdash xaa\underset{q_3}{b}byB \vdash xaa\underset{q_3}{b}byB \vdash xa\underset{q_3}{a}abbyB$

$\vdash \underset{q_3}{x}aabb\ yB \vdash x\underset{q_0}{a}abbyB$

$\vdash xx\underset{q_1}{a}bb\ yB \vdash xxa\underset{q_1}{b}byB \vdash xxab\underset{q_1}{b}yB$

$\vdash xxabb\underset{q_1}{y}B \vdash x\ xab\underset{q_2}{b}yB \vdash xxa\underset{q_3}{b}yyB$

$\vdash xx\underset{q_3}{a}byyB \vdash x\underset{q_3}{x}abyyB \vdash xx\underset{q_0}{a}byyB$

$\vdash xxx\underset{q_1}{b}yyB \vdash xxxb\underset{q_1}{y}yB \vdash xxx\underset{q_2}{b}yyB$

$\vdash xxx\underset{q_3}{x}yyyB \vdash xxx\underset{q_0}{y}yyB \vdash xxx\underset{q_4}{y}yyB$

[Accept]

Example 7.2.5

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

Scanned by CamScanner

Solution :

B 1 1 0 B ⊢ B 0 1 0 B [As per (S_0, 1, 0,S_0 ,L)]

↑ ↑

S_0 S_0

⊢ B B 0 1 0 B [If there is less then 2 blanks on the left then the tape head halt with a crash]

↑

Halt

2) 0011B

B 0 0 1 1 B ⊢ B 0 0 1 1 B [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

B 0 1 0 1 B ⊢ B 0 0 1 1 B [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 7.2.6

Design a right shifting turing machine over alphabet {0,1}

Solution :

A right shifting turing machine will shift the input string, right by 1 place.

Algorithm

Right shifting should start from the rightmost character. Each character is shifted right starting from right end and working towards left end.

Cycles of computation for an input string 1010 are shown in Fig. Ex. 7.2.6 :

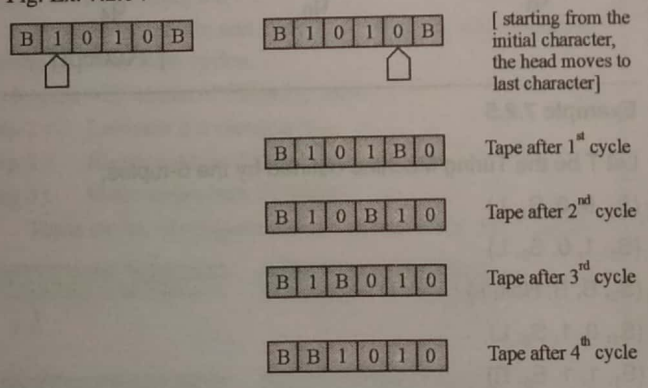


Fig. Ex. 7.2.6 : Cycles of computation

Turing machine is shown in Fig. Ex. 7.2.6(a).

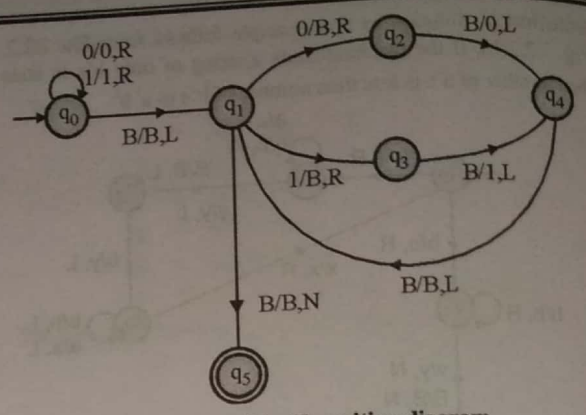


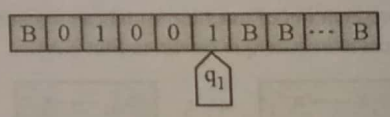
Fig. Ex. 7.2.6(a) : Transition diagram

	0	1	B
→q_0	(q_0,0,R)	(q_0,1,R)	(q_1,B,L)
q_1	(q_2,B,R)	(q_3,B,R)	(q_5,B,N)
q_2	–	–	(q_4,0,L)
q_3	–	–	(q_4,1,L)
q_4	–	–	(q_1,B,L)
q_5^*	q_5	q_5	q_5

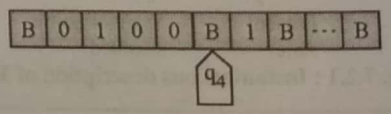
Fig. Ex. 7.2.6(b) : Transition table

Meaning of various states

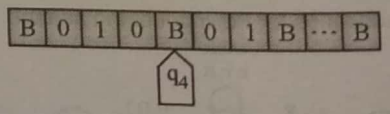
- Initial state q_0 is being used to skip the string of 0's and 1's so that the head can be positioned on last character.
- On seeing a B (blank), the head takes a left turn and positions itself on the last character. Machine enters state q_1.



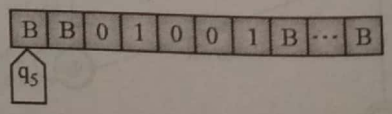
- A path through $q_1 \rightarrow q_3 \rightarrow q_4$ is for right shifting 1.



- A path through $q_1 \rightarrow q_2 \rightarrow q_4$ is for right shifting 0.



- Machine halts in state q_5.



Scanned by CamScanner

Example 7.2.7

Design a TM to make a copy of a string over {0,1}

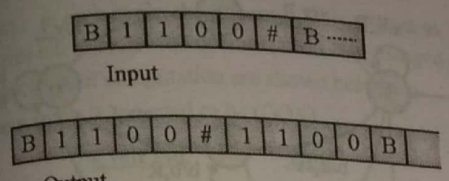


Fig. Ex. 7.2.7

Solution :

- Two copies of the strings are separated by #.
- Machine is described in Fig. Ex. 7.2.7(a).

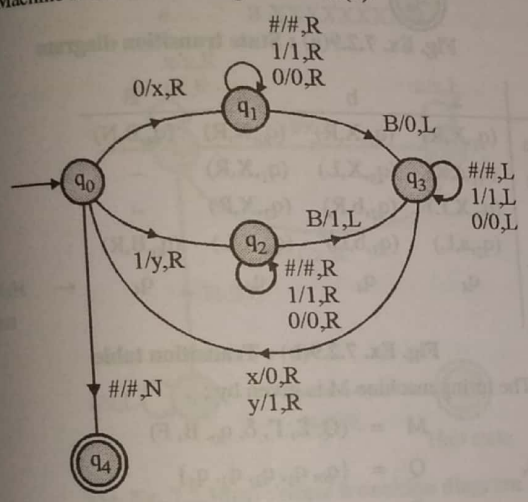


Fig. Ex. 7.2.7(a) : Transition diagram

	0	1	#	B	x	y
$\rightarrow q_0$	(q_1,x,R)	(q_2,y,R)	$(q_4,\#,N)$	–	–	–
q_1	$(q_1,0,R)$	$(q_1,1,R)$	$(q_1,\#,R)$	$(q_3,0,L)$	–	–
q_2	$(q_2,0,R)$	$(q_2,1,R)$	$(q_2,\# R)$	$(q_3,1,L)$	–	–
q_3	$(q_3,0,L)$	$(q_3,1,L)$	$(q_3 \#,L)$	–	$(q_0,0,R)$	$(q_0,1,R)$
q_4^*	q_4	q_4	q_4	q_4	q_4	q_4
↑ Halting state						

Fig. Ex. 7.2.7(b) : Transition table

- The turing machine M is given by :

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

Where,

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

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

$\Gamma = \{0, 1, x, y, \#, B\}$

q_0 = Initial state

B = Blank symbol

$F = \{q_4\}$

- Working of machine for an input 011 is shown in Fig. Ex. 7.2.7(c) :

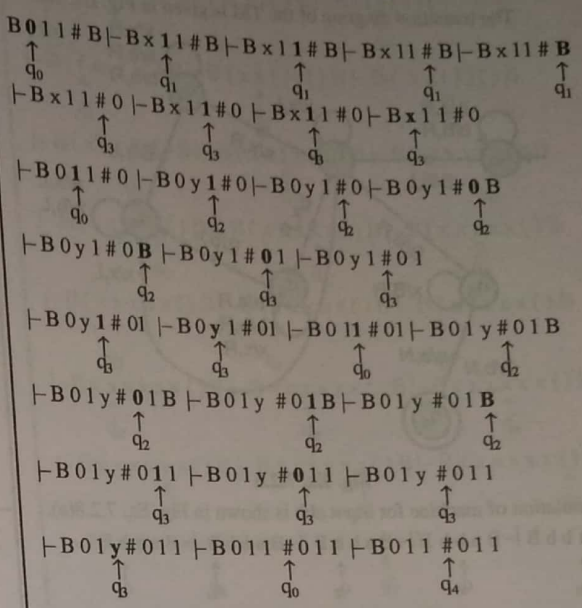


Fig. Ex. 7.2.7(c)

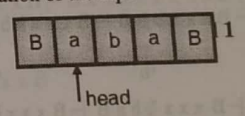
Example 7.2.8

Design TM to reverse a string $\Sigma = \{a, b\}$

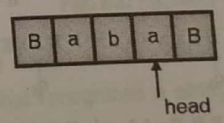
Solution :

The working of the TM is being explained with the help of the string aba.

Initial configuration of the tape



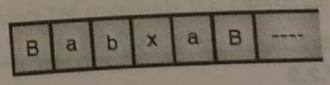
Step 1 : Head is moved to the rightmost symbol



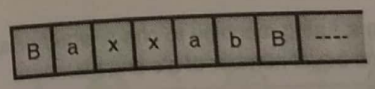
Step 2 : We create another copy of aba in the reverse order. It will require 3 cycles as the length of aba is 3. In each cycle, the input symbol 'a' and 'b' are renamed as x.

Step 3 :

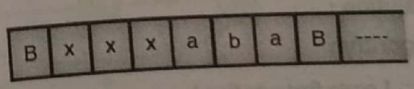
After 1st cycle



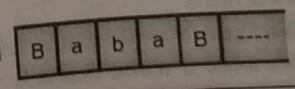
After 2nd cycle



After 3rd cycle



x's are erased



Scanned by CamScanner

The transition diagram of the TM is given in Fig. Ex. 7.2.8.

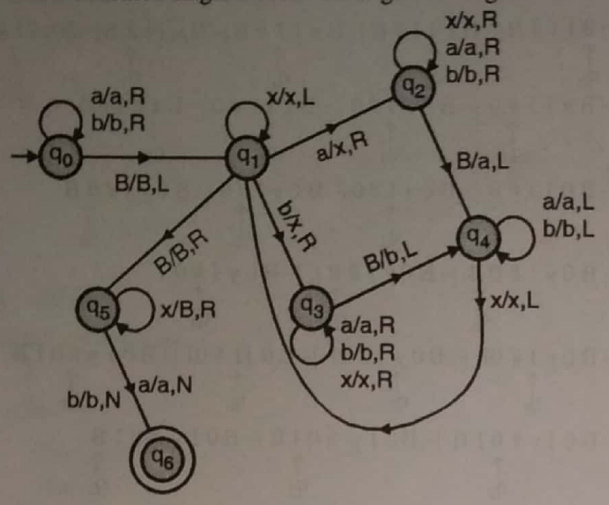


Fig. Ex. 7.2.8

Simulation of machine for input abb is shown in Fig. Ex. 7.2.8(a).

B a b b B ⊢ B a b b B ⊢ B a b b B ⊢ B a b b B ⊢ B a b b B
↑ q_0 ↑ q_0 ↑ q_0 ↑ q_0 ↑ q_1

⊢ B a b x B ⊢ B a b x b B ⊢ B a b x b B ⊢ B a x x b B
↑ q_3 ↑ q_4 ↑ q_1 ↑ q_3

⊢ B a x x b B ⊢ B a x x b B ⊢ B a x x b b B ⊢ B a x x b b B
↑ q_3 ↑ q_3 ↑ q_4 ↑ q_4

⊢ B a x x b b B ⊢ B a x x b b B ⊢ B x x x b b B ⊢ B x x x b b B
↑ q_1 ↑ q_1 ↑ q_2 ↑ q_2

⊢ B x x x b b B ⊢ B x x x b b B ⊢ B x x x b b B ⊢ B x x x b b a B
↑ q_2 ↑ q_2 ↑ q_2 ↑ q_4

⊢ B x x x b b a B ⊢ B x x x b b a B ⊢ B x x x b b a B
↑ q_4 ↑ q_4 ↑ q_1

⊢ B x x x b b a B ⊢ B x x x b b a B ⊢ B x x x b b a B
↑ q_1 ↑ q_1 ↑ q_5

⊢ B B x x b b a B ⊢ B B B x b b a ⊢ B B B B b b a B
↑ q_5 ↑ q_5 ↑ q_5

⊢ B B B B b b a B
↑ q_6 – Halt

Fig. Ex. 7.2.8(a)

Example 7.2.9

Design a turing machine to check whether a string over {a,b} contains equal number of a's and b's.

Solution :

Algorithm

1. Locate first a or first b.
2. If it is 'a' then locate 'b' rewrite them as x.
3. If it is 'b' then locate 'a' rewrite them as x.
4. Repeat steps from 1 to 3 till every a or b is re-written as x.

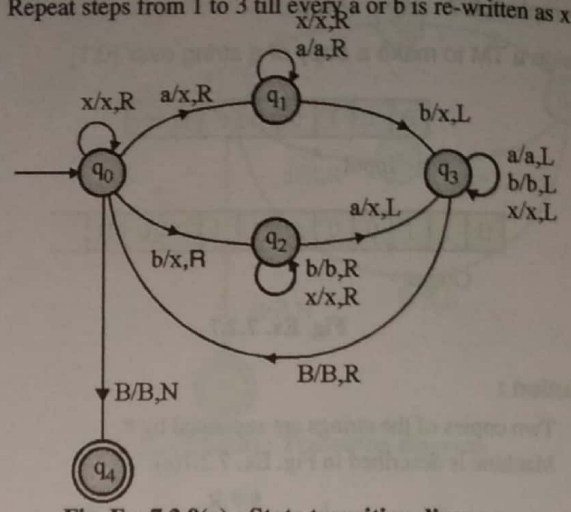


Fig. Ex. 7.2.9(a) : State transition diagram

	a	b	X	B	
$\rightarrow q_0$	(q_1,X,R)	(q_2,X,R)	(q_0,X,R)	(q_4,B,N)	
q_1	(q_1,a,R)	(q_3,X,L)	(q_1,X,R)	–	
q_2	(q_3,X,L)	(q_2,b,R)	(q_2,X,R)	–	
q_3	(q_3,a,L)	(q_3,b,L)	(q_3,X,L)	(q_0,B,R)	
q_4^*	q_4	q_4	q_4	q_4	← Halting state

Fig. Ex. 7.2.9(b) : Transition table

The turing machine M is given by :

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

Where,

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

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

$\Gamma = \{a, b, X, B\}$

q_0 = Initial state

B = Blank symbol

$F = \{q_4\}$

Working of machine for an input abba is shown in Fig. Ex. 7.2.9(c)

⊢ 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

Fig. Ex. 7.2.9(c)

Scanned by CamScanner

Example 7.2.10

Construct a TM for checking well formedness of parentheses.

Solution : In each cycle, the left-most ')' is written as X, then the head moves left to locate the nearer '(' and it is changed to X.

The cycles of computation are shown 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

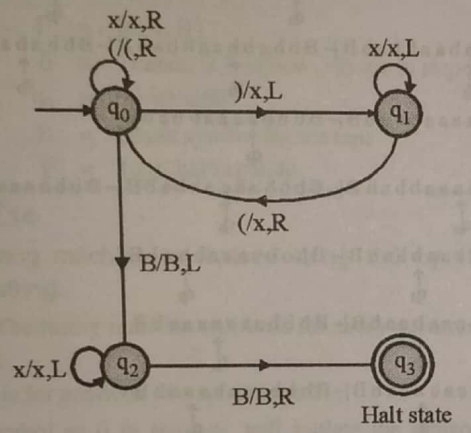


Fig. Ex. 7.2.10(a) : State transition diagram

	(	)	x	B
q_0	$(q_0,(,R)$	(q_1,x,L)	(q_0,x,R)	(q_2,B,L)
q_1	(q_0,x,R)	–	(q_1,x,L)	–
q_2	–	–	(q_2,x,L)	(q_3,B,R)
q_3^*	q_3	q_3	q_3	q_3

⇓ Halting state

Fig. Ex. 7.2.10(b) : State transition table

The Turing machine M is given by :

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

where,

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

$$\Sigma = \{(,)\}$$

$$\Gamma = \{(,), x, B\}$$

δ is given in Fig. Ex. 7.2.10(a) or Ex. 7.2.10(b)

q_0 = Initial state

B = Blank symbol

F = $\{q_3\}$, halting state

Making of the machine for input (()())() is given in Fig. Ex. 7.2.10(c) :

$\underset{q_0}{B}(()())()B \vdash B(\underset{q_0}{(})())()B \vdash B(\underset{q_0}{(})())()$

$\vdash B(\underset{q_1}{(}x())()B \vdash B(x\underset{q_0}{x}())()B \vdash B(x\underset{q_0}{x}())()B$

$\vdash B(xx\underset{q_0}{(})()B \vdash B(xx\underset{q_1}{(}x)()B \vdash B(xxx\underset{q_0}{x})()B$

$\vdash B(xxxx\underset{q_0}{)})()B \vdash B(xxxx\underset{q_1}{x}()B \vdash B(xx\underset{q_1}{x}xx()B$

$\vdash B(\underset{q_1}{x}xxxx()B \vdash B(\underset{q_1}{x}xxxx()B \vdash B(\underset{q_1}{x}xxxx()B$

$\vdash Bx\underset{q_0}{x}xxxx()B \vdash Bxx\underset{q_0}{x}xxx()B \vdash Bxxx\underset{q_0}{x}xx()B$

$\vdash Bxxxx\underset{q_0}{x}x()B \vdash Bxxxxx\underset{q_0}{x}()B \vdash Bxxxxxx\underset{q_0}{()}B$

$\vdash Bxxxxxx\underset{q_0}{()}B \vdash Bxxxxxx\underset{q_1}{(}xB$

$\vdash Bxxxxxxx\underset{q_0}{x}B \vdash Bxxxxxxx\underset{q_0}{x}B$

$\vdash Bxxxxxxx\underset{q_2}{x}B \vdash Bxxxxxx\underset{q_2}{x}xB \vdash Bxxxxx\underset{q_2}{x}xxB$

$\vdash Bxxxx\underset{q_2}{x}xxxB \vdash Bxxx\underset{q_2}{x}xxxxB \vdash Bxx\underset{q_2}{x}xxxxxB$

$\vdash Bx\underset{q_2}{x}xxxxxxB \vdash B\underset{q_2}{x}xxxxxxxB \vdash \underset{q_2}{B}xxxxxxxB$

$\vdash B\underset{q_3}{x}xxxxxxxB$

Fig. Ex. 7.2.10(c)

Example 7.2.11

Design a TM that recognizes a string containing aba as a substring.

Solution : This problem can be solved by a DFA. A problem that can be solved by a DFA can also be solved by a Turing machine.

- The DFA is shown in Fig. Ex. 7.2.11(a).
- The equivalent TM is shown in Fig. Ex. 7.2.11(b).
- The head scans the input from left to right without modifying it.

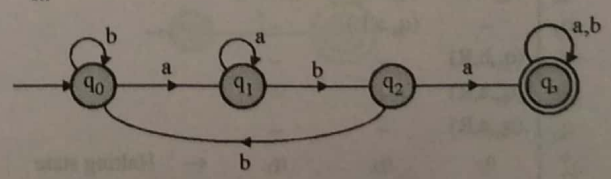


Fig. Ex. 7.2.11(a) : DFA for substring aba

Scanned by CamScanner

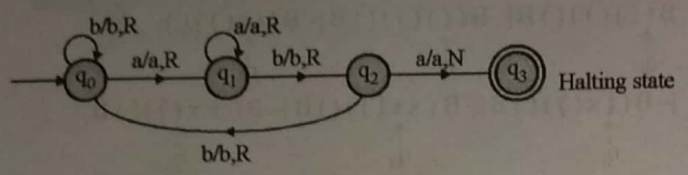


Fig. Ex. 7.2.11(b) : TM obtained from the DFA of Fig. Ex. 7.2.11(a)

The Turing machine M is given by :

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

Where,

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{a, b, B\}$$

δ is given in Fig. Ex. 7.2.11(b)

q_0 = Initial state

B = Blank symbol,

F = $\{q_0\}$, Halting state

Example 7.2.12

Design a TM that replaces every occurrence of abb by baa.

Solution :

- The state q_2 implies that the preceding two characters are ab.
- In input b in state q_2, completes the sequence abb.
- abb is changed to baa by a transition through $q_2 \to q_3 \to q_4 \to q_5$.

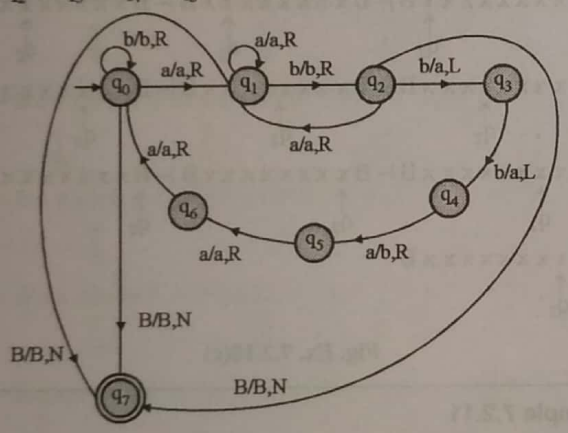


Fig. Ex. 7.2.12(a) : Transition diagram

- 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. 7.2.12(b) : State transition table

The Turing machine M is given by

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

where,

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{a, b, B\}$$

δ is given is Fig. Ex. 7.2.12(a) or (b)

q_0 = initial state

B = blank symbol

F = $\{q_7\}$, Halting state

Working of the machine for input bbabbaabbab is shown in Fig. Ex. 7.2.12(c) :

$\underset{q_0}{B}\,\mathbf{b}\,b\,a\,b\,b\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,\underset{q_0}{\mathbf{b}}\,a\,b\,b\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,\underset{q_0}{\mathbf{a}}\,b\,b\,a\,a\,b\,b\,a\,b\,B$

$\vdash B\,b\,b\,\underset{q_1}{\mathbf{a}}\,\mathbf{b}\,b\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,a\,b\,\underset{q_2}{\mathbf{b}}\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,a\,\underset{q_3}{\mathbf{b}}\,a\,a\,a\,b\,b\,a\,b\,B$

$\vdash B\,b\,b\,\underset{q_4}{\mathbf{a}}\,a\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,b\,\underset{q_5}{\mathbf{a}}\,a\,a\,a\,b\,b\,a\,b\,B$

$\vdash B\,b\,b\,b\,\underset{q_6}{\mathbf{a}}\,a\,a\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,\underset{q_0}{\mathbf{a}}\,a\,b\,b\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_1}{\mathbf{a}}\,b\,b\,a\,b\,B$

$\vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_1}{\mathbf{b}}\,b\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_2}{\mathbf{b}}\,a\,b\,B$

$\vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_3}{\mathbf{b}}\,a\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_4}{\mathbf{a}}\,a\,a\,b\,B$

$\vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_5}{\mathbf{b}}\,a\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,a\,\underset{q_6}{\mathbf{b}}\,a\,\mathbf{a}\,b\,B$

$\vdash B\,b\,b\,b\,a\,a\,a\,b\,\underset{q_0}{\mathbf{a}}\,a\,b\,B \vdash B\,b\,b\,b\,a\,a\,a\,b\,a\,a\,\underset{q_1}{\mathbf{b}}\,B$

$\vdash B\,b\,b\,b\,a\,a\,a\,b\,a\,a\,a\,\underset{q_2}{\mathbf{b}}\,B \vdash B\,b\,b\,b\,a\,a\,a\,b\,a\,a\,a\,b\,\underset{q_7}{\mathbf{B}}$

Fig. Ex. 7.2.12(c)

Example 7.2.13

Construct Turing Machine that recognizes the language :

$L = \{x \in \{0,1\}^* \mid x \text{ ends in } 00\}$

Solution : The given language L can have an equivalent DFA. Therefore, we will first design a DFA and then convert this DFA into a TM.

Step 1 : DFA for the given language L.

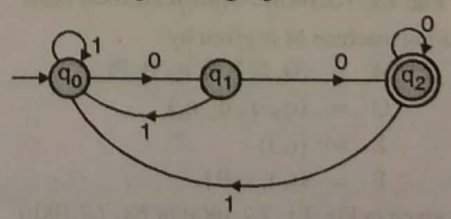


Fig. Ex. 7.2.13(a) : DFA for L

Scanned by CamScanner

Step 2 : TM from the DFA.

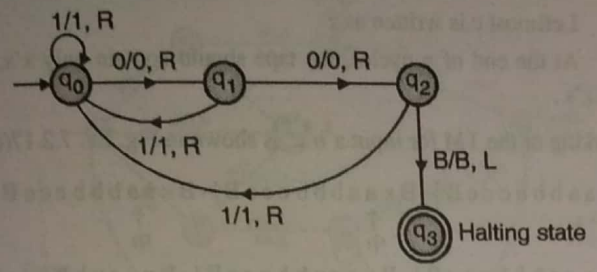


Fig. Ex. 7.2.13(b) : TM from DFA

– An input symbol B in either q_0 or q_1 will imply rejection.

The turing machine M is given by :

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

Where,

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

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

$\Gamma = \{0, 1, B\}$

δ = Transition function is given in step 2

q_0 = initial state

B = Blank symbol for the tape

F = $\{q_3\}$, halting state

Example 7.2.14

Design a turing machine to replace string 110 by 101 in binary input string.

Solution : The turing machine will look for every occurrence of the string 110.

– State q_2 is for previous two symbols as 11.

– Next symbol as 0 in state q_2, will initiate the replacement process to replace 110 by 101.

The turing machine M is given by :

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

Where,

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{0, 1, B\}$

δ = Transition function is shown using the transition diagram

B = Blank symbol for the tape

F = $\{q_5\}$, halting state

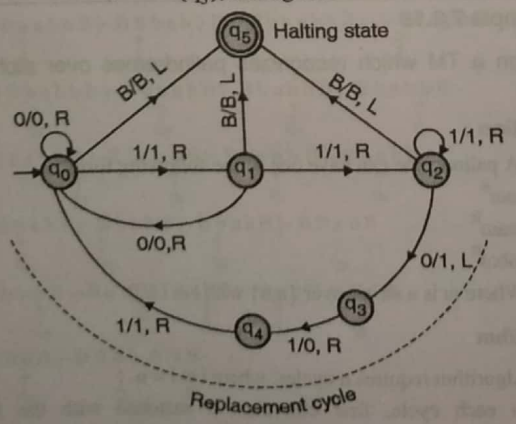


Fig. Ex. 7.2.14

Working of the machine for input 0101101 is shown in Fig. Ex. 7.2.14(a) :

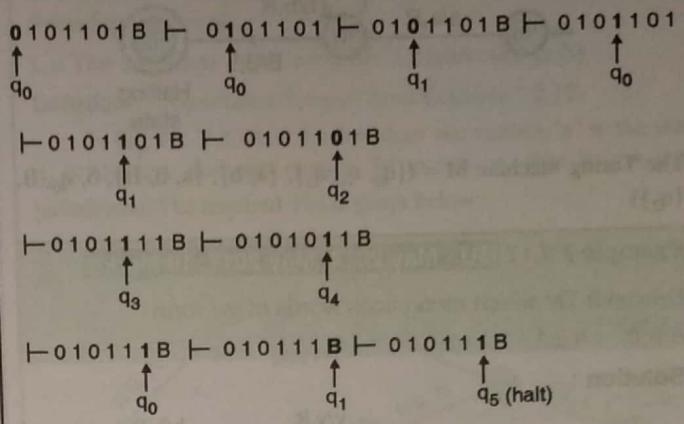


Fig. Ex. 7.2.14(a)

Example 7.2.15

Design and write out in full a turing machine that scans to the right until it finds two consecutive a's and then halts over the language of {a,b}.

Solution : The solution can be given in two steps :

1. Design a DFA to recognize strings with substring aa.
2. Designing a TM from the DFA.

Step 1 : DFA for strings having aa in it.

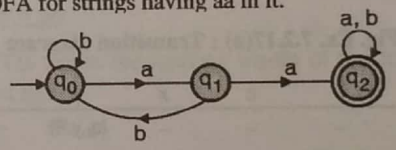


Fig. Ex. 7.2.15(a)

Step 2 : TM from the DFA

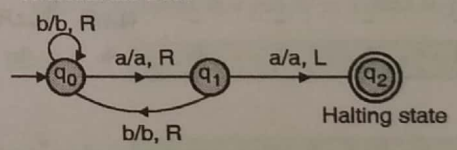


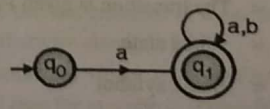
Fig. Ex. 7.2.15(b)

Example 7.2.16

Design a Turing machine with no more than three states that accepts the string a (a + b)*.

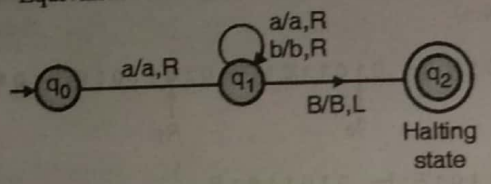
Solution : This problem can be solved by a DFA. A problem that can be solved by a DFA can also be solved by a TM.

Step 1 : DFA for (a(a + b)*)



Scanned by CamScanner

Step 2 : Equivalent TM from the DFA.



The Turing machine $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_2\})$

Example 7.2.17 **MU - May 2014, May 2017, 10 Marks**

Design a TM which recognizes words of the form $a^n b^n c^n \mid n \geq 1$.

Solution :

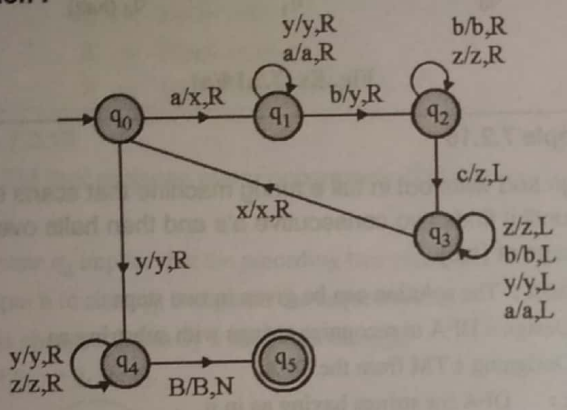


Fig. Ex. 7.2.17(a) : Transition diagram

	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^* Hating state	q_5	q_5	q_5	q_5	q_5	q_5	q_5

Fig. Ex. 7.2.17(b) : Transition table

The Turing machine M is given by :

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

Where,

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, x, y, z, B\}$

δ = The transition is given Fig. Ex. 7.2.17(a, b)

q_0 = Initial state

B = Blank symbol

F = $\{q_5\}$, Halting state

Algorithm

For a string $a^n b^n c^n$, the TM will need n cycles. In each cycle :

1. Leftmost a is written as x
2. Leftmost b is written as y
3. Leftmost c is written as z

At the end of n cycles, the tape should contain only x's, y's and z's .

Working of the TM for input $a^3b^3c^3$ is shown in Fig. Ex. 7.2.17(c) :

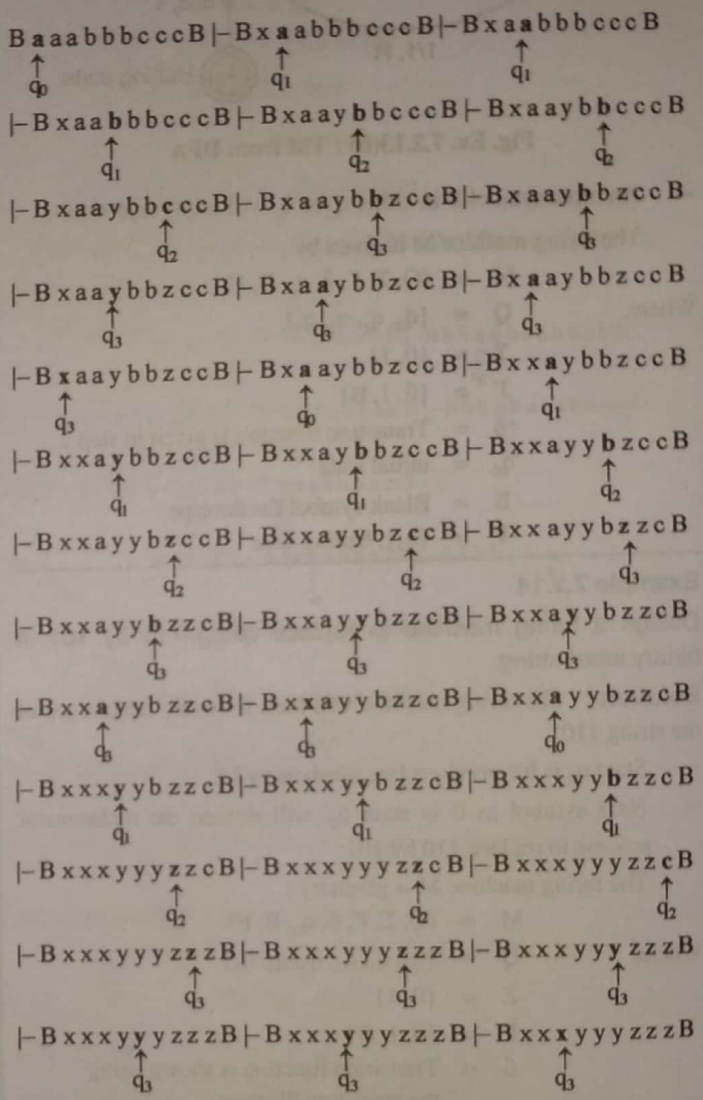


Fig. Ex. 7.2.17(c)

Example 7.2.18

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.

Scanned by CamScanner

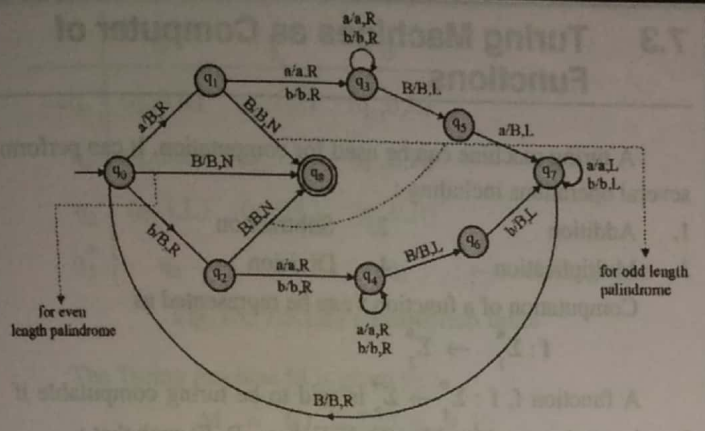


Fig. Ex. 7.2.18(a) : Transition diagram

3. If the leftmost character is 'a' the machine takes a path through

   $q_0 \to q_1 \to q_3 \to q_5 \to q_7$, looking for last character as 'a'.

4. If the leftmost character is 'b', the machine takes a path through

   $q_0 \to q_2 \to q_4 \to q_6 \to q_7$, 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. 7.2.18(a)

q_0 = initial state

B = blank symbol

F = $\{q_8\}$, halting state

Working of TM for input abbabba is shown in Fig. Ex. 7.2.18(a) :

$$\underset{q_0}{\mathbf{B}}\mathbf{a}bbabbaB \vdash B\underset{q_1}{\mathbf{B}}\mathbf{b}babbaB \vdash BB\underset{q_3}{\mathbf{b}}\mathbf{b}abbaB \vdash BB\underset{q_3}{\mathbf{b}}\mathbf{a}bbaB \vdash B\underset{q_3}{\mathbf{b}}\mathbf{b}abbaB$$

$$\vdash B\underset{q_3}{\mathbf{b}}\mathbf{a}\mathbf{b}baB \vdash B\underset{q_3}{\mathbf{b}}\mathbf{b}a\mathbf{b}aB \vdash B\underset{q_3}{\mathbf{b}}\mathbf{b}ab\mathbf{a}B \vdash B\underset{q_3}{\mathbf{b}}\mathbf{b}abb\mathbf{a}\mathbf{B}$$

$$\vdash B\underset{q_5}{\mathbf{b}}\mathbf{b}ab\mathbf{a}B \vdash B\underset{q_7}{\mathbf{b}}\mathbf{b}a\mathbf{b}BB \vdash B\underset{q_7}{\mathbf{b}}\mathbf{b}\mathbf{a}bBB \vdash B\underset{q_7}{\mathbf{b}}\mathbf{b}abbB$$

$$\vdash B\underset{q_7}{\mathbf{b}}\mathbf{b}abbB \vdash B\underset{q_7}{\mathbf{b}}\mathbf{b}abbB \vdash \underset{q_7}{\mathbf{B}}bbabbB \vdash \underset{q_0}{\mathbf{B}}\mathbf{b}\mathbf{b}abbB$$

$$\vdash BB\underset{q_2}{\mathbf{b}}\mathbf{a}bbB \vdash Bb\underset{q_4}{\mathbf{a}}bbB \vdash Bba\underset{q_4}{\mathbf{b}}bB \vdash Bbab\underset{q_4}{\mathbf{b}}B$$

$$\vdash Bbabb\underset{q_4}{\mathbf{B}} \vdash Bbab\underset{q_6}{\mathbf{b}}B \vdash Bba\underset{q_7}{\mathbf{b}}BB \vdash Bbab\underset{q_7}{\mathbf{B}}$$

$$\vdash B\underset{q_7}{\mathbf{b}}abB \vdash \underset{q_7}{\mathbf{B}}babB \vdash B\underset{q_0}{\mathbf{b}}abB \vdash BB\underset{q_2}{\mathbf{a}}bB$$

$$\vdash Ba\underset{q_4}{\mathbf{b}}B \vdash Ba\underset{q_4}{\mathbf{b}}\underset{q_4}{\mathbf{B}} \vdash Ba\underset{q_6}{\mathbf{b}}B \vdash B\underset{q_7}{\mathbf{a}}BB \vdash \underset{q_7}{\mathbf{B}}aB$$

$$\vdash B\underset{q_0}{\mathbf{a}}B \vdash BB\underset{q_1}{\mathbf{B}} \vdash BB\underset{q_8}{\mathbf{B}}$$

Fig. Ex. 7.2.18(a)

Example 7.2.19

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 7.2.18.

In Fig. Ex. 7.2.18, if the tape does not contain 'a' in the state q_5 or the tape does not contain 'b' in state q_6 then it is a non-palindrome. The required TM is given below :

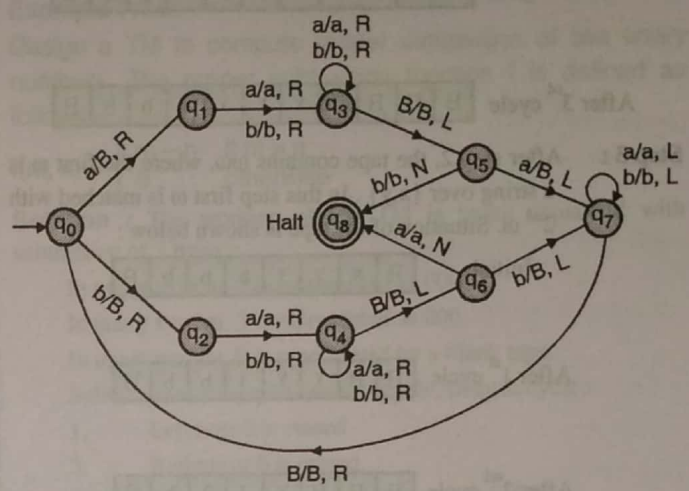


Fig. Ex. 7.2.19 : TM

Example 7.2.20

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	a	b	b	a	b	b	a	b	b	B
After 1st cycle	B	x	y	b	a	b	b	a	b	2	B
After 2nd cycle	B	x	y	y	x	b	b	a	2	2	B
After 3rd cycle	B	x	y	y	x	y	y	1	2	2	B

1.2 Locate the last alphabet of 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 tape 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}.

Scanned by CamScanner

Situation of the tape is shown below :

Initial	B	x	y	y	x	y	y	1	2	2	B
After 1st cycle	B	B	y	y	x	y	y	a	2	2	B
After 2nd cycle	B	B	B	y	x	y	y	a	b	2	B
After 3rd cycle	B	B	B	B	x	y	y	a	b	b	B

Step 3 : After step 2, the tape contains 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	x	y	y	a	b	b	B
After 1st cycle	B	B	y	y	1	b	b	B
After 2nd cycle	B	B	B	y	1	2	b	B
After 3rd cycle	B	B	B	B	1	2	2	B

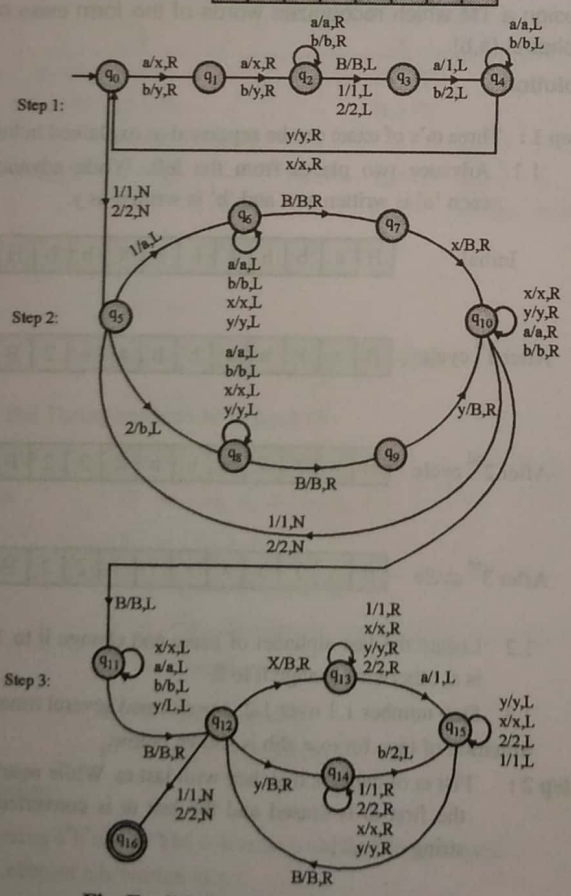


Fig. Ex. 7.2.20 : Transition diagram

7.3 Turing Machines as Computer of Functions

A turing machine can be used for computation. It can perform several operations including :

1. Addition
2. Subtraction
3. Multiplication
4. Division

Computation of a function f can be represented as

$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

A function f, $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is said to be turing computable if there is a turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ such that :

$$\boxed{(q_0, B\omega B) \overset{*}{\underset{M}{\vdash}} (q_f, B\mu B),}$$

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 7.3.1

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.

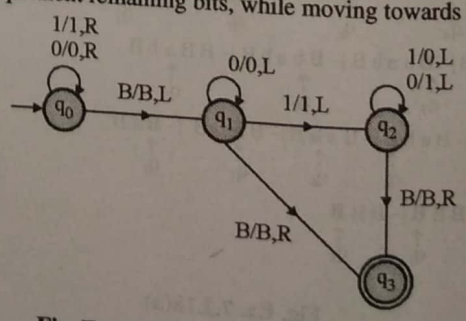


Fig. Ex. 7.3.1(a) : Transition diagram

Scanned by CamScanner

	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. 7.3.1(b) : Transition table

The Turing machine M is given by :

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

Where,

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

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

$$\Gamma = \{0, 1, B\}$$

The transition function δ is given in Fig. Ex. 7.3.1(a) or (b),

q_0 = Initial state

B = Blank symbol

F = $\{q_3\}$, Halting state

Working of TM for input 0101101000 is shown in Fig. Ex. 7.3.1(c) :

$\underset{q_0}{B0}1011101000B \vdash B0\underset{q_0}{1}011101000B \vdash B01\underset{q_0}{0}11101000B$

$\vdash B010\underset{q_0}{1}1101000B \vdash B0101\underset{q_0}{1}01000B \vdash B01011\underset{q_0}{0}1000B$

$\vdash B010110\underset{q_0}{1}000B \vdash B01011\underset{q_0}{0}1000B \vdash B0101101\underset{q_0}{0}00B$

$\vdash B01011010\underset{q_0}{0}0B \vdash B0101101000\underset{q_0}{B} \vdash B010110100\underset{q_1}{0}B$

$\vdash B01011010\underset{q_1}{0}0B \vdash B0101101\underset{q_1}{0}00B \vdash B010110\underset{q_1}{1}000B$

$\vdash B01011\underset{q_2}{0}1000B \vdash B0101\underset{q_2}{1}111000B$

$\vdash B0101\underset{q_2}{0}11000B \vdash B010\underset{q_2}{0}011000B \vdash B01\underset{q_2}{1}0011000B$

$\vdash \underset{q_2}{B0}0100011000B \vdash \underset{q_2}{B}10100011000B \vdash \underset{q_3}{B1}0100011000B$

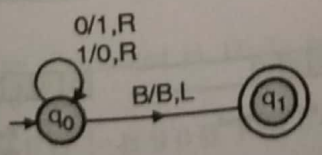
Fig. Ex. 7.3.1(c)

Example 7.3.2

Design a TM to find the 1's complement of a given binary input.

Solution :

1's complement of a binary number can be computed by complementing every bit of the given binary number.



(a) Transition diagram

	0	1	B	
q_0	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_1, B, L)	
q_1^*	q_1	q_1	q_1	←Halting state

(b) Transition table

Fig. Ex. 7.3.2

The Turing machine $M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_1\})$

The transition function δ is given through transition diagram/table.

Example 7.3.3

Design a TM to compute proper subtraction of two unary numbers. The proper subtraction function f is defined as follows :

$$f(m, n) = \begin{cases} m - n & \text{if } m > n \\ 0 & \text{otherwise} \end{cases}$$

Solution : The working of the TM is being explained with subtraction of 3 from 5.

In unary system, 5 is represented as 00000.

In unary system, 3 is represented as 000.

In unary system, 0 is represented by a blank tape.

Subtraction will require several cycle. In each cycle :

1. Leftmost 0 is erased
2. Rightmost 0 is erased.

Situation of tape after each cycle is shown below :

Initial	B	0	0	0	0	0	#	0	0	0	B	---
After 1st cycle	B	B	0	0	0	0	#	0	0	B	B	---
After 2nd cycle	B	B	B	0	0	0	#	0	B	B	B	---
After 3rd cycle	B	B	B	B	0	0	#		B	B	---	

Transition diagram and transition table are given in Fig. Ex. 7.3.3(a) and (b).

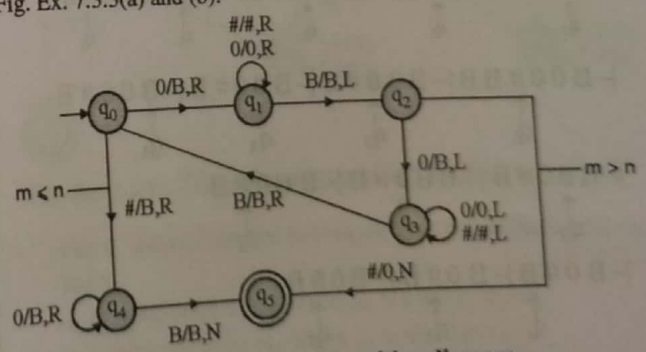


Fig. Ex. 7.3.3(a) : Transition diagram

	0	#	B	
$\rightarrow q_0$	(q_1,B,R)	(q_4,B,R)	–	
q_1	$(q_1,0,R)$	$(q_1,\#,R)$	(q_2,B,L)	
q_2	(q_3,B,L)	$(q_5\ 0,N)$	–	
q_3	$(q_3,0,L)$	$(q_3,\#,L)$	(q_0,B,R)	
q_4	(q_4,B,R)	–	(q_5,B,N)	
q_5^*	q_5	q_5	q_5	← Halting state

Fig. Ex. 7.3.3(b) : Transition table

Scanned by CamScanner

The Turing machine M is given by :

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

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = (0, 1, \#, B)$

The transition function δ is given in Fig. Ex. 7.3.3(a) and (b)

q_0 = initial state,

B = blank symbol

F = $\{q_5\}$, Halting state

The working of TM is being simulated for 5-3 is shown in Fig. Ex. 7.3.3(c) :

```
B00000#000B|-BB0000#000B|-BB00000#000B
 ↑q0          ↑q1            ↑q1
|-BB 0000#000B|-BB00000#000B|-BB00000#000B
     ↑q1          ↑q1            ↑q1
|-BB 00000#000B|-BB00000#00B|-BB00000#00B
        ↑q1          ↑q1          ↑q1
|-BB 00000#00B|-B00000#00B|-B00000#00BB
          ↑q1          ↑q2          ↑q3
|-B00000#00B|-B00000#00B|-B00000#00B|-B00000#00B
      ↑q3          ↑q3          ↑q3          ↑q3
|-B00000#00B|-B00000#00B|-B00000#00B|-B00000#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
```

Fig. Ex. 7.3.3(c)

Example 7.3.4

Design TM to compare two numbers, which will produce the output L if first number is less than the second number, output G if first is greater than the second number and E otherwise.

Solution : This machine is an extension of machine for proper subtraction given in Example 7.3.3.

- An input # in state q_2 means m > n. Everything is erased from the tape and an alphabet G is written on the tape.
- A # followed by 0 in state q_0 means m < n. L is written on the tape.
- A # followed by B(blank) in state q_0 means m = n. E is written on the tape.

The Turing machine

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{0\}, \{0, \#, B, G, L, E\}, \delta, q_0, B, \{q_7\})$$

The transition function δ is specified using the transition diagram.

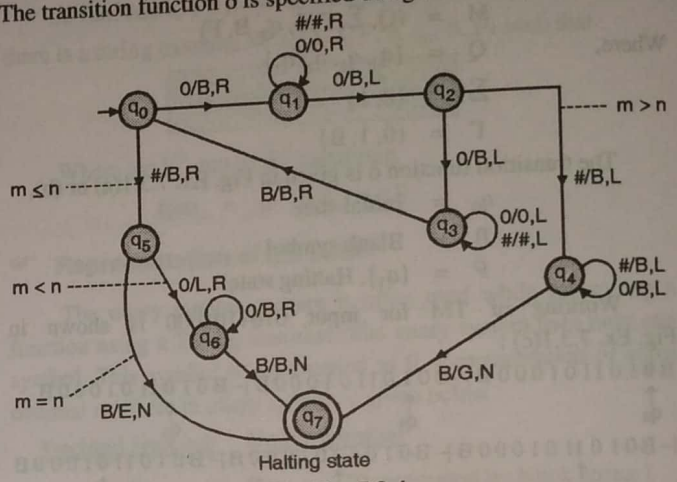


Fig. Ex. 7.3.4

Example 7.3.5

Design a TM that computes the function

$g(x, y) = x + y$, if $x \geq y$

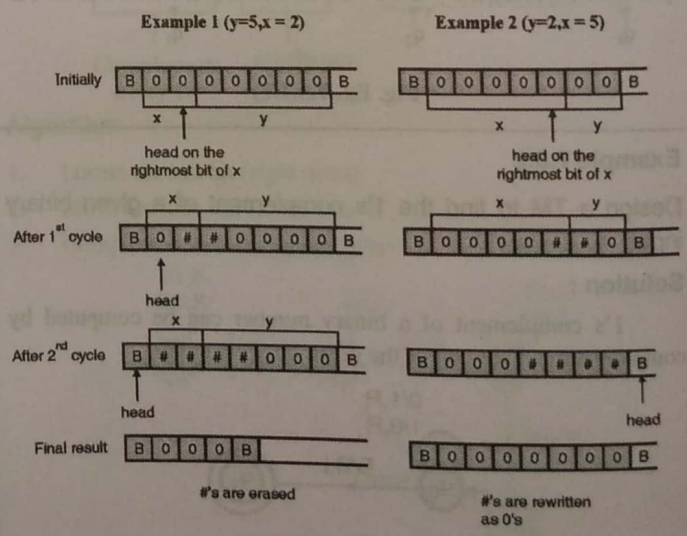
$= y - x$, if $y > x$

Simulate working for

(i) $x = 2, y = 2$ (ii) $x = 0, y = 2$.

Solution : Working of the machine is being explained with the help of two examples :

1. $y > x$ [y = 5, x = 2]
2. $y \leq x$ [y = 2, x = 5]



- Initially, head is positioned on the rightmost bit of x.

Scanned by CamScanner

In each cycle :

1. Nearest (from head) zero of x is written as #
2. Nearest zero of y is written as #

If the head hits the B(blank) on the left side before B(blank) on the right side then #'s are erased to perform y – x.

If the head hits the B(blank) on the right side before B(blank) on the left side then #'s are re-written as 0's to perform x + y.

The transition diagram of the TM is shown in Fig. Ex. 7.3.5.

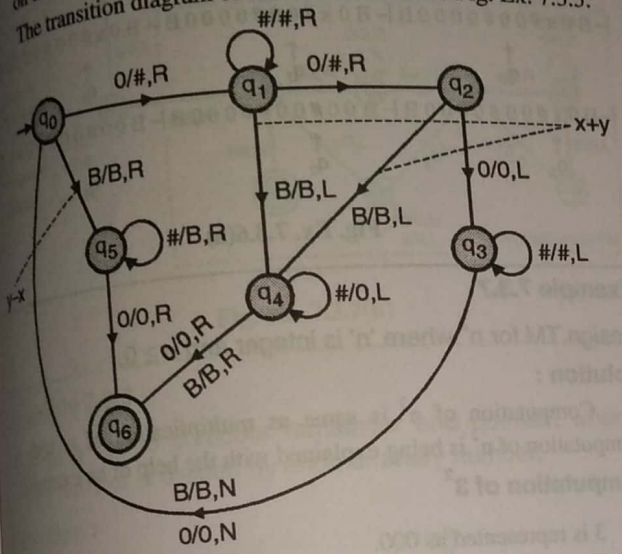


Fig. Ex. 7.3.5

The Turing machine $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0\}, \{0, \#, B\}, \delta, q_0, B, \{q_6\})$

Where δ is specified through the transition diagram.

Simulation of machine

(i) x = 2, y = 2

$\vdash B0000B \vdash B0\#00B \vdash B0\#\#0B \vdash B0\#\#0B$
(q_0, q_1, q_2, q_3)

$\vdash B0\#\#0B \vdash B0\#\#0B \vdash B0\#\#0B$
(q_3, q_3, q_0)

$\vdash B\#\#\#0B \vdash B\#\#\#0B \vdash B\#\#\#0B \vdash B\#\#\#\#B$
(q_1, q_1, q_1, q_2)

$\vdash B\#\#\#\#B \vdash B\#\#\#0B \vdash B\#\#00B \vdash B\#000B$
(q_4, q_4, q_4, q_4)

$\vdash B0000B \vdash B00000B$
(q_4, q_6 – halt)

(ii) x = 0, y = 2

x = 0, will be represented by blanks and y = 2 will be represented by two 0's.

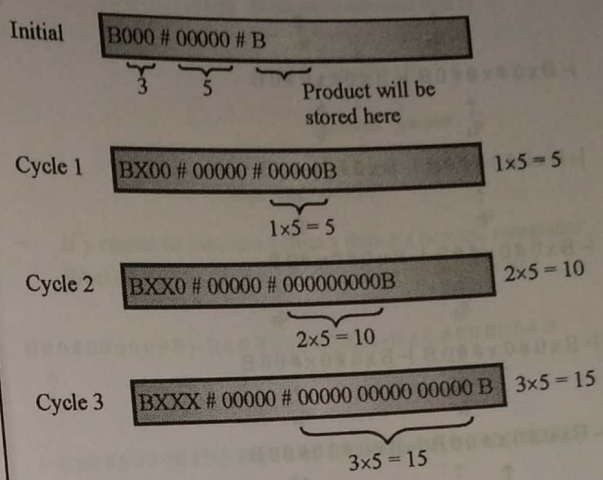
$BB00B \vdash B00B \vdash B00B$
(q_0, q_5, q_6 – halt)

Example 7.3.6

Design a TM to compute multiplication of two unary numbers.

Solution : Multiplication algorithm is being explained with the help of an example.

3 × 5 will require three cycles.



- To calculate 3 × 5, three times, 5 zero's are appended.
  Unary representation of 3 is 000.
  Unary representation of 5 is 00000.
- 3, 5 and the result, are separated by #.
- Inside each major cycles (three cycles for 3), there will be a number of minor cycles (5 minor cycles for 5) to append 0's one at a time.

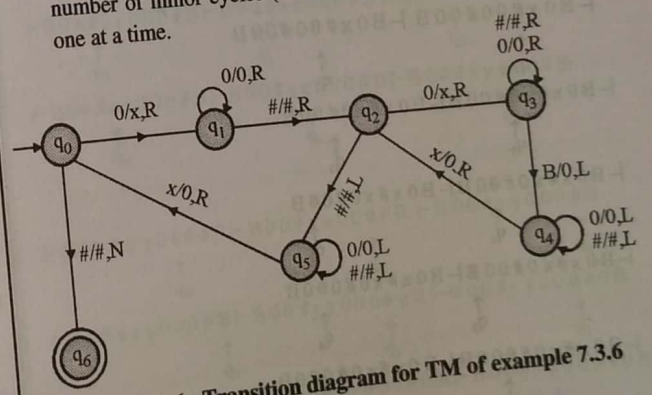


Fig. Ex. 7.3.6 : Transition diagram for TM of example 7.3.6

- Let us assume that the two numbers to be multiplied are x_1 and x_2.
  x_1 is represented by ω_1, where ω_1 is a string of 0's.
  x_2 is represented by ω_2, where ω_2 is a string of 0's.
  $x_1 * x_2$ is represented by ω_3, where ω_3 is a string 0's.
  # separates ω_1 and ω_2, ω_2 and ω_3.
- In the TM shown in Fig. Ex. 7.3.6, there are two cycles.
- The cycle $q_0 \to q_1 \to q_2 \to q_5 \to q_0$ appends ω_2 to ω_3 for every zero in ω_1, with the help of cycle $q_2 \to q_3 \to q_4 \to q_2$
  Working of TM for 2 × 2 is shown in Fig. Ex. 7.3.6(a) :

Scanned by CamScanner

$B\mathbf{0}0\#00\#B \vdash Bx\mathbf{0}\#00\#B \vdash Bx0\#00\#B \vdash Bx0\#\mathbf{0}0\#B$
$\uparrow q_0 \quad \uparrow q_1 \quad \uparrow q_1 \quad \uparrow q_2$

$\vdash Bx0\#x\mathbf{0}\#B \vdash Bx0\#x0\#B$
$\uparrow q_3 \quad \uparrow q_3$

$\vdash Bx0\#x0\#\mathbf{B} \vdash Bx0\#x0\#0B$
$\uparrow q_3 \quad \uparrow q_4$

$\vdash Bx0\#x\mathbf{0}\#0B \vdash Bx0\#\mathbf{x}0\#0B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash Bx0\#0\mathbf{0}\#0B \vdash Bx0\#0x\#0B$
$\uparrow q_2 \quad \uparrow q_3$

$\vdash Bx0\#0x\#\mathbf{0}0B \vdash Bx0\#0x\#0\mathbf{B}$
$\uparrow q_3 \quad \uparrow q_3$

$\vdash Bx0\#0x\#\mathbf{0}0B \vdash Bx0\#0x\#00B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash Bx0\#0\mathbf{x}\#00B \vdash Bx0\#00\#00B$
$\uparrow q_4 \quad \uparrow q_2$

$\vdash Bx0\#0\mathbf{0}\#00B \vdash Bx0\#\mathbf{0}0\#00B \vdash Bx0\#00\#00B$
$\uparrow q_5 \quad \uparrow q_5 \quad \uparrow q_5$

$\vdash Bx\mathbf{0}\#00\#00B \vdash B\mathbf{x}0\#00\#00B \vdash B0\mathbf{0}\#00\#00B$
$\uparrow q_5 \quad \uparrow q_5 \quad \uparrow q_0$

$\vdash B0x\#00\#00B \vdash B0x\#\mathbf{0}0\#00B$
$\uparrow q_1 \quad \uparrow q_2$

$\vdash B0x\#x\mathbf{0}\#00B \vdash B0x\#x0\#00B$
$\uparrow q_3 \quad \uparrow q_3$

$\vdash B0x\#x0\#\mathbf{0}0B \vdash B0x\#x0\#0\mathbf{0}B$
$\uparrow q_3 \quad \uparrow q_3$

$\vdash B0x\#x0\#00\mathbf{B} \vdash B0x\#x0\#0\mathbf{0}0B$
$\uparrow q_3 \quad \uparrow q_4$

$\vdash B0x\#x0\#\mathbf{0}00B \vdash B0x\#x0\#000B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash B0x\#x\mathbf{0}\#000B \vdash B0x\#\mathbf{x}0\#000B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash B0x\#0\mathbf{0}\#000B \vdash B0x\#0x\#000B$
$\uparrow q_2 \quad \uparrow q_3$

$\vdash B0x\#0x\#\mathbf{0}00B \vdash B0x\#0x\#0\mathbf{0}0B$
$\uparrow q_3 \quad \uparrow q_3$

$\vdash B0x\#0x\#00\mathbf{0}B \vdash B0x\#0x\#000\mathbf{B}$
$\uparrow q_3 \quad \uparrow q_3$

Fig. Ex. 7.3.6(a) Contd...

$\vdash B0x\#0x\#0000B \vdash B0x\#0x\#0000B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash B0x\#0x\#\mathbf{0}000B \vdash B0x\#0x\#0000B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash B0x\#0\mathbf{x}\#0000B \vdash B0x\#00\#0000B$
$\uparrow q_4 \quad \uparrow q_4$

$\vdash B0x\#00\#0000B \vdash B0x\#\mathbf{0}0\#0000B \vdash B0x\#00\#0000B$
$\uparrow q_5 \quad \uparrow q_5 \quad \uparrow q_5$

$\vdash B0\mathbf{x}\#00\#0000B \vdash B00\#00\#0000B \vdash B00\#00\#\underbrace{0000B}_{\text{result}}$
$\uparrow q_5 \quad \uparrow q_0 \quad \uparrow q_6$

Fig. Ex. 7.3.6(a)

Example 7.3.7

Design TM for n^2 where 'n' is integer and $n \geq 0$.

Solution :

Computation of n^2 is same as multiplication of n with n. Computation of n^2 is being explained with the help of an example.

Computation of 3^2

3 is represented as 000.

Initially the tape will contain B 0 0 0 # B

we make a copy of the string B 000 # 000 B

Now, the multiplication of 3 with 3 will require three cycles as shown in Fig. Ex. 7.3.7(a).

Initial	B 000 # 000 B
Cycle 1	B x 00 # 000 # 000
Cycle 2	B xx 0 # 000 # 000000
Cycle 3	B xxx # 000 # $\underbrace{000000000}_{n^2}$

Fig. Ex. 7.3.7(a)

- The transition diagram of the required TM is shown in Fig. Ex. 7.3.7(b).
- TM for multiplication is given in Example 7.3.6.
- TM for copy operation is given in Example 7.2.7.

Scanned by CamScanner

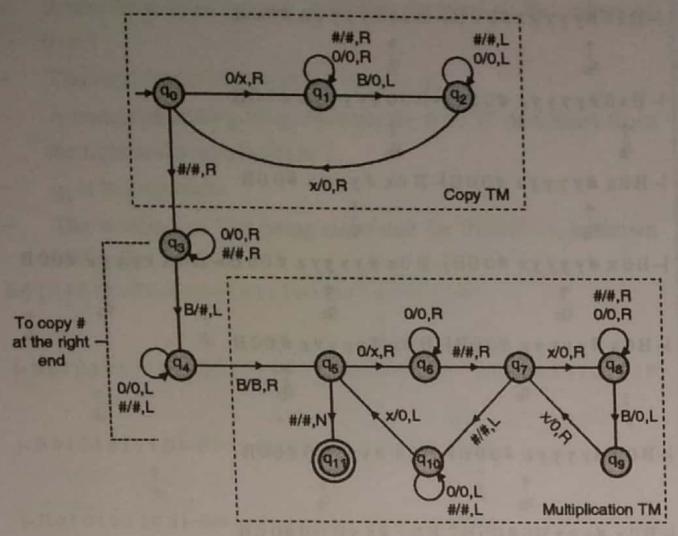


Fig. Ex. 7.3.7(b)

Example 7.3.8

Design a TM to compute remainder and quotient when a unary number is divided by another unary number.

Solution :

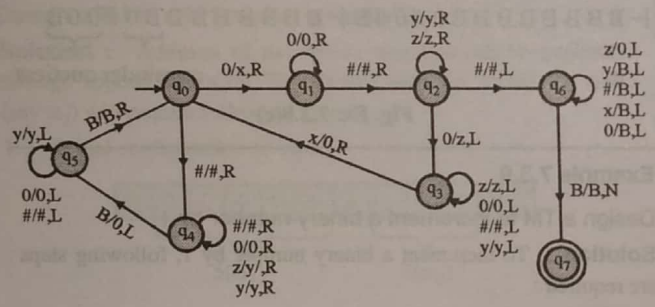


Fig. Ex. 7.3.8 : Transition diagram

- Division can be performed through repeated subtractions.
- To perform x ÷ y, both x and y are represented in unary system. y is divisor and x is dividend.
- y is repeatedly subtracted from x, as long as x ≥ y each time, y is subtracted from x, quotient is incremented by 1.
- Initial situation of tape for 5 ÷ 2 is shown in Fig. Ex. 7.3.8(a).

B	0	0	#	0	0	0	0	0	#	B	B ----

2 5 quotient will be stored here.

Fig. Ex. 7.3.8(a)

Working of machine for 5 ÷ 2 is shown in Fig. Ex. 7.3.8(b).

$q_0 \to q_1 \to q_2 \to q_3 \to q_0$ $q_0 \to q_4 \to q_5 \to q_0$ $q_2 \to q_6 \to q_7$

1. **Initially** B00#00000#B
2. B00#zz000#B [2 is subtracted from 5]
3. B00#yy000#0B [quotient is incremented by 1]
4. B00#yyzz0#0B [2 is subtracted from 3]
5. B00#yyyy0#00B [quotient is incremented by 1]
6. Bx0#yyyyz#00B [2 can not be subtracted from 1]
7. B0#00B

remainder quotient

Fig. Ex. 7.3.8(b)

- If y cannot be subtracted from x then z's become remainder. Working of TM for 5 ÷ 2 is shown in Fig. Ex. 7.3.8(c)

B**0**0#00000#B ⊢ Bx**0**#00000#B ⊢ Bx0#00000#B
q_0 q_1 q_1

⊢ Bx0#**0**0000#B ⊢ Bx0#z0000#B ⊢ Bx**0**#z0000#B
q_2 q_3 q_3

⊢ B**x**0#z0000#B ⊢ B0**0**#z0000#B ⊢ B0x#z0000#B
q_3 q_0 q_1

⊢ B0x#**z**0000#B ⊢ B0x#z**0**000#B ⊢ B0x#**z**z000#B
q_2 q_2 q_3

⊢ B0x#zz000#B ⊢ B0**x**#zz000#B ⊢ B00#zz000#B
q_3 q_3 q_0

⊢ B00#**z**z000#B ⊢ B00#y**z**000#B ⊢ B00#yy**0**00#B
q_4 q_4 q_4

⊢ B00#yy**0**0#B ⊢ B00#yy00**0**#B ⊢ B00#yy000#B
q_4 q_4 q_4

⊢ B00#yy000#**B** ⊢ B00#yy000#0B ⊢ B00#yy00**0**#0B
q_4 q_5 q_5

⊢ B00#yy**0**0#0B ⊢ B00#yy**0**00#0B ⊢ B00#y**y**000#0B
q_5 q_5 q_5

⊢ B00#**y**y000#0B ⊢ B00#yy000#0B ⊢ B0**0**#yy000#0B
q_5 q_5 q_5

⊢ B**0**0#yy000#0B ⊢ **B**00#yy000#0B ⊢ B**0**0#yy000#C
q_5 q_5 q_0

Fig. Ex. 7.3.8(c) Contd...

Scanned by CamScanner

$\vdash$ B x **0** # y y 0 0 0 # 0 B $\vdash$ B x 0 # y y 0 0 0 # 0 B $\vdash$ B x 0 # **y** y 0 0 0 # 0 B
q_1 q_1 q_2

$\vdash$ B x 0 # **y y** 0 0 0 # 0 B $\vdash$ B x 0 # y y **0** 0 0 # 0 B $\vdash$ B x 0 # **y y** z 0 0 # 0 B
q_2 q_2 q_3

$\vdash$ B x 0 # **y** y z 0 0 # 0 B $\vdash$ B x 0 # y y z 0 0 # 0 B $\vdash$ B x **0** # y y z 0 0 # 0 B
q_3 q_3 q_3

$\vdash$ B **x** 0 # y y z 0 0 # 0 B $\vdash$ B 0 **0** # y y z 0 0 # 0 B
q_3 q_0

$\vdash$ B 0 x # y y z 0 0 # 0 B $\vdash$ B 0 x # **y** y z 0 0 # 0 B $\vdash$ B 0 x # **y y** z 0 0 # 0 B
q_1 q_2 q_2

$\vdash$ B 0 x # y y **z** 0 0 # 0 B $\vdash$ B 0 x # y y z **0** 0 # 0 B
q_2 q_2

$\vdash$ B 0 x # y y **z** z 0 # 0 B $\vdash$ B 0 x # y **y** z z 0 # 0 B
q_3 q_3

$\vdash$ B 0 x # **y** y z z 0 # 0 B $\vdash$ B 0 x # y y z z 0 # 0 B
q_3 q_3

$\vdash$ B 0 **x** # y y z z 0 # 0 B $\vdash$ B 0 0 # y y z z 0 # 0 B
q_3 q_0

$\vdash$ B 0 0 # **y** y z z 0 # 0 B $\vdash$ B 0 0 # y **y** z z 0 # 0 B
q_4 q_4

$\vdash$ B 0 0 # y y **z** z 0 # 0 B $\vdash$ B 0 0 # y y y **z** 0 # 0 B
q_4 q_4

$\vdash$ B 0 0 # y y y y **0** # 0 B $\vdash$ B 0 0 # y y y y 0 # 0 B
q_4 q_4

$\vdash$ B 0 0 # y y y y 0 # **0** B $\vdash$ B 0 0 # y y y y 0 # 0 **B**
q_4 q_4

$\vdash$ B 0 0 # y y y y 0 # **0** 0 B $\vdash$ B 0 0 # y y y y 0 # 0 0 B $\vdash$ B 0 0 # y y y y **0** # 0 0 B
q_5 q_5 q_5

$\vdash$ B 0 0 # y y y **y** 0 # 0 0 B $\vdash$ B 0 0 # y y **y y** 0 # 0 0 B $\vdash$ B 0 0 # y y y y 0 # 0 0 B
q_5 q_5 q_5

$\vdash$ B 0 0 # **y** y y y 0 # 0 0 B $\vdash$ B 0 0 # y y y y 0 # 0 0 B $\vdash$ B **0 0** # y y y y 0 # 0 0 B
q_5 q_5 q_5

$\vdash$ B **0** 0 # y y y y 0 # 0 0 B $\vdash$ **B** 0 0 # y y y y 0 # 0 0 B $\vdash$ B **0** 0 # y y y y 0 # 0 0 B
q_5 q_5 q_0 remainder quotient

$\vdash$ B x **0** # y y y y 0 # 0 0 B $\vdash$ B x 0 # **y** y y y 0 # 0 0 B $\vdash$ B x 0 # **y y** y y 0 # 0 0 B
q_0 q_1 q_2

$\vdash$ B x 0 # y **y** y y 0 # 0 0 B $\vdash$ B x 0 # y y **y y** 0 # 0 0 B $\vdash$ B x 0 # y y y **y** 0 # 0 0 B
q_2 q_2 q_2

$\vdash$ B x 0 # y y y y **0** # 0 0 B $\vdash$ B x 0 # y y y y **y** z # 0 0 B
q_2 q_3

$\vdash$ B x 0 # y y **y** y z # 0 0 B $\vdash$ B x 0 # y **y** y y z # 0 0 B $\vdash$ B x 0 # **y** y y y z # 0 0 B
q_3 q_3 q_3

Fig. Ex. 7.3.8(c) Contd...

$\vdash$ B x 0 # y y y y z # 0 0 B $\vdash$ B x **0** # y y y y z # 0 0 B
q_3 q_3

$\vdash$ B **x** 0 # y y y y z # 0 0 B $\vdash$ B 0 **0** # y y y y z # 0 0 B
q_3 q_0

$\vdash$ B 0 x **#** y y y y z # 0 0 B $\vdash$ B 0 x # **y** y y y z # 0 0 B
q_1 q_2

$\vdash$ B 0 x # **y** y y y z # 0 0 B $\vdash$ B 0 x # y y **y** y z # 0 0 B $\vdash$ B 0 x y y y **y** z # 0 0 B
q_2 q_2 q_2

$\vdash$ B 0 x # y y y y **z** # 0 0 B $\vdash$ B 0 x # y y y y z # 0 0 B
q_2 q_2

$\vdash$ B 0 x # y y y y **z** # 0 0 B $\vdash$ B 0 x # y y y **y** 0 # 0 0 B
q_6 q_6

$\vdash$ B 0 x # y y **y** B 0 # 0 0 B $\vdash$ B 0 x # y **y** B B 0 # 0 0 B
q_6 q_6

$\vdash$ B 0 x # **y** B B B 0 # 0 0 B $\vdash$ B 0 x # B B B B 0 # 0 0 B
q_6 q_6

$\vdash$ B 0 **x** B B B B B 0 # 0 0 B $\vdash$ B **0** B B B B B B 0 # 0 0 B
q_6 q_6

$\vdash$ **B** B B B B B B B 0 # 0 0 B $\vdash$ **B** B B B B B B B B 0 # 0 0 B
q_6 q_7 remainder quotient

Fig. Ex. 7.3.8(c)

Example 7.3.9

Design a TM to increment a binary number by 1.

Solution : To increment a binary number by 1, following steps are required :

1. Trailing 1's should become 0.
2. First '0' from right end should become 1.
3. Remaining bits will remain as it is.

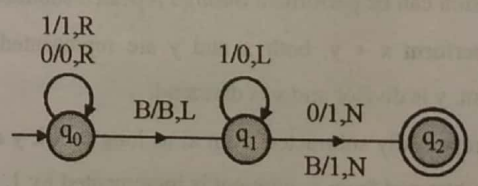


Fig. Ex. 7.3.9(a) : Transition diagram

	0	1	B	
$\rightarrow q_0$	$(q_0,0,R)$	$(q_0,1,R)$	(q_1,B,L)	
q_1	$(q_2,1,N)$	$(q_1,0,L)$	$(q_2,1,N)$	
eq_2^*	q_2	q_2	q_2	$\leftarrow$ Halting state

Fig. Ex. 7.3.9(b) : Transition table

– State q_0 moves the head to the first blank on the right side.

Scanned by CamScanner

- A transition from q_0 to q_1, positions the head on the rightmost 0 or 1.
- Trailing 1's are written as 0's in state q_1.
- A transition from q_1 to q_2 changes the first '0' or a blank from the right end towards left to 1.
- q_2 is halting state.
- The working of TM being simulated for 01010111, is shown in Fig. Ex. 7.3.9(c).

```
B01010111B⊢B01010111B⊢B01010111B
 ↑q0        ↑q0        ↑q0
⊢B01010111B⊢B01010111B⊢B01010111B⊢B01010111B
   ↑q0        ↑q0        ↑q0        ↑q0
⊢B01010111B⊢B01010111B⊢B01010111B
     ↑q0        ↑q0        ↑q1
⊢B01010110B⊢B01010100B⊢B01010000B
     ↑q1        ↑q1        ↑q1
⊢B01011000B
   ↑q2
```

Fig. Ex. 7.3.9(c)

Example 7.3.10

Design a TM to add two unary numbers.

Solution : Addition of two unary numbers can be performed through append operation. To add two numbers 5 (say ω_1) and 3 (say ω_2) will require following steps :

1. Initial configuration of tape :

B	0	0	0	0	0	#	0	0	0	B ...

$5(\omega_1)$ $3(\omega_2)$

2. ω_1 is appended to ω_2.

B	0	0	0	0	0	0	0	0	B	....

ω_2

While every '0' from ω_1 is getting appended to ω_2, '0' from ω_1 is erased. ω_2 contains 8 0's, which is sum of 5 and 3.

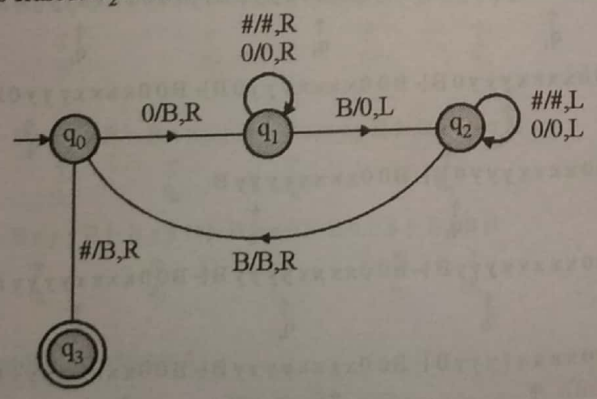


Fig. Ex. 7.3.10(a) : Transition diagram

	0	#	B	
$\rightarrow q_0$	(q_1,B,R)	(q_3,B,R)	–	
q_1	$(q_1,0,R)$	$(q_1,\#,R)$	$(q_2,0,L)$	
q_2	$(q_2,0,L)$	$(q_2,\#,L)$	(q_0,B,R)	
q_3^*	q_3	q_3	q_3	← Halting state

Fig. Ex. 7.3.10(b) : Transition table

The turing machine M is given by :

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

Where
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, \#\}$
$\Gamma = \{0, \#, B\}$
δ = Transition function is given in Fig. Ex. 7.3.10 (a), (b)
q_0 = initial state
B = blank symbol
F = $\{q_3\}$, halting state.

Example 7.3.11

Design a TM to find the GCD (greatest common divisor) of two unary numbers.

Solution : Two numbers x and y are stored on the tape without any separator. Head is positioned on the last digit of x.

Algorithm for finding GCD is based on repeated subtraction of x from y if y ≥ x or y from x if y < x. ultimately either x or y will become 0.

- If x becomes 0 then y is GCD
- If y becomes 0 then x is GCD

Algorithm for finding GCD is given below in C-language :

```
while(x != 0 && y != 0)
{           if(x >= y)
                x = x – y;
            else
                y = y – x;
}
if(x > 0)
    printf (" % d",x);
else
    printf (" % d",y) ;
```

Situation of tape for x = 6 and y = 4 is shown in Fig. Ex. 7.3.11 :

Initially B 000000 0000 B

x y

q_0

Fig. Ex. 7.3.11

1st cycle :

Subtraction is carried out through several cycles. In each cycle, the last '0' of 'x' is written as x and the first '0' of y is written as Y.

Scanned by CamScanner

1. B00000 X Y 000 B
2. B0000 XX YY 00B
3. B000 XXX YYY 0B
4. B00 XXXX YYYY B

Since, x is greater than y, y will end first.

At this point Y's are erased and X's are written as 0's

B 00 0000 B
x↑ y
q_0

2nd cycle

1. B0XY000B
2. BXXYY00B

B 00 00 B
x↑ y
q_0

Since, y is greater than x, x will end first.

At this point, X's are erased and Y's are written as 0's.

B 00 00 B
x↑ y
q_0

3rd cycle

1. B0XY0B
2. BXXYYB

Since, y is equal to x, both x and y, have ended at the same time. At this point, x's are erased and Y's are written as 0's. The tape contains the final result.

B00B
result

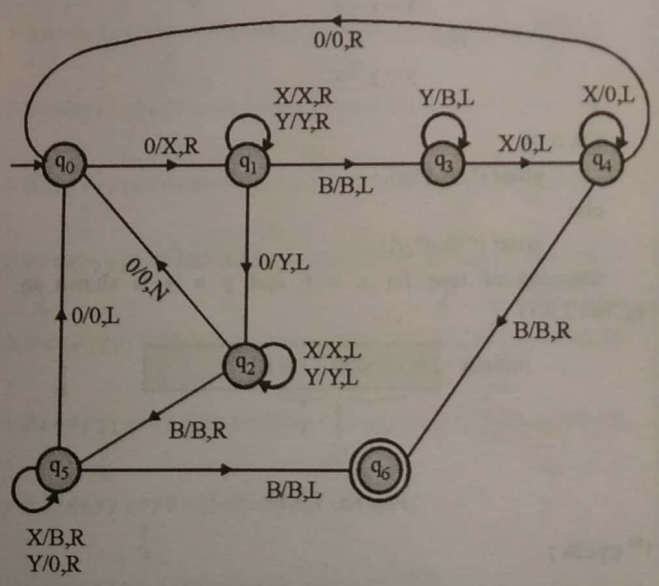


Fig. Ex. 7.3.11(a) : Transition diagram

The transition diagram of the TM is given in Fig. Ex. 7.3.11(a).

The TM can be defined as :

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

Where
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
$\Sigma = \{0\}$
$\Gamma = \{0, X, Y, B\}$
δ = Transition function is given in Fig. Ex. 7.3.11(a)
q_0 = Initial state
B = Blank symbol
F = $\{q_6\}$, halting state

Working of TM for GCD (6, 4) is shown in Fig. Ex. 7.3.11(b)

6 is represented as 000000

4 is represented as 0000

Initial configuration B000000 0000B (6, 4), head is positioned on the last digit of 6.
↑ q_0

B00000**0**0000B ⊢ B00000x**0**000B ⊢ B00000**x**y000B
↑ q_0 ↑ q_1 ↑ q_2

⊢ B00000**0**xy000B ⊢ B0000x**x**y000B ⊢ B0000xx**y**000B
↑ q_2/q_0 ↑ q_1 ↑ q_1

⊢ B0000xx**y**000B ⊢ B0000xx**y**y00B ⊢ B0000x**x**yy00B
↑ q_1 ↑ q_2 ↑ q_2

⊢ B0000**x**xyy00B ⊢ B0000**0**xyy00B ⊢ B0000**0**xyy00B
↑ q_2 ↑ q_2 ↑ q_0

⊢ B000x**x**xyy00B ⊢ B000xx**x**yy00B ⊢ B000xxx**y**y00B
↑ q_1 ↑ q_1 ↑ q_1

⊢ B000xxx**y**y00B ⊢ B000xxxy**y**00B ⊢ B000xxxyy**y**0B
↑ q_1 ↑ q_1 ↑ q_2

⊢ B000xxxy**y**y0B ⊢ B000xx**x**yyy0B ⊢ B000x**x**xyyy0B
↑ q_2 ↑ q_2 ↑ q_2

⊢ B000**x**xxyyy0B ⊢ B00**0**xxyyy0B ⊢ B00**0**xxyyy0B
↑ q_2 ↑ q_2 ↑ q_0

⊢ B00x**x**xxyyy0B ⊢ B00xx**x**xyyy0B ⊢ B00xxx**x**yyy0B
↑ q_1 ↑ q_1 ↑ q_1

⊢ B00xxxx**y**y0B ⊢ B00xxxxy**y**y0B ⊢ B00xxxxyy**y**0B
↑ q_1 ↑ q_1 ↑ q_1

⊢ B00xxxxyy**0**B ⊢ B00xxxxyy**y**B
↑ q_1 ↑ q_2

⊢ B00xxxxy**y**yyB ⊢ B00xxxx**y**yyyB ⊢ B00xxx**x**yyyyB
↑ q_2 ↑ q_2 ↑ q_2

⊢ B00xx**x**xyyyyB ⊢ B00x**x**xxyyyyB ⊢ B00**x**xxxyyyyB
↑ q_2 ↑ q_2 ↑ q_2

Fig. Ex. 7.3.11(b) contd....

Scanned by CamScanner

$\vdash$ B**0**xxxxyyyyB $\vdash$ B0**0**xxxxyyyyB $\vdash$ B0x**x**xxxxyyyyB
$\uparrow q_2$ $\uparrow q_0$ $\uparrow q_1$

$\vdash$ B0xx**x**xxyyyyB $\vdash$ B0xxx**x**xyyyyB $\vdash$ B0xxxx**x**yyyyB
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_1$

$\vdash$ B0xxxxx**y**yyyB $\vdash$ B0xxxxx**y**yyyB $\vdash$ B0xxxxxy**y**yyB
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_1$

$\vdash$ B0xxxxxyy**y**yB $\vdash$ B0xxxxxyyyy**B** $\vdash$ B0xxxxxyyyyB
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_3$

$\vdash$ B0xxxxxyy**y**B $\vdash$ B0xxxxxy**y**B $\vdash$ B0xxxxx**y**B
$\uparrow q_3$ $\uparrow q_3$ $\uparrow q_3$

$\vdash$ B0xxxxx**x**B $\vdash$ B0xxxx**x**0B $\vdash$ B0xx**x**00B
$\uparrow q_3$ $\uparrow q_4$ $\uparrow q_4$

$\vdash$ B0x**x**000B $\vdash$ B0**x**0000B $\vdash$ B**0**00000B $\vdash$ B0**0**0000B (x, y)
$\uparrow q_4$ $\uparrow q_4$ $\uparrow q_4$ $\uparrow q_0$

$\vdash$ B0x**0**000B $\vdash$ B0**x**y000B $\vdash$ B**0**xy000B
$\uparrow q_1$ $\uparrow q_2$ $\uparrow q_2$

$\vdash$ B**0**xy000B $\vdash$ Bx**x**y000B $\vdash$ Bxx**y**000B $\vdash$ Bxxy**0**00B
$\uparrow q_0$ $\uparrow q_1$ $\uparrow q_1$ $\uparrow q_1$

$\vdash$ Bxx**y**y00B $\vdash$ Bx**x**yy00B $\vdash$ B**x**xyy00B
$\uparrow q_2$ $\uparrow q_2$ $\uparrow q_2$

$\vdash$ **B**xxyy00B $\vdash$ B**x**xyy00B $\vdash$ B**x**yy00B $\vdash$ B**y**y00B
$\uparrow q_2$ $\uparrow q_5$ $\uparrow q_5$ $\uparrow q_5$

$\vdash$ B0**y**00B $\vdash$ B000**0**0B $\vdash$ B0**0**00B (x, y)
$\uparrow q_5$ $\uparrow q_5$ $\uparrow q_0$

$\vdash$ B0x**0**0B $\vdash$ B0**x**y0B $\vdash$ B**0**xy0B $\vdash$ B**0**xy0B
$\uparrow q_1$ $\uparrow q_2$ $\uparrow q_2$ $\uparrow q_0$

$\vdash$ Bx**x**y0B $\vdash$ Bxx**y**0B $\vdash$ Bxxy**0**B $\vdash$ Bxx**y**yB
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_1$ $\uparrow q_2$

$\vdash$ Bx**x**yyB $\vdash$ B**x**xyyB $\vdash$ **B**xxyyB $\vdash$ B**x**xyyB
$\uparrow q_2$ $\uparrow q_2$ $\uparrow q_2$ $\uparrow q_5$

$\vdash$ B**x**yyB $\vdash$ B**y**yB $\vdash$ B0**y**B $\vdash$ B00**B**B $\vdash$ B00B
$\uparrow q_5$ $\uparrow q_5$ $\uparrow q_5$ $\uparrow q_5$ $\uparrow q_6$

Result = 00, which is 2.

Fig. Ex. 7.3.11(b)

Example 7.3.12

Design a TM to find the value of $\log_2(n)$, where n is any binary number.

Solution : $\log_2(n)$ of any number n lying between 2^n and 2^{n+1} is given by n.

i.e. if $2^n \le n < 2^{n+1}$, then $\log_2(n) = n$

Let us consider the case of a number

$$n = 36$$
$$2^5 \le 36 < 2^6$$

Therefore, $\log_2(36) = 5$

36 can be written as 100100.

- Any number n satisfying the condition $2^5 \le n < 2^6$ can be written as 1XXXXX (where X stands for either 1 or 0).
- $\log_2$ (1XXXXX) can be calculated by erasing the most significant bit 1 and renaming other bits as '0'. Unary representation of 5 is 00000.

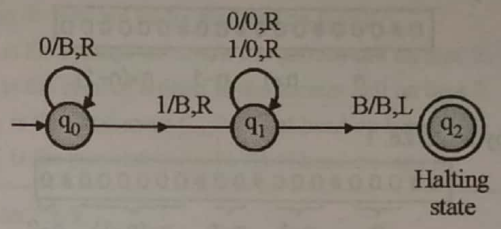


Fig. Ex. 7.3.12(a) : Transition diagram

	0	1	B	
$\rightarrow q_0$	(q_0,B,R)	(q_1,B,R)	–	
q_1	$(q_1,0,R)$	$(q_1,0,R)$	(q_2,B,L)	
q_2^*	q_2	q_2	q_2	← Halting state

Fig. Ex. 7.3.12(b) : Transition table

Working of TM for $(36)_{10}$ is shown in Fig. Ex. 7.3.12(c) :

$$(36)_{10} = (0100100)_2$$

B**0**100100B $\vdash$ B**1**00100B $\vdash$ B**0**0100B
$\uparrow q_0$ $\uparrow q_0$ $\uparrow q_1$

$\vdash$ B0**0**100B $\vdash$ B00**1**00B $\vdash$ B000**0**0B
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_1$

$\vdash$ B00000B $\vdash$ B00000**B** $\vdash$ B00000**0**B
$\uparrow q_1$ $\uparrow q_1$ $\uparrow q_2$

Fig. Ex. 7.3.12(c)

Example 7.3.13

Design a Turing machine to compute n!.

Solution : It is assumed that n is represented in unary system. Factorial of n can be calculated through repeated application of :

1. Multiplication
2. Copy Operations.

Algorithm is being explained with the help of example.

Scanned by CamScanner

Algorithm for 3 .

Initial configuration

0 # 0 0 0 # B B ...

n

Cycle 1

1. Multiplication 0 # 0 0 0 # 0 0 0 B ...

Product

2. Copy n – 1, i.e. 2 0 # 0 0 0 # 0 0 0 # 0 0

n n×1 n–1

Cycle 2

1. Multiplication

0 # 0 0 0 # 0 0 0 # 0 0 # 0 0 0 0 0 0

n n×1 n–1 n×(n–1)

2. Copy n – 2, i.e. 1

0 # 0 0 0 # 0 0 0 # 0 0 # 0 0 0 0 0 0 # 0

n n×1 n–1 n×(n–1) n–2

Cycle 3

1. 0 # 0 0 0 # 0 0 0 # 0 0 # 0 0 0 0 0 0 # 0 # 0 0 0 0 0 0 #

n 1×n n–1 n×(n–1) n–2 n×(n–1)(n–2)

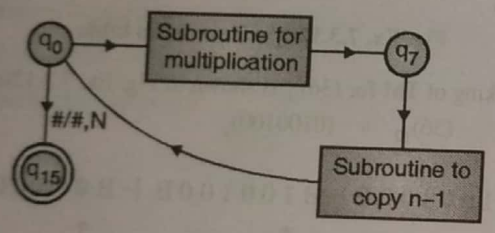


Fig. Ex. 7.3.13(a)

Subroutine for multiplication

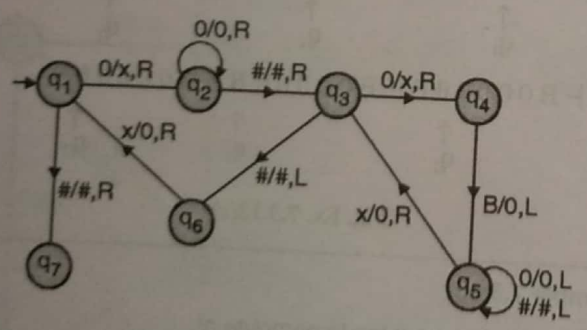


Fig. Ex. 7.3.13(b)

Subroutine to copy n – 1

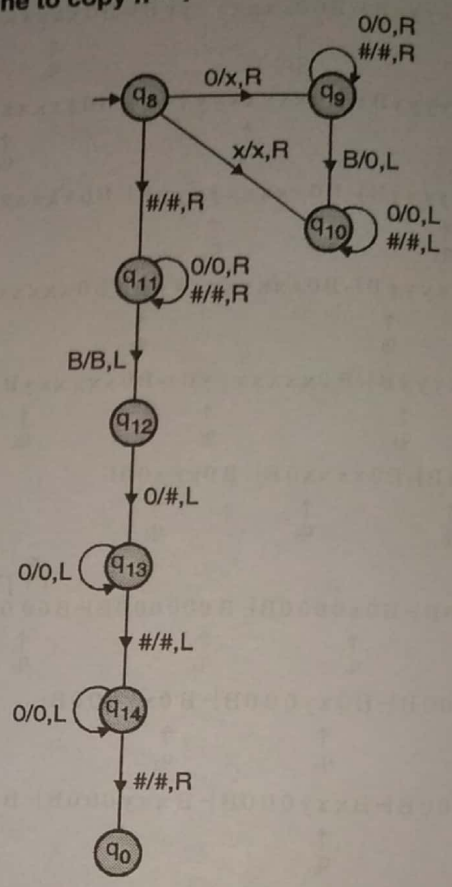


Fig. Ex. 7.3.13(c)

Example 7.3.14 MU - Dec. 2014, 10 Marks

Design a Turing machine to generate the language given by a regular expression : 00*

Solution : Turing machine starts with the first word as 0. This word will appear as # 0 #

(i) The machine in each cycle copies the previous word and appends 0 # to generate the next word of the carry 00*.

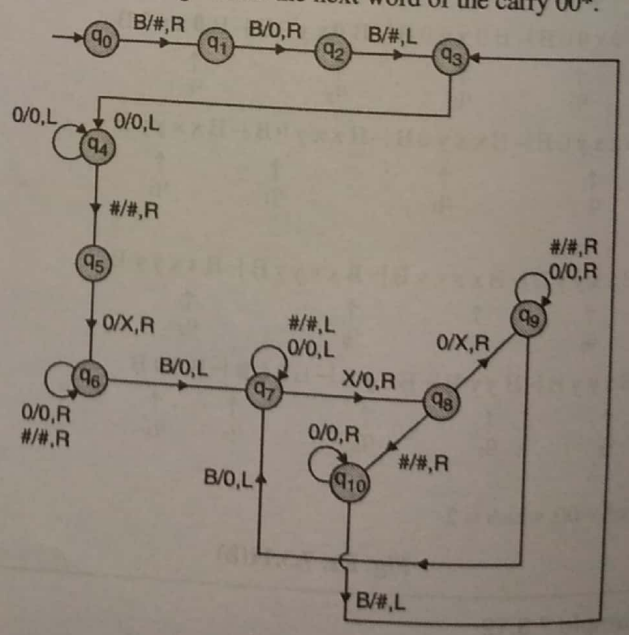


Fig. Ex. 7.3.14

Scanned by CamScanner

7.4 Extension of Turing Machine

➔ **(May 2014, May 2015)**

Q. Write short note on : Variants of a Turing Machine.
MU - May 2014, May 2015, 5/10 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 tapes with each tape having its own independent head.
4. K-dimension tape.
5. Non-deterministic turing machine.

7.4.1 Two-way Infinite Turing Machine

In a standard turing machine number of positions for leftmost blanks is fixed and they are included in instantaneous description, where the right-hand blanks are not included.

In the two way infinite Turing machine, there is an infinite sequence of blanks on each side of the input string. In an instantaneous description, these blanks are never shown.

7.4.2 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.

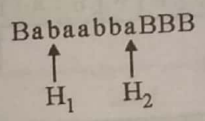


Fig. 7.4.1 : A Turing machine with two heads

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.

Syllabus Topic : Multi-track and Multi-tape TMs

7.4.3 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. 7.4.2.

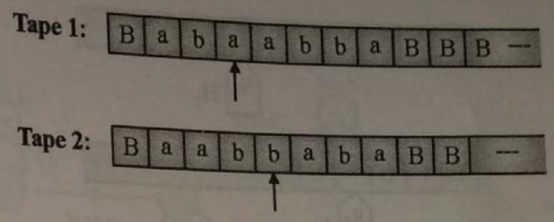


Fig. 7.4.2 : A two-tape turing machine

The transition behavior of a two-tape Turing machine can be defined as :

$$\delta(q_1,a_1,a_2) = (q_2,(S_1,M_1),(S_2,M_2))$$

Where, q_1 is the current state,

q_2 is the next state,

a_1 is the symbol under the head on tape 1,

a_2 is the symbol under the head on tape 2,

S_1 is the symbol written in the current cell on tape 1,

S_2 is the symbol written in the current cell on tape 2,

M_1 is the movement (L, R, N) of head on tape 1,

M_2 is the movement (L, R, N) of head on tape 2.

Example 7.4.1

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. 7.4.1(a).

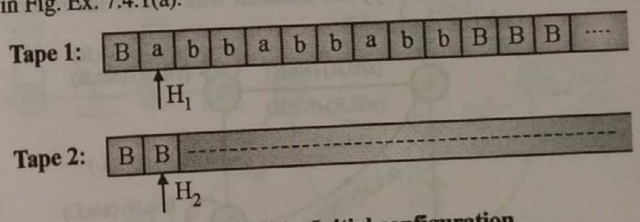


Fig. Ex. 7.4.1(a) : Initial configuration

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.

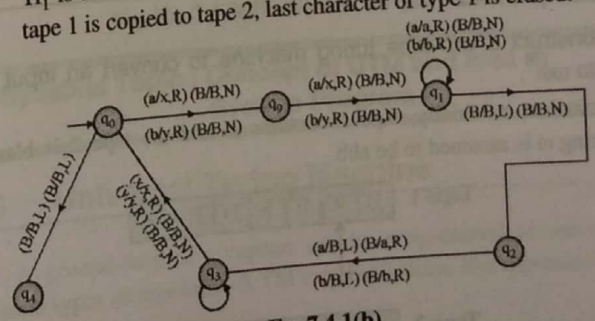


Fig. Ex. 7.4.1(b)

Contents of tapes after step 1 are shown in Fig. Ex. 7.4.1(c)

Scanned by CamScanner

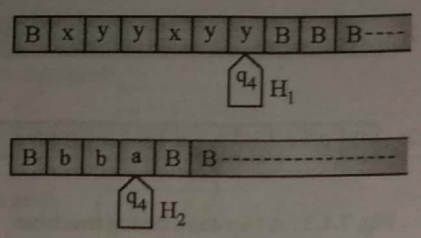


Fig. Ex. 7.4.1(c) : Last ω from tape 1 is copied as ω^R to tape 2

Step 2 : H_2 is moved to leftmost character on tape 2.

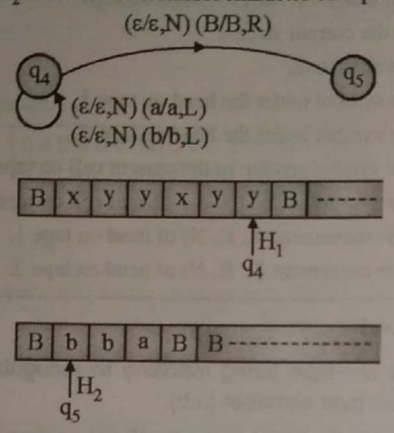


Fig. Ex. 7.4.1(d) : Situation after step 2

Step 3 : Two sets of xyy are matched with bba in the reverse order.

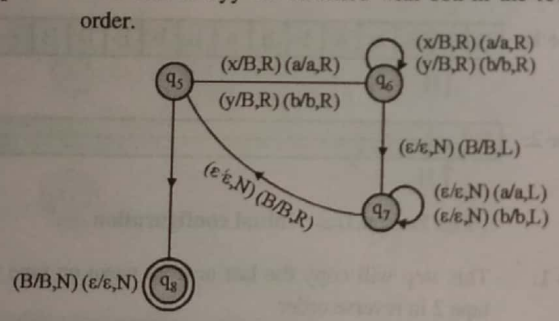


Fig. Ex. 7.4.1(e) : Two sets of xyy are matched with bba in the reverse order

Example 7.4.2

Construct a two-tape turing machine to convert an input ω into $\omega\omega^R$.

Solution : Initially, tape 1 contains ω and the tape 2 is blank String ω is assumed to be abb.

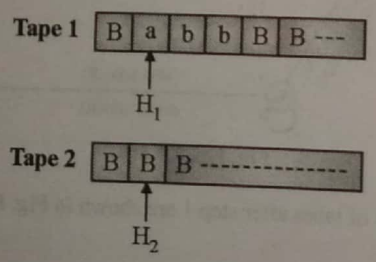
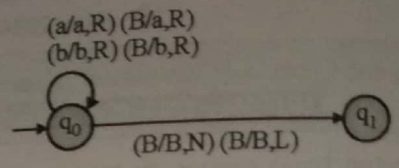


Fig. Ex. 7.4.2(a) : Initial configuration

Step 1 : String abb is copied to tape 2.



Contents of tapes after step1 are shown in Fig. Ex. 7.4.2(b).

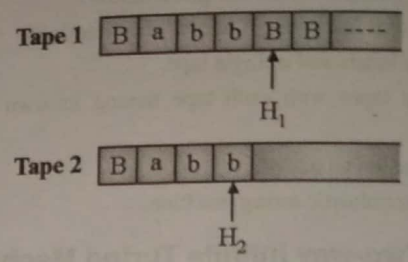
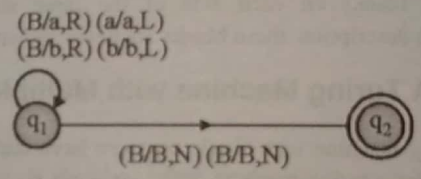


Fig. Ex. 7.4.2(b) : Contents of tapes after step 1

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. 7.4.2(c)

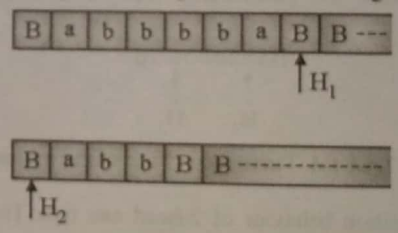


Fig. Ex. 7.4.2(c) : Contents of tapes after step 2

Transition diagram of the turing machine is shown in Fig. Ex. 7.4.2(d)

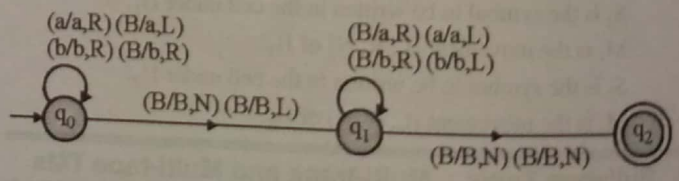


Fig. Ex. 7.4.2(d) : Transition diagram for Example 7.4.2

7.4.4 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.

Scanned by CamScanner

- 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 in Fig. 7.4.3.

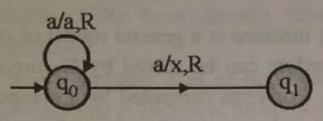


Fig. 7.4.3 : A sample move for NDTM

The transition behaviour for state q_0 for TM of Fig. 7.4.3 can be written as

$$\delta(q_0, a) = \{(q_0, a, R)(q_1, x, R)\}$$

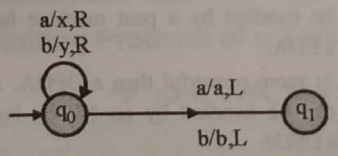
Example 7.4.3

Construct NDTM to recognize words of the form ωω over alphabet {a,b}

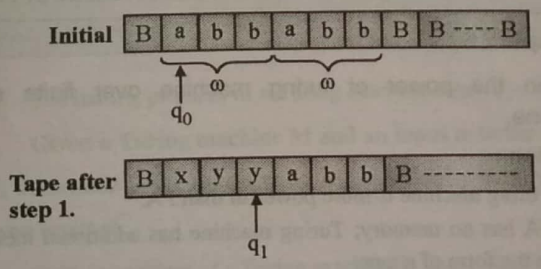
Solution :

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 are shown in Fig. Ex. 7.4.3(b).



(a)



(b)

Fig. Ex. 7.4.3

2. Head is sent back to the first character and then first half is matched with the second half while matching the two ω's every character of second ω is re-written as z.

- 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.

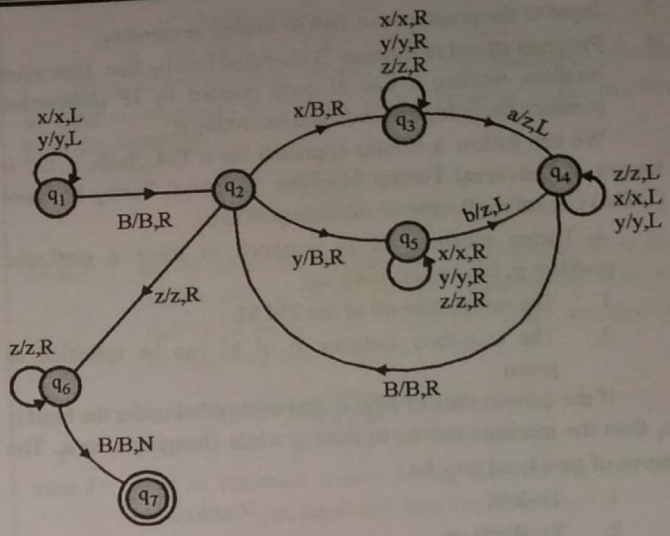


Fig. Ex. 7.4.3(c)

- 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. 7.4.3(d).

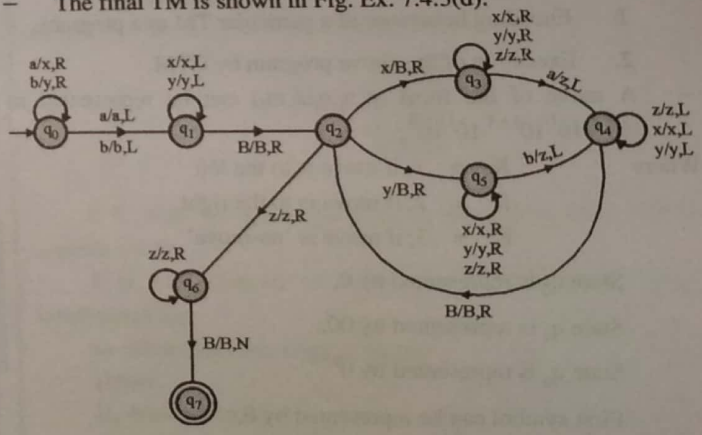


Fig. Ex. 7.4.3(d) : NDTM for ωω

Syllabus Topic : Concept of UTM and Idea of System Program

7.5 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.

Scanned by CamScanner

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 q0 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.

7.5.1 Church-Turing Hypothesis

The Turing machine is a general model of computation. Any algorithmic procedure can be solved by G computer can also be solved by a TM. Problems computed by a computer or a TM are also known as partial recursive functions. Some enhancements to TM made the Church-Turing thesis acceptable. These enhancements are :

1. Multi-tape
2. Multi-head
3. Infinite tapes
4. Non-determinism.

Since the introduction of TM, no one has suggested an algorithm than can be solved by a computer but cannot be solved by a TM.

7.5.2 Power of Various Machines

- A post machine can simulate functioning of a turing machine. Hence a turing machine and a post machine have equal power.
- A post machine is more powerful than a PDA. A string of the $a^n b^n c^n$ can be handled by a post machine but it cannot be handled by a PDA.
- An NPDA is more powerful than a DPDA. A string of the form $\omega\omega^R$ can be handled by an NPDA but it cannot be handled by a DPDA.
- DPDA is more powerful than FA. A string of the form $a^n b^n$ can be handled by a DPDA but it cannot be handled by FA.

Example 7.5.1

Explain the power of turing machine over finite state machine.

Solution :

1. Turing machine is more powerful than FA.
2. FA has no memory; Turing machine has additional memory in the form of a tape.
3. FA cannot modify its input. A TM can modify its own input.
4. FA cannot be used for arithmetic operations. A TM can perform arithmetic operations.
5. A language accepted by finite automata is regular language. It cannot handle context free language, context sensitive or recursive languages. These languages can be handled by a TM.

7.6 Un-decidability

A problem is said to be decidable if there exists a Turing machine that gives the correct answer for every statement in the

Scanned by CamScanner

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.

Syllabus Topic : Issue and Concept of Halting Problem

7.6.1 Halting Problem of a Turing Machine

(Dec. 2014, May 2015, Dec. 2016)

Q. Write short note on : Halting problem.

MU - Dec. 2014, May 2015, 10 Marks

Q. Define halting problem. MU - Dec. 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.

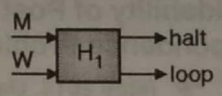
Proof

- Moves of a turing machine can be represented using a binary number. Thus, a Turing machine can be represented using a string over Σ*(0,1). This concept has already been explained in the chapter.
- Insolvability of halting problem of a Turing machine can be proved through the method of contradiction.

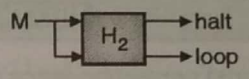
Step 1 : Let us assume that the halting problem of a Turing machine is solvable. There exists a machine H_1(say). H_1 takes two inputs :

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.

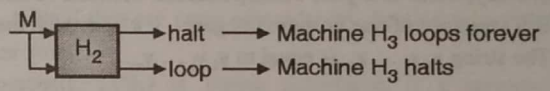


Step 2 : Let us revise the machine H_1 as H_2 to take M as both inputs and H_2 should be able to determine if M will halt on M as its input. Please note that a machine can be described as a string over 0 and 1.



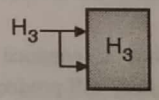
Step 3 : Let us construct a new Turing machine H_3 that takes output of H_2 as input and does the following :

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 case, 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 7.6.1

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.

Scanned by CamScanner

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.

7.6.2 Un-decidability of Post Correspondence Problem

➔ **(May 2014, Dec. 2014, Dec. 2016)**

Q. Write short notes on : Post Correspondence Problem
MU - May 2014, Dec. 2014, 5 Marks

Q. Explain post correspondence problem.
MU - Dec. 2016, 7 Marks

☞ **Definition of post correspondence problem (PCP)**

Let A and B be two non-empty lists of strings over Σ. A and B are given as below :

$$A = \{x_1, x_2, x_3 \dots x_k\}$$
$$B = \{y_1, y_2, y_3 \dots y_k\}$$

We say, there is a post correspondence between A and B if there is a sequence of one or more integers i, j, k …m such that :

The string $x_i\, x_j \dots x_m$ is equal to $y_i\, y_j \dots y_m$.

☞ **Example :** Does the PCP with two lists :

$$A = \{a, aba^3, ab\} \text{ and}$$
$$B = \{a^3, ab, b\}$$

have a solution ?

We will have to find a sequence using which when the elements of A and B are listed, will produce identical strings.

The required sequence is (2, 1, 1 , 3)

$$A_2\, A_1\, A_1\, A_3 = aba^3\, a\, aab = ab\, a^6\, b$$
$$B_2\, B_1\, B_1\, B_3 = aba^3\, a^3\, b = aba^6\, b$$

Thus, the PCP has solution.

We are accepting the un-decidability of post correspondence problem without proof.

Example 7.6.2

Determine the solution for following instance of PCP.

	List A	List B
i	ωi	xi
1	01	0
2	110010	0
3	1	1111
4	11	01

Solution :

The PCP has a solution. The required sequence is (1, 3, 2, 4, 4, 3)

$$\omega_1\omega_3\omega_2\omega_4\omega_4\omega_3 = 01111001011111$$
$$x_1\, x_3\, x_2\, x_4\, x_4 x_3 = 01111001011111$$

Example 7.6.3

Does PCP with following two list :

A = (10, 011, 101) and B = (101, 11, 011) have a solution ? Justify your answer.

Solution :

A_2 and A_3 differ from B_2 and B_3 at the first of place. Therefore, we must pick A_1 and B_1

Sequence	String
(1)	$(A_1 = 10)$ $(B_1 = 101)$

The next string to be picked up must be A_3 and B_3. Any other sequence will not lead to a solution.

Sequence	String
(1, 3)	$(A_1A_3 = 10101)$ $(B_1B_3 = 101011)$

The next string to be picked up must be A_3 and B_3. Any other sequence will not lead to a solution.

Sequence	String
(1, 3, 3)	$(A_1A_3A_3 = 10101101)$
	$(B_1B_3B_3 = 101011011)$

There is only choice of next string. This choice is A_3 and B_3. This does not lead to a solution. The PCP has no solution.

Example 7.6.4

Prove that there exists no algorithm for deciding whether a given CFG is ambiguous.

Solution : The post correspondence problem can be used to prove the un-decidability of whether a given CFG is ambiguous.

Let us consider two sequences of strings over Σ.

$$A = \{u_1, u_2, u_3 \dots u_m\}$$
$$B = \{v_1, v_2, v_3 \dots v_m\}$$

Let us take a new set of symbols $a_1, a_2 \dots a_m$ such that

$$\{a_1, a_2 \dots a_m\} \cap \Sigma = \phi.$$

Symbols $a_1, a_2 \dots a_m$ are being taken as index symbols. The index symbol a_i represents a choice of u_i from A and v_i from the list B.

A string of the form $u_i\, u_j\, u_k \dots a_k\, a_j\, a_i$. Over alphabet $\Sigma \cup \{a_1, a_2, \dots a_m\}$ can be defined using the set of productions :

$$G_A = \left\{\begin{matrix} A \to u_1\, A\, a_1 \mid u_2\, A\, a_2 \mid \dots \mid u_m\, Aa_m \\ u_1\, a_1 \mid u_2\, a_2 \mid \dots \mid u_m\, a_m \end{matrix}\right\}$$

Similarly a string of the form $v_i\, v_j\, v_k \dots a_k\, a_j\, a_i$ over alphabet $\Sigma \cup \{a_1, a_2 \dots a_m\}$ can be defined using the set of productions :

$$G_B = \left\{\begin{matrix} B \to v_1\, A\, a_1 \mid v_2\, A\, a_2 \mid \dots \mid v_m\, Aa_m \\ v_1\, a_1 \mid v_2\, a_2 \mid \dots \mid v_m\, a_m \end{matrix}\right\}$$

Finally, we can combine the languages and grammars of two lists to form a grammar G_{AB} :

- A new start symbol S is added to G_{AB}
- Two new productions are added to G_{AB}

$$S \to A$$
$$S \to B$$

- All productions of G_A and G_B are taken.

Now, we will show that G_{AB} is ambiguous if and only if an instance (A, B) of PCP has a solution.

Assumption

Suppose the sequence $i_1, i_2, \dots, i_m$ is a solution to this instance of PCP. Two derivations for the above string in G_{AB} is :

Scanned by CamScanner

$S \Rightarrow A \Rightarrow u_{i_1} A a_{i_1} \Rightarrow u_{i_1} u_{i_2} A a_{i_1} a_{i_2} \Rightarrow \ldots \Rightarrow$

$u_{i_1} u_{i_2} \ldots u_{i_m} a_{i_1} a_{i_2} \ldots a_{i_m}$

$S \Rightarrow B \Rightarrow v_{i_1} B a_{i_1} \Rightarrow v_{i_1} v_{i_2} B a_{i_1} a_{i_2} \Rightarrow \ldots \Rightarrow$

$v_{i_1} v_{i_2} \ldots v_{i_m} a_{i_1} a_{i_2} \ldots a_{i_m}$

Consequently, if G_{AB} is ambiguous, then the post correspondence problem with the pair (A, B) has a solution. Conversely, if G_{AB} is unambiguous, then the post correspondence cannot 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 7.6.5

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 un-decidable, the same must be true for the blank tape halting problem.

Example 7.6.6

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) $L(G_1) = L(r)$

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

$L(G_2) = \{u_1 u_2 \ldots u_n C v_n^R v_{n-1}^R \ldots v_1^R, \mid n \geq 1 \text{ and } (u_i, v_i) \in P\}$

- 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.

7.7 Exam Pack (University Questions)

Syllabus Topic : Deterministic TM

Q. Write short note on : Turing machines.

(Refer section 7.1) (10 Marks) **(Dec. 2014)**

Example 7.2.17(10 Marks) **(May 2014, May 2017)**

Scanned by CamScanner

Example 7.3.14 (10 Marks) **(Dec. 2014)**

☞ **Syllabus Topic : Issue and Concept of Halting Problem**

Q. Write short note on : Halting problem.

(Refer section 7.6.1) (10 Marks)

(Dec. 2014, May 2015)

Q. Define halting problem.

(Refer section 7.6.1) (5 Marks) **(Dec. 2016)**

Q. Write short notes on : Post Correspondence Problem

(Refer section 7.6.2) (5 Marks)

(May 2014, Dec. 2014)

Q. Explain post correspondence problem.

(Refer section 7.6.2) (7Marks) **(Dec. 2016)**

❑❑❑

Scanned by CamScanner

CHAPTER 8

Applications of Automata

Syllabus Topics

1. Power and Limitations of Regular and Context Free Grammars and Machines
2. Designing Functions:
   FA: Acceptor and Verifier.
   FSM : Translator
   PDA : Simple Parser for WF parenthesis, palindromes etc.
   TM : Basic bit wise calculator(+ /-/AND/OR) and Translator

Syllabus Topic : Power and Limitation of Regular and Context Free Grammars and Machines

8.1 Power and Limitation of Regular and Context Free Grammars and Machines

- A regular grammar can handle only a small subset of languages. A regular grammar is for regular language. The machine for regular language is Finite Automata.
- FA has no memory
- FA can not modify its input.
- FA cannot be used for arithmetic operations.
- FA cannot handle context free language, context sensitive or recursive language.
- Pumping lemma for regular language can be used to find the languages which can be accepted by Finite automata.

A context free language is represented using a context free grammar. PDA is a machine for context free language. Most of the programming languages fall in this category.

CFG cannot be used for computation. It cannot be used for recursive and context sensitive languages.

Syllabus Topic : Designing Functions

8.2 Designing Function

It is useful to understand the software implementation of FA, PDA and TM. Each state S of the machine can be replaced with the **function S(x)**, where x is the next input.

Syllabus Topic : FA as Acceptor and Verifier

8.2.1 FA as Acceptor and Verifier

An FA, M can be used to check whether a given string ω belongs to L(M). The transition function of an FA can be converted into software function as given below :

1. Each state of FA can be written as a function. For example, the state S of the FA can be written as S (x), where x is the next input.
2. Each transition can be written as a function call or a goto statement.
3. Beginning with the starting state, a string is accepted if it ends in a terminating state.

Example 8.2.1

Write the necessary functions for the given automata.

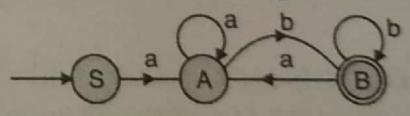


Fig. Ex. 8.2.1

Solution :

```
S(x)
  {
        if (x=='a') call A (next);
        else print ("error");
  }
A (x)
  {
        if(x =='a') call A (next);
        else if (x=='b') call B (next);
```

Scanned by CamScanner

```
        else print ("error")
    }
    B(x)
    {
        if (x=='a') call A (next);
        else if (x=='b') call B (next);
        else if (end of input) print ("Accept");
        else print ("Error");
    }
```

* A (next) – call function A( )with the next input

Syllabus Topic : FSM : Translator

8.2.2 FM as Translator

A finite machine Mealy and Moore machine can be used for translation of an input string. For the sake of simplicity, Mealy machine model is used for translation. The transition function of a mealy machine can be converted into software function as given below :

1. Each state of a Mealy machine can be written as a function. For example, the state S of the Mealy machine can be written as S(x), where x is the next input.
2. Each transition can be written as a function call or a goto statement.

Example 8.2.2

Write the necessary function for the given Mealy machine.

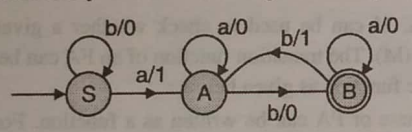


Fig. Ex. 8.2.2

Solution :

```
S(x)
  {
        if (x ='b') { print ("0"); call S(next);}
        else if(x='a'){Print("1"); call A(next);}
  }
A(x)
  {     if(x='a'){print("0"); call A(next);}
        else if(x='b') {print("0"); call B(next);}
  }
B(x)
  {     if (x='a'){print ("0"); call B(next); }
        else if(x='b') {print ("1"); call A (next);}
  }
```

Syllabus Topic : Simple Parser for WF Parenthesis, Palindromes etc.

8.2.3 Simple Parser using PDA

Transition function of a PDA can be implemented in software. Each transition of a PDA depends on the followings :

1. Current state.
2. Current input.
3. top of the stack symbol.

As explained for FA, each state of a PDA can be written as a software function.

Example 8.2.3

Write the necessary functions for a PDA for WF parenthesis.

Solution :

The transition diagram of the PDA is given in Fig. Ex. 8.2.3, where initial stack symbol is Z_o.

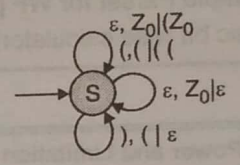


Fig. Ex. 8.2.3

String is accepted through an empty stack. In the software function, the string will be accepted. If at the end of the input the stack containing the symbol Z_o.

```
S(x)
  {     if (x= = '(' ) push ('(' );
        else if (x= ='j') and (stacktop( ) = = '(' )
                POP();
    else if (end of input) and (stacktop ( ) = ='Z0')
            print("Accept");
        else
            print ("error");
  }
```

Syllabus Topic : TM : Basic Bitwise Calculator (+ /–/AND/OR) and Translator

8.2.4 TM for Bitwise Operation

Transition function of a TM can be implemented in software. Each transition of a PDA depends on the following :

1. Current state.
2. Current input.
3. Symbol under the head.

The read/write tape of the TM can be represented as an array a[ ]. If i gives the current position of the head then left or write movement of the head can be implemented as i – – or i ++ respectively.

Scanned by CamScanner

Example 8.2.4

Design the necessary function to implement AND operation using a TM. Both the numbers are assumed to be of equal length. These numbers are separated by the symbol '#'.

Solution :

The initial condition of the tape is given below :

B <first input># <second input># **result.**

The required TM is given in Fig. Ex. 8.2.4.

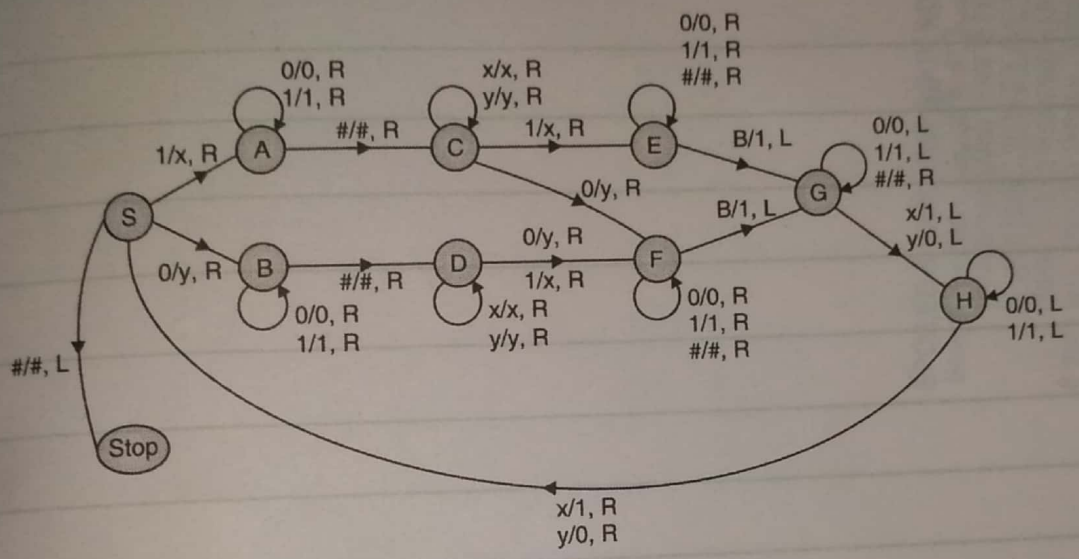


Fig. Ex. 8.2.4

The variable i gives the position of the head. It is initially set to 1.

```
S(i)
  { if (a[i]=='1') { a[i] = 'x'; i++; call A(i),}
    else if (a[i]=='0'){a[i]='y',i++; call B(i);}
    else if (a[i]=='#') stop;
    else print ("error");
  }
A(i)
  { if (a[i]=='0') {i++; call A(i);}
    else if(a[i]=='1'){ i++; call A(i);}
    else if (a[i]=='#') {i++; call C(i);}
  }
```

Other functions can be written in the similar fashion. □□□

Scanned by CamScanner

Automata Theory

(Semester IV - Information Technolgy (Mumbai University))

Dilip Kumar Sultania

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

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

t Edition : January 2018

edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated count
unauthorized by the publisher.

Scanned by CamScanner