

**THIRD YEAR B. Sc.
COMPUTER SCIENCE**

SEMESTER-V

**NEW SYLLABUS
CBCS PATTERN**

THEORETICAL COMPUTER SCIENCE

Dr. Ms. MANISHA BHARAMBE



SPPU New Syllabus

A Book Of

THEORETICAL COMPUTER SCIENCE

**For T.Y.B.Sc. Computer Science : Semester – V
[Course Code CS - 356 : Credits - 2]**

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Dr. Ms. Manisha Bharambe

M.Sc. (Comp. Sci.), M.Phil. Ph.D. (Comp. Sci.)

Vice Principal, Associate Professor, Department of Computer Science
MES's Abasaheb Garware College
Pune

Price ₹ 360.00

 **NIRALITM
PRAKASHAN**
ADVANCEMENT OF KNOWLEDGE

N5866

THEORETICAL COMPUTER SCIENCE**ISBN 978-93-5451-193-6****Second Edition : September 2022****© : Author**

The text of this publication, or any part thereof, should not be reproduced or transmitted in any form or stored in any computer storage system or device for distribution including photocopy, recording, taping or information retrieval system or reproduced on any disc, tape, perforated media or other information storage device etc., without the written permission of Author with whom the rights are reserved. Breach of this condition is liable for legal action.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may have crept in. Any mistake, error or discrepancy so noted and shall be brought to our notice shall be taken care of in the next edition. It is notified that neither the publisher nor the author or seller shall be responsible for any damage or loss of action to any one, of any kind, in any manner, there from. The reader must cross check all the facts and contents with original Government notification or publications.

Published By :**NIRALI PRAKASHAN**

Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39
Email : niralipune@pragationline.com

Polyplate**Printed By :****YOGIRAJ PRINTERS AND BINDERS**

Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

DISTRIBUTION CENTRES**PUNE****Nirali Prakashan****(For orders outside Pune)**

S. No. 28/27, Dhayari Narhe Road, Near Asian College
Pune 411041, Maharashtra
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

Nirali Prakashan**(For orders within Pune)**

119, Budhwar Peth, Jogeshwari Mandir Lane
Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

MUMBAI**Nirali Prakashan**

Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES**DELHI****Nirali Prakashan**

Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

BENGALURU**Nirali Prakashan**

Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

NAGPUR**Nirali Prakashan**

Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhanshi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

KOLHAPUR**Nirali Prakashan**

438/2, Bhosale Plaza, Ground Floor
Khasbag, Opp. Balgopal Talim
Kolhapur 416 012, Maharashtra
Mob : 9850046155
Email : kolhapur@niralibooks.com

JALGAON**Nirali Prakashan**

34, V. V. Golani Market, Navi Peth,
Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395
Mob : 94234 91860
Email : jalgaon@niralibooks.com

SOLAPUR**Nirali Prakashan**

R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

marketing@pragationline.com | www.pragationline.com**Also find us on  www.facebook.com/niralibooks**

Preface ...

I take an opportunity to present this Text Book on "**Theoretical Computer Science**" to the students of Third Year B.Sc. (Computer Science) Semester-V as per the New Syllabus, June 2021.

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts. The book covers theory of Finite Automaton, Regular Expressions and Languages, Context-free Grammars and Languages, Pushdown Automata and Turing Machine.

A special word of thank to Shri. Dineshbhai Furia, and Mr. Jignesh Furia for showing full faith in me to write this text book. I also thank to Mr. Amar Salunkhe and Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

I also thank Ms. Chaitali Takle for Graphic Designing, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

Author



Syllabus ...

1. Finite Automation

(10 Lectures)

- Introduction – Symbol, Alphabet, String, Prefix and Suffix of Strings, Formal Language, Operations on Languages.
- Deterministic Finite Automaton – Definition, DFA as Language Recognizer, DFA as Pattern Recognizer.
- Non-deterministic Finite Automaton – Definition and Examples.
- NFA To DFA (Myhill-Nerode Method)
- NFA with ϵ – Transitions Definition and Examples.
- NFA with ϵ – Transitions to DFA and Examples
- Finite Automaton with Output – Mealy and Moore Machine, Definition and Examples.
- Minimization of DFA, Algorithm and Problem using Table Method.

2. Regular Expressions and Languages

(06 Lectures)

- Regular Expressions (RE) – Definition and Examples.
- Regular Expressions Identities.
- Regular Language – Definition and Examples.
- Conversion of RE to FA – Examples.
- Pumping Lemma for Regular Languages and Applications.
- Closure Properties of Regular Languages.

3. Context-Free Grammars and Languages

(10 Lectures)

- Grammar – Definition and Examples.
- Derivation – Reduction – Definition and Examples.
- Chomsky Hierarchy.
- CFG – Definition and Examples. LMD, RMD, Parse Tree.
- Ambiguous Grammar – Concept and Examples.
- Simplification of CFG – Removing Useless Symbols, Unit Production, ϵ -Production and Nullable Symbol.
- Normal Forms – Greibach Normal Form (GNF) and Chomsky Normal Form (CNF).
- Regular Grammar – Definition.
- Left Linear and Right Linear Grammar – Definition and Example.
- Equivalence of FA and Regular Grammar.
- Construction of Regular Grammar Equivalent to a given DFA.
- Construction of a FA from the given Right Linear Grammar.

4. Pushdown Automata

(05 Lectures)

- Definition of PDA and Examples.
- Construction of PDA using Empty Stack and Final State Method – Examples using Stack Method.
- Definition DPDA and NPDA, their Correlation and Examples of NPDA.
- CFG (in GNF) to PDA – Method and Examples.

5. Turing Machine

(05 Lectures)

- The Turing Machine Model, Definition and Design of TM.
- Problems on Language Recognizers.
- Language Accepted by TM.
- Types of Turing Machines (Multitrack TM, Two-way TM, Multitape TM, Non-deterministic TM).
- Introduction to LBA (Basic Model) and CSG, (Without Problems).



Contents ...

1. Finite Automation **1.1 – 1.102**

2. Regular Expressions and Languages **2.1 – 2.36**

3. Context-Free Grammars and Languages **3.1 – 3.74**

4. Pushdown Automata **4.1 – 4.32**

5. Turing Machine **5.1 – 5.32**



Finite Automaton

Objectives ...

- To study Basic Concepts in Theoretical Computer Science
- To learn Finite Automata
- To understand Deterministic Finite Automata
- To learn Non-deterministic Finite Automaton

1.0 INTRODUCTION

- Theoretical computer science is a division of general computer science and mathematics. It focuses on more mathematical and abstract aspects of computing.
- Theoretical computer science includes the theory of computation. In this chapter, we introduce mathematical terms necessary for understanding the automata theory.

1.1 BASIC TERMS

- In this section we will study the basic terms used in theoretical computer science like symbol, alphabet, string, prefix and suffix of strings, formal language, operations on languages and so on.

Symbol:

- The symbol is the smallest building block in the theory of computation and can be any letter, number or even pictograms.
- Symbol is an abstract or a user-defined entity. It cannot be formally defined (like "point" in geometry).
- For example, digits (0, 1, 2, ...) and letters (a, b, c, ...).
- We can define many interesting things based on symbol. Using symbols, we can generate strings.

Alphabet (Σ):

- Alphabet is a finite set of symbols.
- Every element of an alphabet is called a symbol of alphabet Σ .

For examples: $\Sigma_D = \{0, 1, 2, \dots, 9\}$

$\Sigma_R = \{a, b, c, p, D, @ \}$

Here, set R is containing heterogeneous (not to the same type) entities, though it is an alphabet. The concept of alphabet is similar to the well-known alphabet set $\{a, b, c, \dots, z\}$ for the English language.

String:**[April 16, Oct. 16, 17]**

- A string is a finite ordered sequence of symbols from the alphabet.
- A string is defined as, a finite sequence of sequence of symbols from the alphabet Σ .
- For example, the a, b and c are symbols and abcb is a string.
- A string may have no symbols at all such string is called as **empty string** and is denoted by ϵ .
- The **length** of a string is number of symbols in the string. If string is ω ; its length is denoted as $|\omega|$ and $|\epsilon| = 0$, (length of null string = 0).

Prefix and Suffix of Strings:**[April 16, 18, 19, Oct. 16, 17, 18]**

- A prefix of a string is any number of leading symbols of that string.
- **For example**, let $x = abc$ the prefixes of x are ϵ , a, ab, abc.
- The prefix other than the string itself is called as proper prefix. In the above example, ϵ , a, ab are proper prefixes of abc.
- A suffix of a string is any number of trailing symbols in it.
- **For example**, In above example, suffixes of x are ϵ , c, bc, abc.
- The suffix other than the string itself is called a proper suffix. ϵ , c, bc are proper suffixes of abc.
- The **concatenation** of two strings is the string formed by writing the first followed by the second with no intervening space.
- **For example**, if w and x are strings then ' ωx ' is concatenation.

$$|\omega x| = |\omega| + |x|$$

- The length of the concatenated string is equal to the sum of the lengths of those strings. The empty string is the identity for the concatenation operator. So,

$$\epsilon \omega = \omega \epsilon = \omega$$

Formal Language:**[April 19]**

- Formal language is a set of strings of symbols from some alphabet.
- The empty set ϕ and the set consisting of empty string i.e. $\{\epsilon\}$ are also languages.
- Set of all strings over a fixed alphabet Σ is language denoted by Σ^* .

For example,

1. Let $\Sigma = \{a\}$ then
 $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\}$
2. Let $\Sigma = \{0, 1\}$ then
 $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$
3. Language over alphabet $\{a, b\}$ having equal number of a's and b's.
 $\Sigma^* = \{\epsilon, ab, ba, abab, baab, aabb, \dots\}$

- From the above language Σ^* , if we exclude ϵ , the empty word, the remaining set is a language denoted by Σ^+ .
- **For example**, $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, \dots\}$ for second example.

- The language we have defined above is the formal language. The word 'formal' refers to the fact that all the rules for the language are explicitly stated in terms of what strings of symbols can occur, and which are the valid sentences.
- Formal language will be considered as symbols on paper and not as expressions of ideas as in Natural language like English. The rules are called as formal rules.
- The term formal used here emphasizes that it is the form of the string of symbols, we are interested in, not the meaning.

Operations on Languages:

- In this point we will study various operations on languages.

Union:

- Union of languages L_1 and L_2 is the language (L) containing all strings of L_1 and all strings of L_2 .

$$L = L_1 \cup L_2$$

Examples:

- (i) If $L_1 = \{a, b\}$ and $L_2 = \{c, d\}$ then

$$L = L_1 \cup L_2 = \{a, b, c, d\} \text{ (finite set)}$$

- (ii) If $L_1 = a^*b$ and $L_2 = b^*$ then

$$L = L_1 \cup L_2$$

$$\text{Here, } L_1 = \{b, ab, aab, aaab \dots\}$$

$$L_2 = \{\epsilon, b, bb, bbb, \dots\}$$

$$L = L_1 \cup L_2 = \{\epsilon, b, ab, bb, aab, bbb, aaab \dots\} \text{ (infinite set)}$$

- (iii) If $L_1 = \{a^m b^m \mid m \geq 1\}$ and $L_2 = \{b^m \mid m \geq 0\}$

$$\text{Here, } L_1 = \{ab, aabb, aaabbb \dots\}$$

$$\text{and } L_2 = \{\epsilon, b, bb, bbb \dots\} \text{ then}$$

$$L = L_1 \cup L_2 = \text{containing strings having equal number of a's and b's and also having strings of } b^*.$$

Intersection:

- Intersection of languages L_1 and L_2 is the language L containing common strings of L_1 and L_2 .

$$L = L_1 \cap L_2$$

Examples:

- (i) If $L_1 = \{a, b\}$ and $L_2 = \{a, c\}$ then

$$L = L_1 \cap L_2 = \{a\}$$

- (ii) If $L_1 = \{aa, bb, aab\}$ and $L_2 = \{aa, aab\}$ then

$$L = L_1 \cap L_2$$

$$= \{aa, aab\}$$

- (iii) If $L_1 = a^*$ and $L_2 = b^*$ then

$$L = L_1 \cap L_2 = \emptyset \text{ (No string common)}$$

- (iv) If L_1 contains equal number of a's and b's over $\Sigma = \{a, b\}$ and $L_2 = a^* b^*$ find intersection of L_1 and L_2 .

Here, $L_1 = \text{EQUAL} = \{\epsilon, ab, ba, aabb, abab, abba, baab, baba, bbaa, aaabbb \dots\}$

and $L_2 = \{\epsilon, a, b, ab, aabb, aaabbb \dots\}$

$\therefore L_1 \cap L_2 = \{ab, aabb, aaabbb \dots\}$

$L = L_1 \cap L_2 = \{a^n b^n\}$

- (v) If L_1 containing all string starting with 'a' over $\{a, b\}$ and L_2 containing all strings ending with 'a' over $\{a, b\}$ then, $L = L_1 \cap L_2 =$ containing all string starting with 'a' and ending with 'a' and also only 'a'.

$\therefore L = \{a, aa, aba, abba, abbaa \dots\}$

Concatenation:

- The concatenation of languages L_1 and L_2 is the language L containing all strings of L_1 followed by all strings of L_2 without space.

$$L = L_1 L_2$$

Examples:

(i) $L_1 = \{a\}$

$L_2 = \{b\}$

$L = L_1 L_2 = \{ab\}$

(ii) $L_1 = \{a, aa, aaa\}$

$L_2 = \{bb, bbb\}$

Then, $L = L_1 L_2 = \{abb, abbb, aabb, aabbb, aaabb, aaabbb\}$

(iii) $L_1 = \{a, bb, bab\}$ and $L_2 = \{a, ab\}$

$L = L_1 L_2 = \{aa, aab, bba, bbab, baba, babab\}$

(iv) $L_1 = \{a, bb, bab\}$ and $L_2 = \{\epsilon, bbbb\}$

$L = L_1 L_2 = \{a, bb, bab, abbbb, bbbbbb, babbbbb\}$

(v) $L_1 = \{\epsilon, x\}$ and $L_2 = y^*$

Here, $L_2 = \{\epsilon, y, yy, yyy, \dots\}$

Which, is infinite set.

$L = L_1 L_2 = \{\epsilon, xy, xyy, xyxy, xyxyy, y, yy, yyy, \dots\}$

1.2 FINITE AUTOMATON

[Oct. 17]

- Usually, in theoretical computer science, we consider the mathematical models of machines which are the abstractions and simplifications of how certain actual machines do work.
- The word 'automaton' (plural - 'automata') is derived from the Greek word 'automatos' meaning 'self-acting'.
- Finite Automata or FA is considered as the mathematical model of finite state machine. A Finite State Machine (FSM) is a mathematical model of computation.

- FA consists of a finite set of states and a set of transitions from state to state that occur on input symbol chosen from an alphabet Σ .
- Computer itself can be viewed as a finite state system. The state of the central processor, main memory and auxiliary storage at any time is one of a very large but fixed number of states.
- Finite automaton is an abstract model (formally describe the behavior of the computer system) of a digital computer which has three components namely, Input tape, Control unit and Output.
- Using abstract model, the behavior of the actual system can be understood and build to perform various activities.
- The pictorial/graphical presentation/block diagram of finite automata is shown in Fig. 1.1. The FA contains:
 - **Input Tape:** It is divided into number of cells/blocks/squares. Each input symbol is placed in each cell. Each cell contains a single symbol from input alphabet Σ .
 - **Finite State Control:** The finite automaton has some states one of which is the start state designed as q_0 and at least one final state. Apart from these it has some finite states denoted by $q_1, q_2, q_3 \dots q_n$. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read. Based on the current input symbol, the state can be change.
 - **Output:** The output of FA may be accept or reject depending on the input. When end of the input is encountered, the control unit may be in accept or reject state.

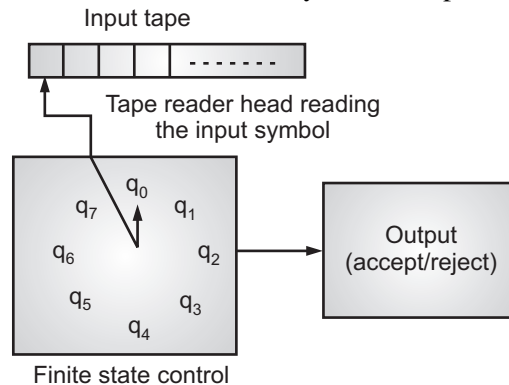


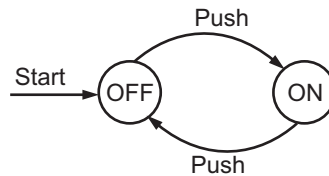
Fig. 1.1: Pictorial Presentation of Finite Automaton (FA)

Working of Finite Automaton:

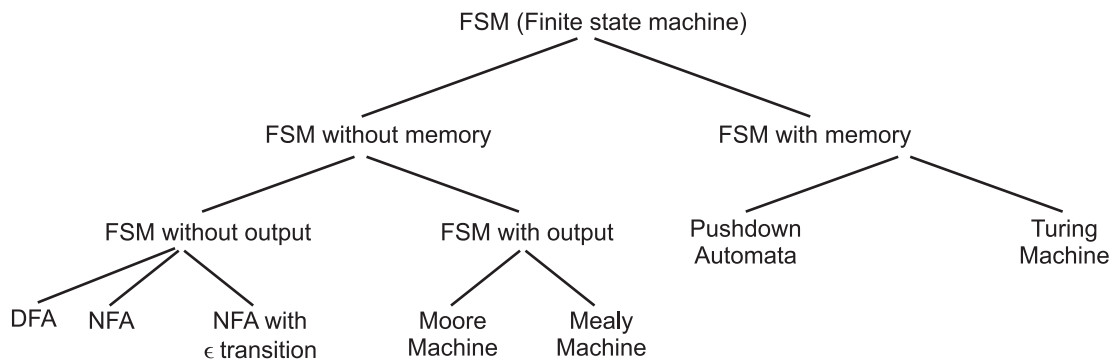
- Fig. 1.1 shows the working of the Finite Automation.
- The machine is assumed to be in start state q_0 .
- The input pointer points to the first cell of the tape pointing to the string to be processed.
- After scanning the current input symbol, the machine can enter into any of the states $q_0, q_1, q_2 \dots q_n$ the input pointer automatically points to the next character by moving one cell towards right.
- When the end of the string is encountered, the string is accepted if and only if the automation will be in one of the final states. Otherwise, the string is rejected.

Example of Finite Automaton:

- Consider an electric switch which has only two states "OFF" and "ON". To start with, the switch will be in OFF state. When we push the button, it goes to ON state. If we push once again it goes to OFF state (See Fig. 1.2).
- Note that the circles represent the states and the labels associated with arcs represent the input given to go to another state. The arrow with the label Start (not originating from any state) is considered as the start state.

**Fig. 1.2: Example of Finite Automaton (FA)**




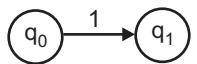
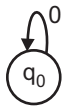

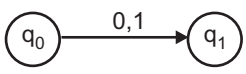
- An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM). Fig. 1.3 shows different forms/kinds of FSM.

**Fig. 1.3**

- Finite Automaton can be classified into two types namely, Deterministic Finite Automaton (DFA) and Non-deterministic Finite Automaton (NFA).
- In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called Deterministic Automaton.
- As it has a finite number of states, the machine is called Deterministic Finite Machine or Deterministic Finite Automaton.
- In NFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-deterministic Automaton.
- As it has finite number of states, the machine is called Non-deterministic Finite Machine or Non-deterministic Finite Automaton.

Symbols used in Finite Automaton:

- We use following notations/symbols for representing the FA:

Symbol	Description
	A circle is used to represent a state of the machine (here, q_0 is a state).
	A circle with an arrow which is not originating from any node represents the start state of machine.
	Two circles are used to represent a final state of the machine (here, q_0 is the final state).
	An arrow with label 1 goes from state q_0 to state q_1 indicates there is a transition (change of state) from state q_0 on input symbol 1 to state q_1 . This is represented as below: $\delta(q_0, 1) = q_1$
	An arrow with label 0 starts and ends in q_0 indicates the machine in state q_0 on reading a 0, remains in same state q_0 . This is represented as below: $\delta(q_0, 1) = q_0$
	The hang (or trap) state is represented by the symbol ' ϕ ' within a circle.
	An arrow with label 0, 1 goes from state q_0 to state q_1 indicates that the machine in state q_0 on reading a 0 or a 1 enters into state q_1 . This is represented as below: $\delta(q_0, 0) = q_1$ $\delta(q_0, 1) = q_1$

1.3 DETERMINISTIC FINITE AUTOMATON [April 16, 18 Oct. 16, 17]

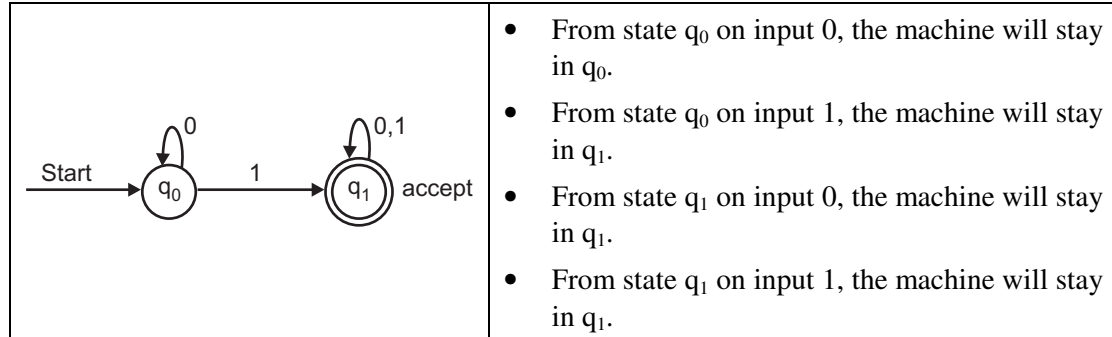
- A Deterministic Finite Automata (DFA) is a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet Σ . For each input symbol, there is exactly one transition from each state.
- Definition of DFA:** Deterministic Finite Automation (DFA) is a 5-tuple or quintuple namely, $(Q, \Sigma, \delta, q_0, F)$, where
 - Q is a finite set of states.
 - Σ is a finite set of symbols called the alphabet.
 - δ is a transition function mapping $Q \times \Sigma$ to Q .
 - q_0 is the initial state from where any input is processed ($q_0 \in Q$).
 - F is a set of final state/states of Q ($F \subseteq Q$).
- The name DFA emerges from the following facts: **[Oct. 18, April 19]**
 - D (Deterministic):** There is exactly one transition for every input symbol from the state. So, it is possible to determine exactly to which state the machine enters into after consuming the input symbol. So, the machine is deterministic.
 - F (Finite):** Has finite number of states and arcs. So, it is deterministic and finite.
 - A (Automation):** Automation is a machine which may accept the string or reject the string. So, it is deterministic finite automation.

- Following are the some common terms used in DFA:

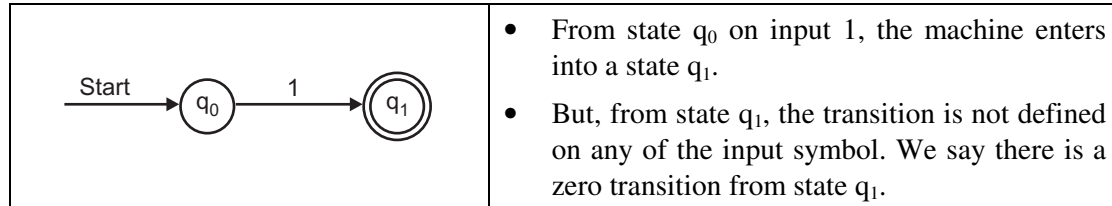
1. Transitions:

- Transition is nothing but change of state after consuming an input symbol.

Example: Consider following DFA and observe the various transactions:



From the above diagram observe that, there is exactly one transition defined from a state on an input symbol. Sometimes, there can be no transitions from a state as shown below:



Note that in a DFA there can be zero or one transition from a state on an input symbol and at any point of time, the DFA will be one state.

2. **Transition Equations:** A finite automation can be represented with the help of transition equations. For each transition there exists a transition equation. For example, consider a finite automation M defined as follows:

$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, \{q_3\})$ is a DFA, where δ is given by

$$\delta(q_1, 0) = q_2 \quad \delta(q_1, 1) = q_1$$

$$\delta(q_2, 0) = q_3 \quad \delta(q_2, 1) = \phi$$

$$\delta(q_3, 0) = q_2 \quad \delta(q_3, 1) = q_3$$

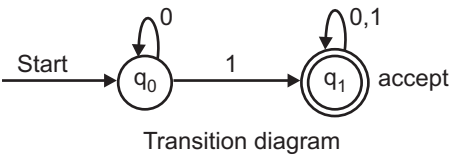
Above finite automation is DFA as it follows the transition function $Q \times \Sigma \rightarrow Q$. This represents that there are total three states in this DFA $\{q_1, q_2, q_3\}$ with q_1 as initial or start state and q_3 as final state or accepting state, the option of input are $\{0, 1\}$.

- The transition equation $\delta(q_1, 0) = q_2$, shows that there is a transition from state q_1 to q_2 on input 0.
- The transition equation $\delta(q_1, 1) = q_1$, shows that there is a transition from state q_1 to q_1 on input 1.
- The transition equation $\delta(q_2, 0) = q_3$, shows that there is a transition from state q_2 to q_3 on input 0.
- The transition equation $\delta(q_2, 1) = \phi$, shows that there is a transition from state q_2 on input 1.

- o The transition equation $\delta(q_3, 0) = q_2$, shows that there is a transition from state q_3 to q_2 on input 0.
- o The transition equation $\delta(q_3, 1) = q_3$, shows that there is a transition from state q_3 to q_3 on input 1.

3. Transition Table: A DFA can also be represented by a transition table. A transition table is the tabular representation of the transition system of the automation. A transition table is also known as transition function table or state table.

Example: The transition diagram and its equivalent transition table are shown below:

 <p>Transition diagram</p>		<table><tr><td colspan="2"></td><th colspan="2">Columns</th></tr><tr><td colspan="2"></td><th>δ</th><th></th></tr><tr><th rowspan="2">Rows</th><th>q_0</th><td>q_0</td><td>q_1</td></tr><tr><th>$*q_1$</th><td>q_1</td><td>q_1</td></tr></table> <p>Transition table</p>			Columns				δ		Rows	q_0	q_0	q_1	$*q_1$	q_1	q_1
		Columns															
		δ															
Rows	q_0	q_0	q_1														
	$*q_1$	q_1	q_1														
<ul style="list-style-type: none">The transition diagram of DFA has two states q_0 and q_1.	\Rightarrow	<ul style="list-style-type: none">Represented using two rows q_0 and q_1.															
<ul style="list-style-type: none">There are two input symbols 0 and 1.	\Rightarrow	<ul style="list-style-type: none">The two input symbols 0 and 1 correspond to two columns.															
<ul style="list-style-type: none">Start state is represented using an arrow mark and not originating from any state and labeled start.	\Rightarrow	<ul style="list-style-type: none">The start state is identified by putting an arrow with direction towards right.															
<ul style="list-style-type: none">The final states are represented by two circles.	\Rightarrow	<ul style="list-style-type: none">The final states are represented by putting star (*) by the side of states.															

4. Language of DFA:

- The language of DFA consist a set of all strings chosen from Σ^* that all are accepted by machine M.

5. State Transition Graph:

- A transition diagram or state transition graph or transition system is the graphical representation of a DFA.
- A transition system or a transition graph is a finite labeled graph in which each vertex (or node) represents a state and directed edge indicate the transition of a state and the edges are labeled with Input/Output.
- Fig. 1.4 shows an example of transition diagram in DFA. In Fig. 1.4,
 - From initial state q_0 and input 0, go to the next state q_1 .
 - From state q_0 and input 1, go to the next state q_0 .
 - From state q_1 and input 0, go to the next state q_1 .
 - From state q_1 and input 1, go to the next state q_2 .
 - From state q_2 and input 0, go to the next state q_2 .
 - From state q_2 and input 1, go to the next state q_2 .

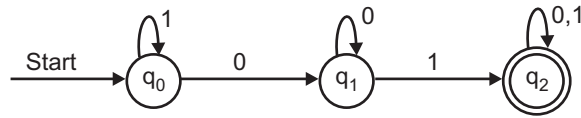


Fig. 1.4

6. Transition Function:

- A DFA may also be represented by transition function δ . The transition function of the Fig. 1.4 is represented as follows:

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

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

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

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

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

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

- Here, the first argument of transition function δ represents the present state and second argument represents the input letter and the right hand side represents the next states.

For example, in equation $\delta(q_0, 0) = q_1$ where, q_0 is a present state, 0 is an input alphabet and q_1 is the next state

- The DFA accepts a string x if the sequence of transitions corresponding to the symbols of x leads from start state to a final state. **[Oct. 18]**
- Finite control representation of DFA is shown in Fig. 1.5.

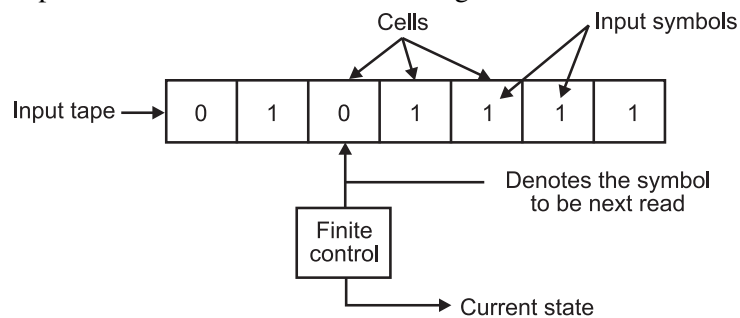


Fig. 1.5: Representation of DFA

- In DFA input tape is divided into squares (cells) containing input symbols from input alphabet.
- A pointer or head points to the cell on the tape from which it is about to read the next symbol, also the head is labeled by the state label in which currently it is residing.
- Input to the finite control is symbol '0' and state of the machine is q_0 . It can take following actions:
 1. Move reading head to the next input symbol.
 2. Change the state of machine.

- **Example**, consider transition diagram in Fig 1.6.

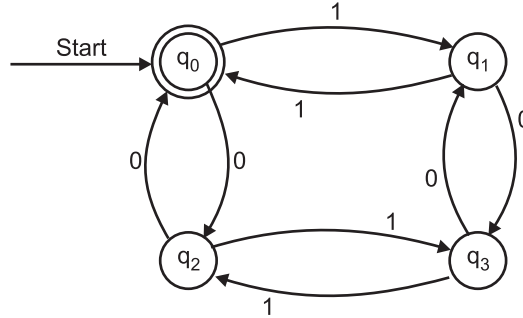


Fig. 1.6: Transition Diagram of a DFA

- The FA accepts all strings containing even number of 1's and even number of 0's (say even-even). We can formally define the FA as, $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$, where the transition function δ is as shown in the Table 1.1.
- In Fig. 1.6, initial state q_0 is shown by arrow labeled start and final state q_0 is shown by two concentric circles.
- The δ is a transition function on a symbol i.e. $\delta(q, a) = p$ means if input symbol is 'a', FA changes state from q to p moving reading head to next input symbol to the right. If the head has moved off the right end of the tape, then it accepts the entire tape.
- We must extend the transition function δ to apply to a state and a string rather than a state and a symbol.
- We define a function $\hat{\delta}$ from $Q \times \Sigma^*$ to Q i.e. $\hat{\delta}(q, w) = p$ means finite automata changes its state from q to p after reading string w .
- We define $\hat{\delta}$ as follows:
 - $\hat{\delta}(q, \epsilon) = q$
 - for all strings w and input symbol 'a', $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.
- Thus, (i) states that without reading an input symbol, the FA cannot change state, and (ii) tells us how to find the state after reading a non-empty input string wa .
- As $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$, δ and $\hat{\delta}$ give same resulting states. Consider example 1. FA always starts in initial state q_0 .
- Suppose 110101 is input to M . We note that $\delta(q_0, 1) = q_1$ and $\delta(q_1, 1) = q_0$. Thus,

Table 1.1

$Q \backslash \Sigma$	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$$\begin{aligned}
 \delta(q_0, 11) &= \delta(\delta(q_0, 1), 1) = \delta(q_1, 1) = q_0 \\
 \delta(q_0, 110) &= \delta(\delta(q_0, 11), 0) = \delta(q_0, 0) = q_2 \\
 \delta(q_0, 1101) &= \delta(\delta(q_0, 110), 1) = \delta(q_2, 1) = q_3 \\
 \delta(q_0, 11010) &= \delta(\delta(q_0, 1101), 0) = \delta(q_3, 0) = q_1 \\
 \delta(q_0, 110101) &= \delta(\delta(q_0, 11010), 1) = \delta(q_1, 1) = q_0
 \end{aligned}$$

- The entire sequence of states is

$$q_0^1 \ q_1^1 \ q_0^0 \ q_2^1 \ q_3^0 \ q_1^1 \ q_0$$
- Thus 110101 is accepted by FA ($\because q_0 \in F$)
- A **string** x is said to be accepted by FA, $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) = p$ for some p in F .
- A **language** accepted by M , $L(M)$, is the set $\{x \mid \delta(q_0, x) \text{ is in } F\}$.
 $\therefore 110101 \in L(M)$.
- A language is a **regular set** (or just regular) if it is the set accepted by some FA.
- In Fig. 1.6,
 - If we change final state we get different languages. If final state is q_3 we get language of odd number of 1's and odd number of 0's (odd-odd).
 - If final state is q_0 , we get language of even number of 0's and even number of 1's (even-even).
 - If final state is q_2 , we get language of odd number of 0's and even 1's (odd-even).
 - If final state is q_1 , we get language of odd number of 1's and even 0's (even-odd).

DFA as Language Recognizer:

- A DFA can be viewed as a simple language recognition device. It called language recognizer because it merely recognizes whether the input strings are in the language or not.
- Fig. 1.7 shows a schematic of a DFA as a recognizer. At any point in time, the DFA would have recognized some sequence of source symbols at the start of the source string, which we call the prefix of the source string, and it would be poised to recognize the source symbol pointed to by the pointer next symbol.
- At the start of DFA operation the prefix would be an empty string. The DFA halts when all symbols in the source string are recognized, or an error condition is encountered.
- The validity of a string is determined by giving it at the input of a DFA that is in its initial state.
- The string is valid if and only if the DFA recognizes every symbol in the string and finds itself in a final state at the end of the string.

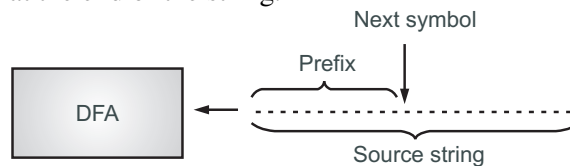


Fig. 1.7: DFA as a Recognizer

- Let us consider a DFA given in Fig. 1.8.

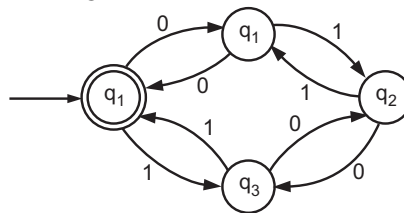


Fig. 1.8

- The Fig. 1.9 processes the strings 1, 00, 01, 010, 011, 1010 and 0100. The processing of two strings 00 and 0110 is terminated to q_0 , which is the final state and rest strings are terminated to a non-final state. Therefore, the strings 00 and 0101 are accepted and rest are not accepted.

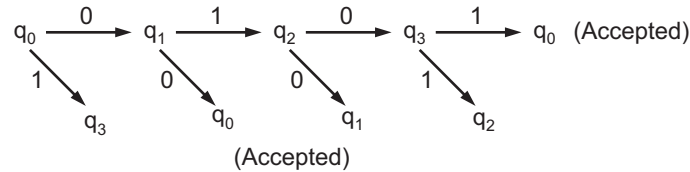


Fig. 1.9: Recognition of Strings 00 and 0101

DFA as Pattern Recognizer:

- DFA is the simplest machine to recognize patterns (strings). For the type of problems that involve pattern recognition, the DFA can be constructed very easily.
- Now, let us see "what are the general steps to be followed while designing the DFA for pattern recognition problems?".
- The steps are:

Step 1 : Identify the minimum string.

Step 2 : Identify the alphabets.

Step 3 : Construct the skeleton of DFA.

Step 4 : Identify other transitions not defined in Step 3.

Step 5 : Construct the DFA using transitions in Step 3 and Step 4.

Example: Drawing a DFA to accept string of a's having at least one 'a'.

Solution: The DFA can be constructed as shown below:

Step 1 : Identifying the minimum string, 'a'.

Step 2 : Identify the alphabets, $\Sigma = \{a\}$.

Step 3 : Construct the skeleton DFA. The DFA for the string 'a' identified in Step 1 can be constructed as shown in Fig. 1.10.

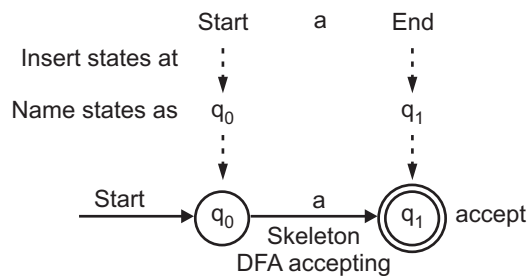


Fig. 1.10

Step 4 : Identify the transitions not defined in Step 3:

Step (i) : $\delta(q_1, a) = ?$ Move q_0 to q_1 on 'a', then think of transition from q_1 on 'a'.

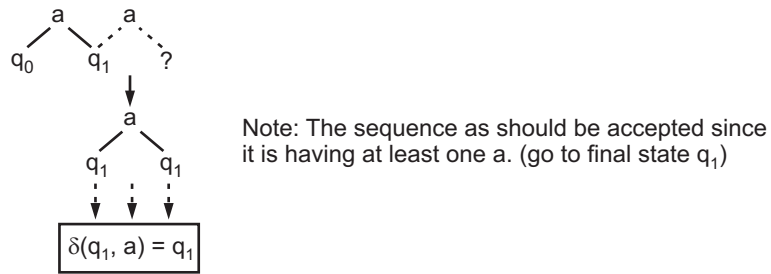


Fig. 1.11

Step 5 : Construct the DFA. The DFA can be obtained by skeleton DFA and transitions obtained from previous step. The DFA is defined as:

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

where,

- o $Q = (q_0, q_1)$
- o $\Sigma = \{a\}$
- o q_0 is the start state
- o $F = \{q_1\}$
- o δ is shown in Fig. 1.12 using the transition diagram and table

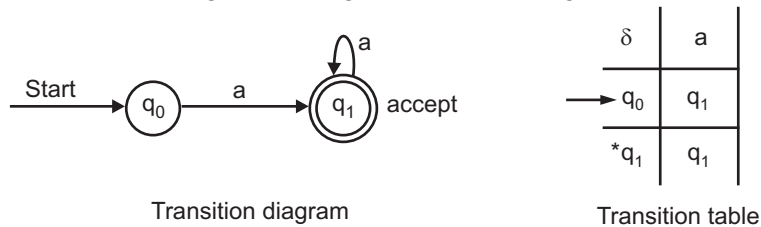


Fig. 1.12

EXAMPLES

[Oct. 17]

Example 1: Construct DFA for language containing strings starting with 'a' and ending with 'b' over alphabet $\{a, b\}$.

Solution:

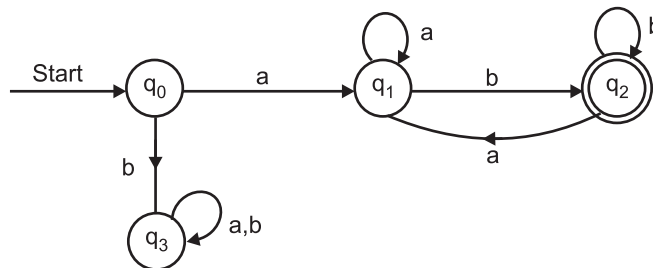


Fig. 1.13: Transition Diagram

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

Where, δ is as given in the Table 1.2.

Table 1.2: Transition table

δ	a	b
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_1	q_2
q_3	q_3	q_3

← rejected state

Example 2: Construct DFA for language which contains all strings with exactly 2 consecutive zero's anywhere.

Solution:

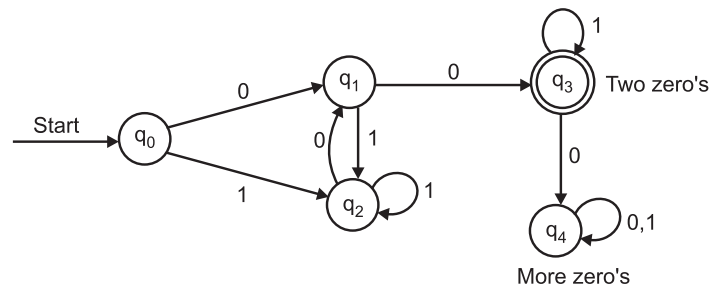


Fig. 1.14: Transition Diagram

Table 1.3: Transition table

δ	0	1
q_0	q_1	q_2
q_1	q_3	q_2
q_2	q_1	q_2
q_3	q_4	q_3
q_4	q_4	q_4

Let,
where,

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2, q_3, q_4\} \\
 \Sigma &= \{0, 1\} \\
 q_0 &= q_0 \\
 F &= \{q_3\}
 \end{aligned}$$

and δ function is as shown in Table 1.3.

Example 3: Construct DFA for a language which contains all strings ending with "ab" or "bc" over $\{a, b, c\}$.

Solution:

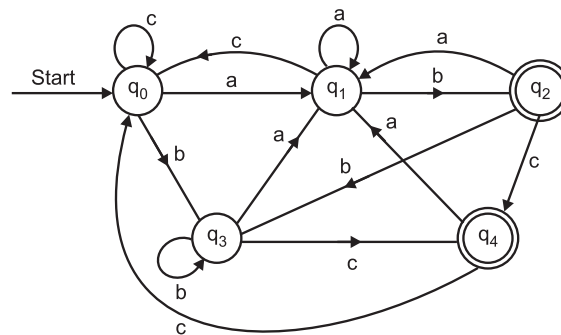


Fig. 1.15: Transition Diagram

Table 1.4: Transition table

δ	a	b	c
q_0	q_1	q_3	q_0
q_1	q_1	q_2	q_0
q_2	q_1	q_3	q_4
q_3	q_1	q_3	q_4
q_4	q_1	q_3	q_0

Let,

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

where,

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

$$\Sigma = \{a, b, c\}$$

$$q_0 = q_0$$

$$F = \{q_2, q_4\}$$

and δ function is as defined in Table 1.4.

Example 4: Construct DFA for accepting strings starting with 'a' and not having "abc" as a substring over $\{a, b, c\}$.

Solution:

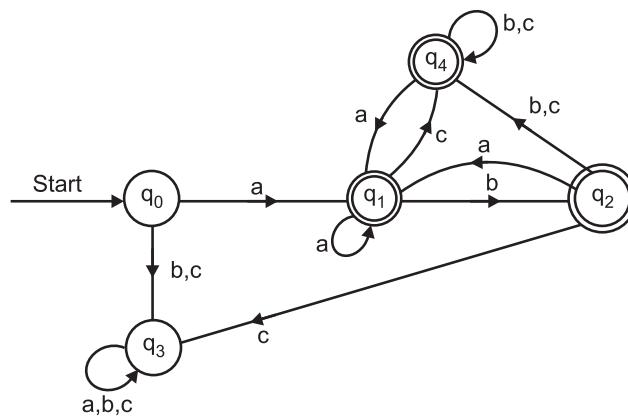


Fig. 1.16: Transition Diagram

Table 1.5: Transition table

δ	a	b	c
q_0	q_1	q_3	q_3
q_1	q_1	q_2	q_4
q_2	q_1	q_4	q_3
q_3	q_3	q_3	q_3
q_4	q_1	q_4	q_4

Let,

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

where,

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

$$\Sigma = \{a, b, c\}$$

δ function is as defined in Table 2.6.

$$q_0 = q_0$$

$$F = \{q_1, q_2, q_4\}$$

Example 5: Construct a DFA for a language containing binary strings from LSB and string is acceptable if it is divisible by 2.

Solution: When the binary string is given from a LSB and we are required to check the divisibility by 2, it will be a easier task. For a binary number to be divisible by 2, the last bit must be zero i.e. LSB = 0. We are given number from LSB, the first bit will indicate divisibility. In other words, it is a string starting with zero.

e.g. 100 is written as 001.

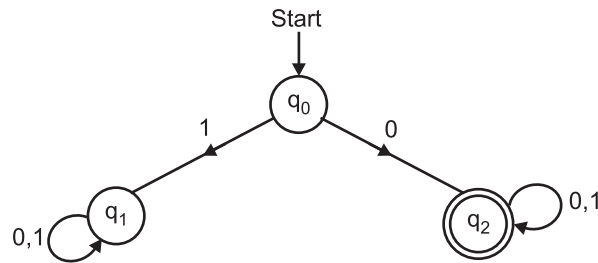


Fig. 1.17: Transition diagram

Table 1.6: Transition table

δ	0	1
q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_2	q_2

$$\therefore M \text{ is } Q = \{q_0, q_1, q_2\}$$

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

$$q_0 = q_0$$

$$F = \{q_2\}$$

The δ function is as shown in the Table 1.6.

Example 6: Construct DFA for language containing string which contains 'a' at every even position in the string.

Solution:

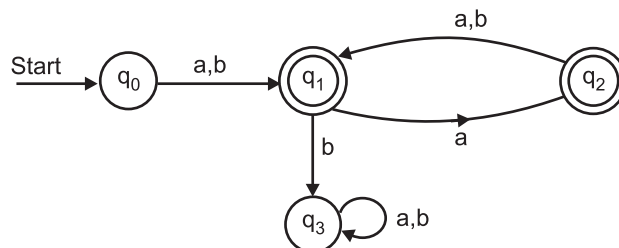


Fig. 1.18: Transition diagram

As the even positions are occupied by 'a', odd positions can be filled by 'a' or 'b'.

Table 1.7: Transition table

δ	a	b
q_0	q_1	q_1
q_1	q_2	q_3
q_2	q_1	q_1
q_3	q_3	q_3

\therefore Finite automata M is,

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

$$\Sigma = \{a, b\}$$

$$q_0 = q_0$$

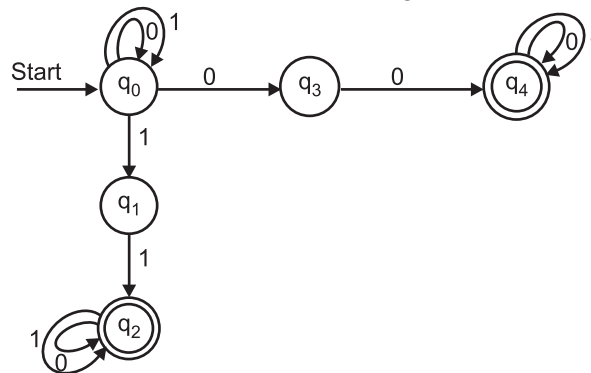
$$F = \{q_1, q_2\}$$

δ function is as shown in Table 1.7.

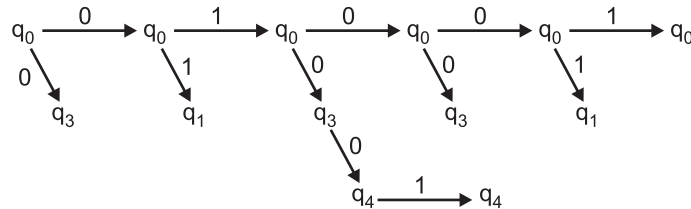
1.4 NON-DETERMINISTIC FINITE AUTOMATON

[April 16]

- In Non-Deterministic Finite Automata (NDFA/NFA) may be more than one move or no move from a given state to the next state of any input symbol.
- NDFA is a simple machine that is used to recognize the pattern by consuming the string of symbols and alphabets for each input symbol.
- Consider a transition diagram for a NFA for a language containing strings with either two consecutive 0's or two consecutive 1's as shown in Fig. 1.19.

**Fig. 1.19: Transition diagram for NFA**

- Consider string 01001, the proliferation of states of an NFA can be shown in Fig. 1.20.

**Fig. 1.20: Proliferation of states of an NFA**

- The string is accepted by NFA, because it leads from initial state to final state (q_4) i.e.

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} q_4$$

Definition of NFA:

- NFA is denoted by 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
 - Q is a finite set of states.
 - Σ is a finite set of symbols called the alphabets.
 - δ is the transition function where $\delta: Q \times \Sigma \rightarrow 2^Q$, (here, the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
 - q_0 is the initial state from where any input is processed ($q_0 \in Q$).
 - F is a set of final state/states of Q ($F \subseteq Q$).
- The function δ can be extended to function $\hat{\delta}$ mapping $Q \times \Sigma^*$ to 2^Q and is defined as follows:
 - $\hat{\delta}(q, \epsilon) = \{q\}$
 - $\hat{\delta}(q, wa) = \{p \mid \text{for some state } r \text{ in } \hat{\delta}(q, w), p \text{ is in } \delta(r, a)\}$. Means starting in state q and reading input string w followed by input symbol, it can be in state p if one possible state we can be in after reading w is r and from r we may go to p upon reading a . Note that $\hat{\delta}(q, a) = \delta(q, a)$. Thus we may replace $\hat{\delta}$ to δ . It is also useful to extend δ to arguments in $2^Q \times \Sigma^*$ by, $\delta(P, w) = \bigcup_{q \in P} \delta(q, w)$ for each set of states $P \subseteq Q$.

EXAMPLES

Example 1: Consider NFA of Fig. 1.10 whose transition function δ is as shown in Table 1.8.

Table 1.8

δ	Inputs	
	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	ϕ
q_4	$\{q_4\}$	$\{q_4\}$

Let the input be 01001.

$$\begin{aligned}
 \delta(q_0, 0) &= \{q_0, q_3\} \\
 \delta(q_0, 01) &= \delta(\delta(q_0, 0), 1) = \delta(\{q_0, q_3\}, 1) = 1 \\
 &= \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\} \\
 \delta(q_0, 010) &= \delta(\delta(q_0, 01), 0) = \delta(\{q_0, q_1\}, 0) \\
 &= \delta(q_0, 0) \cup \delta(q_1, 0) \\
 &= \{q_0, q_3\} \cup \phi \\
 &= \{q_0, q_3\}
 \end{aligned}$$

$$\begin{aligned}
 \delta(q_0, 0100) &= \delta(\delta(q_0, 010), 0) \\
 &= \delta(\{q_0, q_3\}, 0) \\
 &= \delta(q_0, 0) \cup \delta(q_3, 0) \\
 &= \{q_0, q_3, q_4\} \\
 \delta(q_0, 01001) &= \delta(\delta(q_0, 0100), 1) = \delta(\{q_0, q_3, q_4\}, 1) \\
 &= \delta(q_0, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1) \\
 &= \{q_0, q_1, q_4\}
 \end{aligned}$$

Thus the string is accepted by the language of NFA, since $\{q_4\} \in F$.

Example 2: Construct NFA for language which accepts a string starting with a, ending with b or starting with b, ending with a.

Solution:

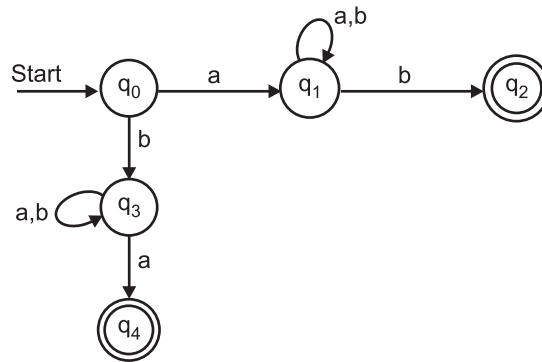


Fig. 1.21: Transition diagram

Table 1.9: Transition table

δ	a	b
q_0	q_1	q_3
q_1	q_1	$\{q_1, q_2\}$
q_2	ϕ	ϕ
q_3	$\{q_3, q_4\}$	q_3
q_4	ϕ	ϕ

Here,

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

where,

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

$$\Sigma = \{a, b\}$$

δ function is as shown in Table 2.10.

$$q_0 = q_0$$

$$F = \{q_2, q_4\}$$

Differences between DFA and NFA/NDFA:**[Oct. 16, April 19]**

Sr. No.	DFA	NFA/NDFA
1.	DFA stands for Deterministic Finite Automaton.	NFA stands for Non-deterministic Finite Automaton.
2.	The transition from a state is to a single particular next state for each input symbol.	The transition from a state can be to multiple next states for each input symbol.
3.	DFA cannot use Empty String transition.	NDFA permits empty string transitions.
4.	DFA is more difficult to construct.	NFA is easier to construct.
5.	All DFA are NFA.	Not all NFA are DFA.
6.	Requires more space.	Requires less space.

1.5 NFA TO DFA**[April 17,19, Oct. 17, 18]**

- Since, every DFA is an NFA, language accepted by DFA is also accepted by NFA. Therefore, if given an NFA, it is possible to convert it to an equivalent DFA.

Theorem: Equivalence of NFA and DFA

- Let L be a set accepted by a NFA. Then there exists a DFA that accepts L.

Conversion Method:

- The major difference between NFA and DFA is because of δ , the transition function of NFA. Since NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where δ is $Q \times \Sigma \rightarrow 2^Q$. Because of this, the entries in the state table are sets instead of singular entry like DFA.
- While converting NFA into equivalent DFA, if we consider $Q' = 2^Q$, as set of states for resulting DFA, the transition function for DFA will become $Q' \times \Sigma \rightarrow Q'$.

Example 1: Construct DFA equivalent to NFA, $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$, where δ function is shown in Table 1.10 for NFA.

Table 1.10: NFA transition table

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

Solution: For given NFA,

$$Q = \{q_0, q_1\}, \Sigma = \{0, 1\}, F = \{q_1\}, q_0 = q_0 \text{ (initial state)}$$

Let the resultant DFA is

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

where $\Sigma = \{0, 1\}$, starting state = q_0 .

The power set of Q is

$$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$$

According to our method, $Q' = 2^Q = \text{set of states of DFA}$

We use '[']' instead of '{ }'.

$$\therefore Q' = \{[q_0], [q_1], [q_0, q_1]\}$$

We have excluded ϕ as it does not denotes any states of given NFA.

Now, we have to find δ' for every state in Q' on both '0' as well as '1'.

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

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

$$\delta'([q_1], 0) = \phi \quad \text{from Table 1.11}$$

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

We have to find δ' for one more state $[q_0, q_1]$.

$$\delta'([q_0, q_1], 0) = \delta(q_0, 0) \approx \delta(q_1, 0) = \{q_0, q_1\} \approx \phi = [q_0, q_1]$$

$$\delta'([q_0, q_1], 1) = \delta(q_0, 1) \approx \delta(q_1, 1) = \{q_1\} \approx \{q_0, q_1\} = [q_0, q_1]$$

\therefore the resultant DFA is as shown in Fig. 1.11.

Table 1.11 Transition table

δ	0	1
q_0	$[q_0, q_1]$	$[q_1]$
q_1	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

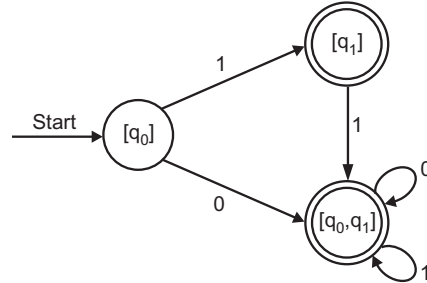


Fig. 1.22: Equivalent DFA

The machine can have one initial state but more than one final state.

Note: Final states are those where at least one final state is included.

Example 2: Find a deterministic acceptor equivalent to

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

Where, δ is as given by Table 1.12.

Table 1.12: State table

State/ Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_2
q_1	q_0	q_1
q_2 (final)	ϕ	q_0, q_1

Solution: The deterministic automaton M_1 equivalent to M is defined as follows:

$$M = (2^Q, \{a, b\}, \delta, [q_0], F')$$

where

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

We start the construction by considering $[q_0]$ first. We get $[q_2]$ and $[q_0, q_1]$. Then we construct δ for $[q_2]$ and $[q_0, q_1]$. $[q_1, q_2]$ is a new state appearing under the input columns. After constructing δ for $[q_1, q_2]$, we do not get any new states and so we terminate the construction of δ . The state table is given by Table 1.13.

Table 1.13: State table of M_1

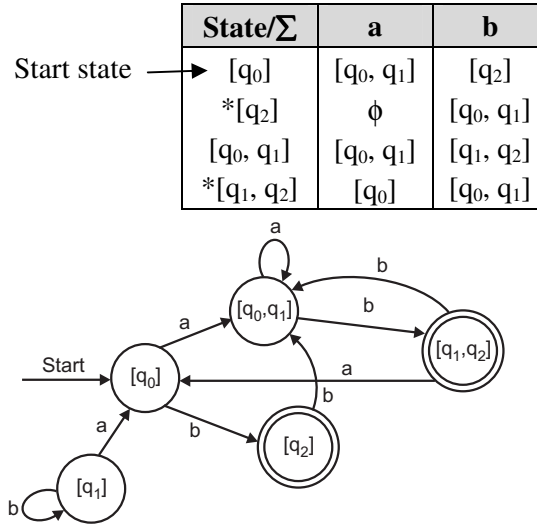


Fig. 1.23

Example 3: Construct a deterministic finite automaton equivalent to

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$$

where δ is given by Table 1.14.

Table 1.14: State table for Example 2.8

State/ Σ	a	b
$\rightarrow q_0$	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
$\odot q_3$	ϕ	q_2

Solution: Let $Q = \{q_0, q_1, q_2, q_3\}$. Then the deterministic automaton M_1 equivalent to M is given by

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F)$$

where F consists of:

$$[q_3], [q_0, q_3], [q_1, q_3], [q_2, q_3], [q_0, q_1, q_3], [q_0, q_2, q_3], [q_1, q_2, q_3]$$

and $[q_0, q_1, q_2, q_3]$

and where δ is defined by the state table given by Table 1.15.

Table 1.15: State table of M_1

State/ Σ	A	b
Start state \rightarrow $[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_3]$
* $[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$
* $[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$	$[q_0, q_1, q_2, q_3]$

1.6 NFA WITH ϵ TRANSITIONS

[April 16, 17, 18, 19, Oct. 18]

- It is an NFA including transitions on the empty input i.e. ' ϵ ' (epsilon). We can extend a NFA by introducing ' ϵ -moves' that allow us to make a transition on the empty string.
- There would be an edge labeled ϵ between two states and this edge allows transition from one state to another even without receiving an input symbol. This is another mechanism that allows NFA to be in multiple states at once.
- Constructing such NFA is easy, but the NFA thus constructed is not that powerful. The NFA with ϵ -moves is given by $M = (Q, \Sigma, \delta, q_0, F)$ where δ is defined as $Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$.

Definition:

- NFA with ϵ -moves is denoted by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$,
 where
 - Q : finite set of states
 - Σ : finite input alphabet
 - q_0 : initial state contained in Q
 - F : set of final states $\subseteq Q$
 and
 - δ : transition function or state function mapping
 $Q \times (\Sigma \cup \{\epsilon\})$ to 2^Q
 i.e.
 - δ : $Q \times (\Sigma \cup \{\epsilon\}) \cup 2^Q$

Basically, it is NFA with few transitions on the empty input ' ϵ '.

Note : DFA can never have ϵ -moves.

Example:

- Let us consider the FA shown in Fig. 1.24. This FA accepts a language consisting any number of 0's followed by any number of 1's followed by any number of 2's.
- For example, the strings $\epsilon, 0, 00, 01, 1, 11, 2, 22, 12, 122, 012, 0012$, etc. are all valid strings of this language.

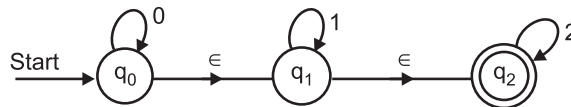
Fig. 1.24: NFA with ϵ -Moves

Table 1.16: Transition table

δ	0	1	2	ϵ
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

- Consider a string "002" which is accepted by NFA of Fig. 1.24 by the path q_0, q_0, q_1, q_2 , q_2 with arcs labeled 0, 0, ϵ , ϵ , 2.
- In real life, ' ϵ ' means 'nothing'. So we cannot say on reading 'nothing' machine has changed its state. Therefore, we want the deterministic machine which tells what will be the output. Hence, we can convert NFA with ϵ -moves to NFA without ϵ -moves.

Definition of ϵ -closure of a State:**[April 18, 19]**

- The ϵ -closure (q), pronounced as epsilon closure of ' q ' is defined as the set of all states ' p ' such that there is a path from ' q ' to ' p ' labelled ' ϵ ' i.e. it is the set of states with distance zero from state ' q '.
- Consider above NFA of Fig. 1.24.

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

- State q_0 is also added to the set because every state is at distance zero from itself and it is the path from q_0 to q_0 with all arcs labelled ϵ , path q_0 to q_1 on ϵ shows that q_1 is in ϵ -closure (q_0) and path $q_0 - q_1 - q_2$ on ϵ shows that q_2 is in ϵ -closure(q_0).
- Similarly, $\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$
 $\epsilon\text{-closure}(q_2) = \{q_2\}$
- Let NFA without ϵ -moves is $M = (Q, \Sigma, \delta'', q_0, F')$, where $\delta'' : Q \times \Sigma \rightarrow 2^Q$. We define new δ' as follows:

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

$$\delta''(q, a) = \epsilon\text{-closure}(\delta(\delta'(q, \epsilon), a))$$

$$\text{where, } \delta'(q, \epsilon) = \epsilon\text{-closure}(q)$$

- Same can be written as $\hat{\delta}(q, a) = \epsilon\text{-closure}(\delta(\hat{\delta}(q, \epsilon), a))$.
- Convert NFA with ϵ -moves in Fig. 1.24 to equivalent NFA without ϵ -moves accepting the same language.
- From the definition of ϵ -closure,

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

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

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

- We see that, q_2 is a final state in Fig. 1.24, since q_2 is in ϵ -closure (q_0) and ϵ -closure (q_1); both q_0, q_1 are in a set of final state.

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

- Now, find $\hat{\delta}$ for resultant NFA.

$$\begin{aligned}
 \hat{\delta}(q_0, 0) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), 0)) \\
 &= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0)) \\
 &= \epsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\
 &= \epsilon\text{-closure}(\{\emptyset\} \cup \emptyset \cup \emptyset) \\
 &= \epsilon\text{-closure}(\{q_0\}) = \{q_0, q_1, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \hat{\delta}(q_0, 1) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), 1)) \\
 &= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 1)) \\
 &= \epsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)) \\
 &= \epsilon\text{-closure}(\emptyset \cup \{q_1\} \cup \emptyset) \\
 &= \epsilon\text{-closure}(q_1) \\
 &= \{q_1, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \hat{\delta}(q_0, 2) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), 2)) \\
 &= \epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 2)) \\
 &= \epsilon\text{-closure}(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)) \\
 &= \epsilon\text{-closure}(\emptyset \cup \emptyset \cup \{q_2\}) \\
 &= \epsilon\text{-closure}(q_2) \\
 &= \{q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \hat{\delta}(q_1, 0) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), 0)) \\
 &= \epsilon\text{-closure}(\delta(\{q_1, q_2\}, 0)) \\
 &= \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0)) \\
 &= \epsilon\text{-closure}(\emptyset \cup \emptyset) \\
 &= \epsilon\text{-closure}(\emptyset) \\
 &= \emptyset
 \end{aligned}$$

$$\begin{aligned}
 \hat{\delta}(q_1, 1) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), 1)) \\
 &= \epsilon\text{-closure}(\delta(\{q_1, q_2\}, 1)) \\
 &= \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1)) \\
 &= \epsilon\text{-closure}(\{q_1\} \cup \emptyset) \\
 &= \epsilon\text{-closure}(q_1) \\
 &= \{q_1, q_2\}
 \end{aligned}$$

- Similarly, we can obtain,

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

$$\hat{\delta}(q_2, 0) = \emptyset$$

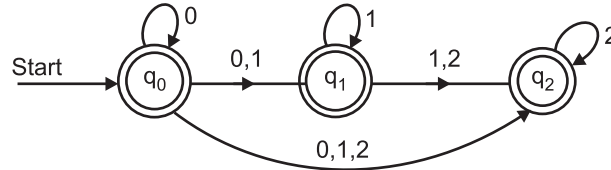
$$\hat{\delta}(q_2, 1) = \emptyset$$

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

- Resultant $\hat{\delta}$ is as shown in the transition Table 1.17.

Table 1.17

δ	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

Fig. 1.25: Resultant NFA without ϵ -Moves

- There is a transition from q_0 to q_1 labeled 0 or 1 meaning there are two transitions from q_0 to q_1 one on 0 and other on 1. Here they are combined for simplicity.

Example 1: Convert NFA with ϵ -moves in Fig. 1.26 to equivalent NFA.

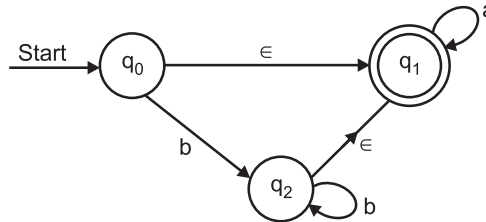


Fig. 1.26: Transition diagram

Solution:

Table 1.18: Transition table

δ	a	b	ϵ
q_0	ϕ	$\{q_2\}$	$\{q_1\}$
q_1	$\{q_1\}$	ϕ	ϕ
q_2	ϕ	$\{q_2\}$	ϕ

Consider NFA with ϵ -transition in Fig. 1.26.

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

$$\epsilon\text{-closure}(q_1) = \{q_1\}$$

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

$$\begin{aligned}
 \hat{\delta}(q_0, a) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), a)) \\
 &= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a)) \\
 &= \epsilon\text{-closure}(\phi \cup q_1) \\
 &= \epsilon\text{-closure}(q_1) \\
 &= \{q_1\}
 \end{aligned}$$

$$\begin{aligned}
\hat{\delta}(q_0, b) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_0, \epsilon), b)) \\
&= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b)) \\
&= \epsilon\text{-closure}(q_2 \cup \phi) \\
&= \epsilon\text{-closure}(q_2) \\
&= \{q_1, q_2\} \\
\hat{\delta}(q_1, a) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), a)) \\
&= \epsilon\text{-closure}(\delta(q_1, a)) \\
&= \epsilon\text{-closure}(q_1) \\
&= \{q_1\} \\
\hat{\delta}(q_1, b) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_1, \epsilon), b)) \\
&= \epsilon\text{-closure}(\delta(q_1, b)) \\
&= \epsilon\text{-closure}(\phi) \\
&= \phi \\
\hat{\delta}(q_2, a) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), a)) \\
&= \epsilon\text{-closure}(\delta(q_1, a) \cup \delta(q_2, a)) \\
&= \epsilon\text{-closure}(q_1 \cup \phi) \\
&= \{q_1\} \\
\hat{\delta}(q_2, b) &= \epsilon\text{-closure}(\delta(\hat{\delta}(q_2, \epsilon), b)) \\
&= \epsilon\text{-closure}(\delta(q_1, b) \cup \delta(q_2, b)) \\
&= \epsilon\text{-closure}(\phi \cup q_2) \\
&= \{q_1, q_2\}
\end{aligned}$$

Therefore equivalent NFA is as shown in the Fig. 1.27.

Table 1.19: Transition table

δ	a	b
q_0	$\{q_1\}$	$\{q_1, q_2\}$
q_1	$\{q_1\}$	ϕ
q_2	$\{q_1\}$	$\{q_1, q_2\}$

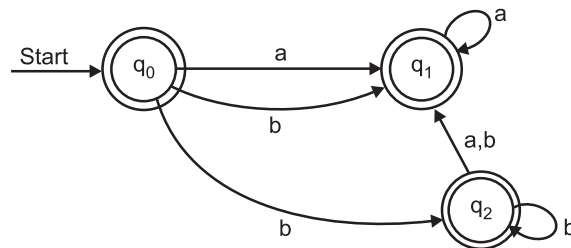


Fig. 1.27: Equivalent NFA

The set of final state contains state q_0 because ϵ -closure of q_0 contains q_1 , which is a final state. State q_2 is also included in the set of final state for the same reason.

1.7 NFA WITH ϵ TRANSITIONS TO DFA

- Since, every DFA is an NFA, language accepted by DFA is also accepted by NFA. Therefore, if given an NFA, it is possible to convert it to an equivalent DFA.

Constructing a DFA from NFA: (NFA to DFA Conversion):

Input: An NFA N.

Output: A DFA D accepting the same language.

Method:

- Each state of D is a set of states which N could be in after reading some sequence of input symbols. The initial state of DFA D is q_0 , which is initial state of NFA N together with all states of N that can be reached from q_0 by means of ϵ -transitions only. The accepting states of D are those sets of states that contain at least one accepting state of N. Here initial state is ϵ -closure (q_0).
- While there is an unmarked state
 $x = \{q_1, q_2, \dots, q_n\}$ of D do
begin
mark x;
for each input symbol a do
begin
let T be the set of states to which there is a transition on 'a' from some state q_i in x;
 $y = \epsilon$ -closure (T);
if y has not yet been added to the set of states of D then make y an "unmarked" state of D;
add a transition from x to y labelled 'a' if not already present.
end
end.
end.

Here, we consider ϵ -closure (q_0) be a state of D. This state is the start state of D. We assume each state of D is initially "unmarked". Then repeat this procedure until no more states can be added in D.

Example 1: Construct equivalent DFA for the NFA of Fig. 1.28 (a).

[April 16]

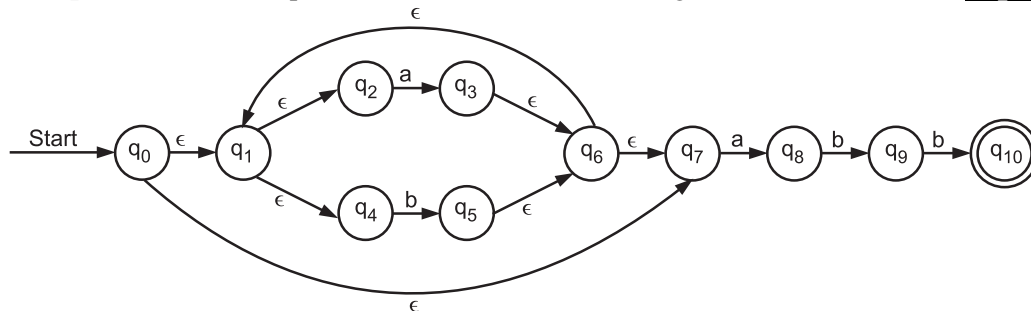


Fig. 1.28: (a) NFA

The language of NFA is containing all strings ends with abb over $\{a, b\}$.

Solution: The initial state of the equivalent DFA is ϵ -closure (q_0) = $\{q_0, q_1, q_2, q_4, q_7\} = A$ (say), since there are exactly the states reachable from state q_0 via a path in which every edge is labeled ϵ and state itself is in ϵ -closure (q_0).

Here we mark set A i.e. set x to A and compute T. The set of states of N having transitions on 'a' from all members of A. So among the states q_0, q_1, q_2, q_4 and q_7 , only states q_2 and q_7 have such transitions to q_3 and q_8 , so

$$\begin{aligned} y = \epsilon\text{-closure}(\{q_3, q_8\}) &= \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \\ &= B \text{ (say)} \end{aligned}$$

Now on input symbol b, from all members of states A we get q_5 is only reachable state.

$$\begin{aligned} \therefore y = \epsilon\text{-closure}(\{q_5\}) &= \{q_1, q_2, q_4, q_5, q_6, q_7\} \\ &= C \text{ (say)} \end{aligned}$$

Now, we continue this process with the unmarked sets B and C.

From B on input 'a' we get,

$$y = \epsilon\text{-closure}(\{q_3, q_8\}) = B$$

From B on input 'b' we get,

$$\begin{aligned} y &= \epsilon\text{-closure}(\{q_5, q_9\}) \\ &= \{q_1, q_2, q_4, q_5, q_6, q_7, q_9\} \\ &= D \text{ (say)} \end{aligned}$$

From C on input 'a' we get,

$$y = \epsilon\text{-closure}(\{q_3, q_8\}) = B$$

From C on input 'b' we get,

$$y = \epsilon\text{-closure}(\{q_5\}) = C$$

From D on input 'a' we get,

$$y = \epsilon\text{-closure}(\{q_3, q_8\}) = B$$

From D on input 'b' we get,

$$\begin{aligned} y &= \epsilon\text{-closure}(\{q_5, q_{10}\}) \\ &= \{q_1, q_2, q_4, q_5, q_6, q_7, q_{10}\} \\ &= E \text{ (say)} \end{aligned}$$

From E on input 'a' we get,

$$y = \epsilon\text{-closure}(\{q_3, q_8\}) = B$$

From E on input 'b' we get,

$$y = \epsilon\text{-closure}(\{q_5\}) = C$$

The transition table of the resulting DFA is shown in Table 1.20. This DFA will also accept the same language as that of NFA and having less number of states.

Table 1.20: Transition table

	δ	a	b
Start \rightarrow	A	B	C
	B	B	D
	C	B	C
	D	B	E
Accept \rightarrow	E	B	C

Here set E contains final state q_{10} of NFA. Therefore E is final state of DFA.

Fig. 1.28 (b) shows the resultant DFA D.

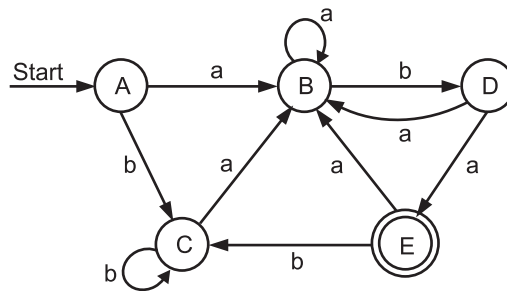


Fig. 1.28: (b) Equivalent DFA

Example 2: Construct DFA equivalent to NFA of Fig. 1.29.

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

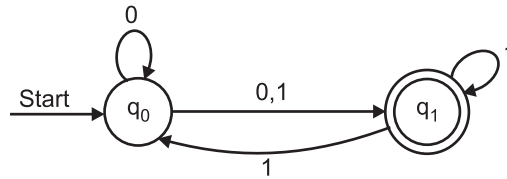


Fig. 1.29: NFA

Solution: Here we are not having any ϵ -transition. So ϵ -closure of any state is that state itself.

Let initial state is ϵ -closure (q_0) = $\{q_0\}$ = A.

From A on input '0' we get set,

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

From A, on input '1', we get set,

$$\{q_1\} = C$$

From B on input '0' we get set = $\{q_0, q_1\}$ = B.

From B on input '1' we get set $\{q_0, q_1\}$ = B.

From C on input '0' we get set = $\{\phi\}$.

From C on input '1' we get set = $\{q_0, q_1\}$ = B.

Therefore resultant transition table is

Table 1.21: Transition table of DFA

δ	0	1
A	B	C
B	B	B
C	ϕ	B

The resultant DFA is shown in Fig. 1.30.

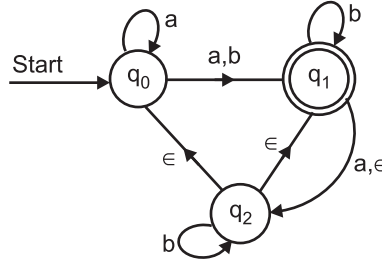


Fig. 1.30: Equivalent DFA

Since the set B and C contains the state q_1 which was final state of NFA, DFA contains B and C has final states.

Here $A = [\{q_0\}]$, $B = [\{q_0, q_1\}]$ and $C = [\{q_1\}]$.

Example 3: Construct equivalent DFA for NFA of Fig. 1.31.

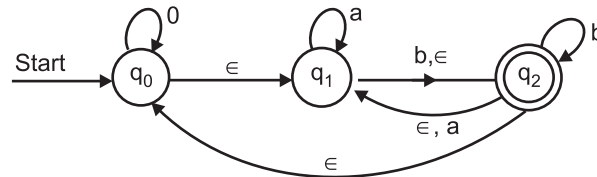


Fig. 1.31: NFA

Solution: The initial state is q_0 .

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} = A \text{ (say)}$$

From A on input 'a' we get the states q_0 and q_1 . Hence, by taking ϵ -closure of these states we get, $\{q_0, q_1, q_2\}$.

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

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

Hence, from all member of A on input 'a' we get

$$\{q_0, q_1, q_2\} = A \text{ (again)}$$

Similarly, from A on input 'b' we get the state q_2 and $\epsilon\text{-closure}(q_2) = \{q_0, q_1, q_2\}$

$$\therefore \text{ We get, } \{q_0, q_1, q_2\} = A \text{ (again)}$$

We get the same set again and again as we proceed further. Therefore transition table we get as follows:

Table 1.22: Transition table

δ	a	b
A	A	A

The equivalent DFA is shown in Fig. 1.32.

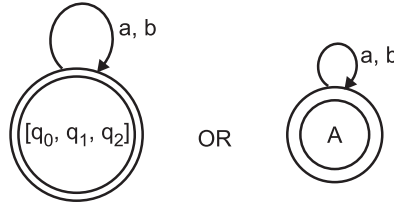


Fig. 1.32: Equivalent DFA

Example 4: Construct equivalent DFA for the NFA of Fig. 1.33.

$$M = (\{q_0, \dots, q_7\}, \{a, b\}, \delta, q_0, \{q_7\})$$

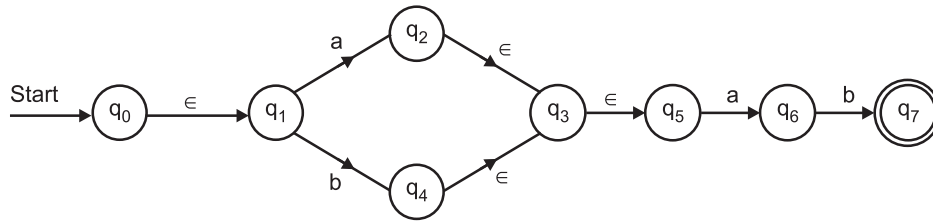


Fig. 1.33: NFA

Solution: Initial state is q_0 , we take:

$$\epsilon\text{-closure}(q_0) = [q_0, q_1] = A \rightarrow \text{initial state of DFA.}$$

1. From A, on input 'a' we get only state q_2 . So

$$\epsilon\text{-closure}(q_2) = \{q_2, q_3, q_5\} = B$$

From A, on input 'b' we get only state q_4 . So

$$\epsilon\text{-closure}(q_4) = \{q_4, q_3, q_5\} = C$$

2. From B, on input 'a' we get only state q_6

$$\epsilon\text{-closure}(q_6) = \{q_6\} = D$$

From B, on input 'b' we get set ϕ .

3. From C, on input 'a' we get only state q_6 .

$$\therefore \epsilon\text{-closure}(q_6) = \{q_6\} = D \text{ (again)}$$

From C, on input 'b' we get set $\{\phi\}$.

4. From D, on input 'a' we get set $\{\phi\}$.

From D, on input 'b' we get only state q_7 .

$$\epsilon\text{-closure}(q_7) = \{q_7\} = E$$

5. From E, on input 'a' we get set $\{\phi\}$.

Also from E, on input 'b' we get set $\{\phi\}$.

Therefore, equivalent DFA for above NFA is shown in Fig. 1.34 along with transition table.

Table 1.23: Transition table

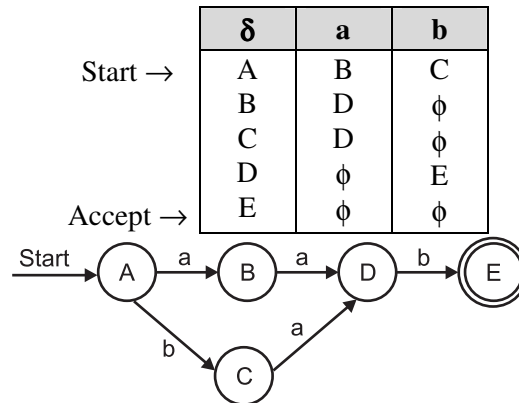


Fig. 1.34: Equivalent DFA

Here, set E contains final state q_7 . Therefore E is a final state.

Example 5: Construct DFA equivalent to NFA of Fig. 1.35.

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

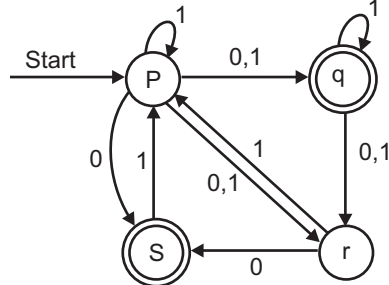


Fig. 1.35: NFA

Solution: Initial state of DFA is $P = A$ (say).

A	$\xrightarrow{0}$	$\{q, r, s\} = B$
A	$\xrightarrow{1}$	$\{p, q, r\} = C$
B	$\xrightarrow{0}$	$\{r, s\} = D$ ($\because q$ on $0 = r$ and r on $0 = s$)
B	$\xrightarrow{1}$	$\{p, q, r\} = C$
C	$\xrightarrow{0}$	$\{q, r, s\} = B$
C	$\xrightarrow{1}$	$\{p, q, r\} = C$
D	$\xrightarrow{0}$	$\{s\} = E$
D	$\xrightarrow{1}$	$\{p\} = A$
E	$\xrightarrow{0}$	$\{\phi\}$
E	$\xrightarrow{1}$	$\{p\} = A$

Therefore equivalent DFA is shown in Fig. 1.36.

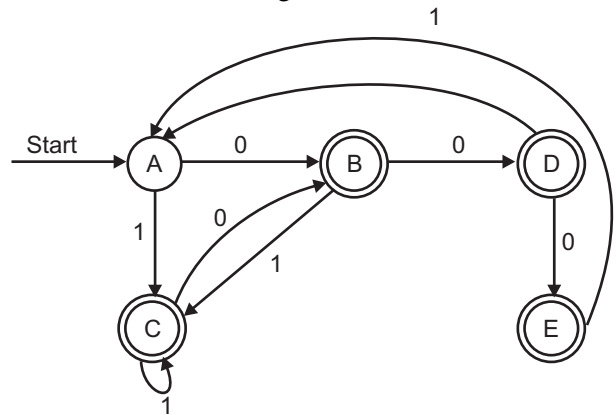


Fig. 1.36: Equivalent DFA

1.8 FINITE AUTOMATA WITH OUTPUT

[Oct. 17, 18]

- As we know, FA is the mathematical model of FSM and is defined by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ which does not include the information about output.
- After reading a string if FA resides in final state, it says that, the string is "accepted" by FA else it says that, the string is "rejected" by the FA.
- But if we need to produce some more precise information and not only 'accept' or 'reject'; then we shall consider the finite set of output symbols and the Machine Function (MAF) as well; that we have considered while learning Finite State Machines (FSM).
- DFA may have outputs corresponding to each transition. There are two different types of machines which can be formulated as FA with output namely, Moore machine and Mealy machine.

Moore Machine:

[April 16, 17, Oct. 17, 18]

- Moore machine is the machine with finite number of states and for which, the output symbol at a given time depends only upon the present state of machine.

Definition: A Moore machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where,

- Q : Finite set of states.
- Σ : Finite input alphabet.
- Δ : An output alphabet.
- δ : State function, $\delta : Q \times \Sigma \rightarrow Q$.
- λ : Machine function, $\lambda : Q \rightarrow \Delta$.
- q_0 : Initial state of the machine.

- Thus, for Moore machine, an output symbol is associated with each state. When the machine is in a particular state, it produces the output, irrespective of what was the input on which the transition is made.

Example 1: Construct a Moore machine to determine the residue (remainder) mod 3 for a binary number.

Solution: (i) If 'i' is a binary number and if we write '0' after 'i' then its value becomes '2i'.

e.g. Consider binary number

$$i = '1' \text{ (value = 1)}$$

If we write '0' after it,

$$'10' = 2 \text{ i.e. value} = 2 \times 1$$

As another example, if we consider

$$i = '100' \text{ (value = 4)}$$

$$\begin{aligned} \text{then } i.0 &= 1000 \text{ has value} = 8 \\ &= 2 \times 4 \end{aligned}$$

(ii) If we write '1' after 'i', where 'i' is any binary number, its value becomes '2i + 1'.

$$\text{e.g. If } i = '1' \text{ (value = 1)}$$

$$i.1 = '11' = 3 \text{ i.e. value} = 2 \times 1 + 1 = 3$$

As another example, if

$$i = '100' \text{ (value 4)}$$

$$i.1 = 1001 = 9$$

$$\text{i.e. value} = 2 \times 4 + 1 = 9$$

As we are constructing a machine to determine remainder, when we divide any binary number by 3, the different remainder values that we can have are 0, 1 and 2.

Table 1.24: Different remainder values

Remainder value (R)	0	1	2
When we write '0' after i + (R) and divide by 3, the remainder = 2R mod 3	0	2	1
When we write '1' after i + (R) and divide by 3, the remainder = (2R + 1) mod 3	1	0	2

In above Table 1.24, if say, the remainder of previous result is '2' i.e. '10' and if we write '0' after it, it becomes '4' i.e. '100'. Now if we divide it by '3' the remainder will be '1'.

To construct a Moore machine, we can associate three different residue values with three different states; i.e. '0' residue value with state 'q₀', residue '1' with 'q₁' and so on as described in machine function stated in Table 1.25.

Table 1.25: Machine function for Moore machine $\lambda: Q \rightarrow \Delta$

State Q	q ₀	q ₁	q ₂
Output Δ	0	1	2

Now, Table 1.26 can be viewed as the state table or transition table for the required Moore machine, given in Table 1.26 and resultant Moore machine is given in the Fig. 1.37.

Table 1.26: Transition table for required Moore machine $\delta: Q \times \Sigma \rightarrow Q$

δ	0	1	Output λ
q ₀	q ₀	q ₁	0
q ₁	q ₂	q ₀	1
q ₂	q ₁	q ₂	2

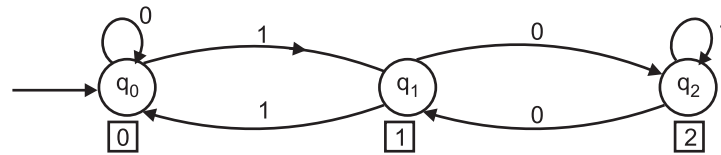


Fig. 1.37: Moore Machine finding residue mod 3 for Binary Number

Example 2: Design a Moore machine for a language over $\{0, 1\}$ which outputs '*' if string contains '11' in it and outputs '#' otherwise.

Solution:

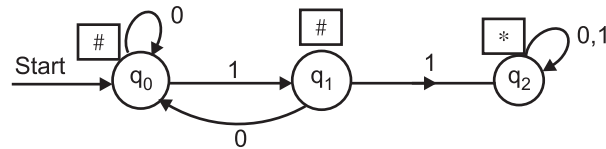


Fig. 1.38

Table 1.27: Transition table

Q	δ		Output λ
	0	1	
q ₀	q ₀	q ₁	#
q ₁	q ₀	q ₂	#
q ₂	q ₂	q ₂	*

Example 3: Design a Moore machine to get 1's complement of a given binary string.

Solution:

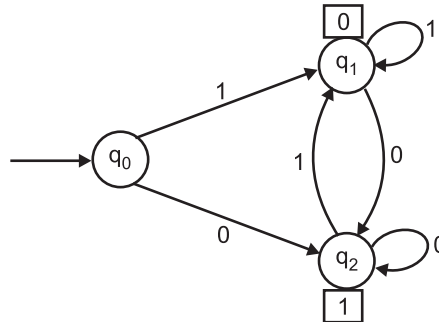


Fig. 1.39

Table 1.28: Transition table

Q	δ		Output λ
	0	1	
q ₀	q ₂	q ₁	0
q ₁	q ₂	q ₁	0
q ₂	q ₂	q ₁	1

Mealy Machine:**[Oct. 16, 17, 18, April 17, 19]**

- It is the machine with finite number of states and for which, the output symbol at a given time is a function of (i.e. it depends on) the present input symbol, as well as the present state of the machine.

Definition: A Mealy machine is denoted by a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$,

- where,
- Q : Finite set of states.
 - Σ : Finite input alphabet
 - Δ : Finite output alphabet
 - δ : State function, $\delta'' : Q \times \Sigma \rightarrow Q$
 - λ : Machine function $\lambda : Q \times \Sigma \rightarrow \Delta$
 - q_0 : Initial state of the machine.

- Thus, for this type of machine the output depends on both current state and the current input symbol.

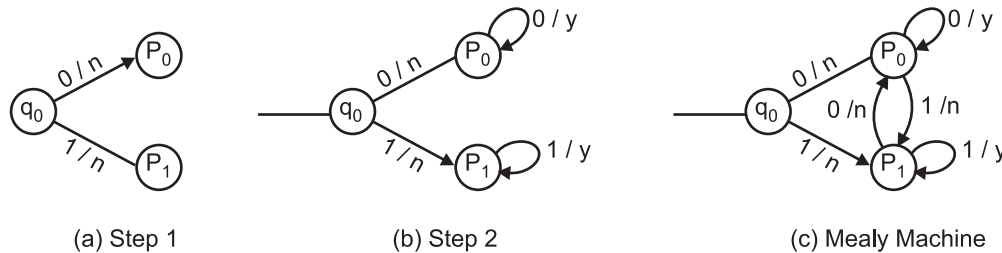
Example 1: Design a Mealy machine accepting the language consisting of strings from Σ^* where $\Sigma = \{0, 1\}$ and ending with double zero's or double one's.

Solution: Let us assume that, the output alphabet

$$\Delta = \{y, n\}$$

'y' i.e. 'yes' will be the output if string is accepted and 'n' i.e. 'no' will be the output if string is not accepted i.e. rejected by the machine.

Let us assume that the initial state is ' q_0 '. From ' q_0 ' there will be two transitions, one on '0' and other on '1' as $\Sigma = \{0, 1\}$. On '0' we should make the transition to some other state, say ' P_0 ', which will look for second consecutive zero to get double zero's. Similarly, on '1', transition will go to ' P_1 ' which will look for second consecutive one to get double one's [refer Fig. 1.40 (a)].

**Fig. 1.40**

In ' P_0 ' we get '0', it should remain in the same state and should produce output 'y' as we got double zero's. It should remain in the same state because the string may be of the form "0000". Similarly, in ' P_1 ' if it gets '1', it should go to itself with output 'y' indicating it has got double '1's. This state is reflected in Fig. 1.40 (b).

Now, in ' P_0 ' if we get '1', machine should make transition to ' P_1 ' which will be checking for second consecutive '1' and in ' P_1 ' if we get '0', it should show the transition from ' P_1 ' to ' P_0 ' which will be looking for second consecutive zero. Here ends the process at considering transitions on both '0' and '1' from every state. Fig. 1.40 (c) shows the final Mealy machine with given requirements.

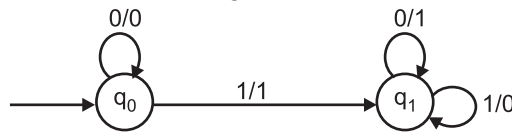
Example 2: Design a Mealy machine to find out 2's complement of a given binary number.

Solution: Steps to be followed are:

- (i) Read bit by bit from LSB.
- (ii) Keep the bits unchanged till you get first '1'; do not replace this '1' by '0'. Keep that also unchanged.
- (iii) The remaining bits are to be changed from '0' to '1' and from '1' to '0'.

The design here requires again two states $Q = \{q_0, q_1\}$ with q_0 as initial state, ' q_0 ' will read '0's without replacing them till it gets first '1'. On getting '1', it makes a transition to ' q_1 ' which now replaces each coming '0' by '1' and '1' by '0'.

The Mealy machine can be drawn as in Fig. 1.41.



**Fig. 1.41: Mealy Machine finding 2's complement of a binary number
(Reading number from LSB)**

e.g. "1010" – gives number reading from right to left.

The output can be obtained by reading bottom to top as we are taking input in reverse order, right to left. Output will be "0110".

Equivalence of Moore and Mealy Machines:

[April 16]

- We will see that the Moore and Mealy machines are equivalent in their capability although these machines differ in their output function. We will discuss how to construct equivalent Mealy machine for a Moore machine and vice-versa.

Theorem: If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore machine, then there is a Mealy machine M_2 equivalent to M_1 and vice-versa.

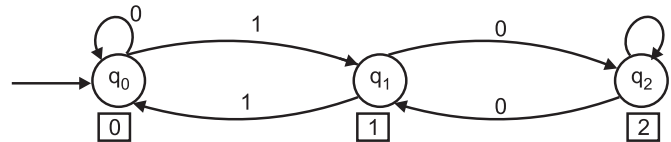
- Moore to Mealy:** To convert Moore machine to Mealy machine, state output symbols are distributed to input symbol paths.
- If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore machine, then equivalent Mealy machine is,

$$M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$$

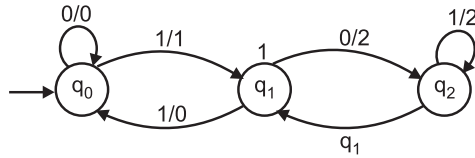
where, $\lambda'(q, a) = \lambda(\delta(q, a))$

\forall (for all) $q \in Q$ and $\forall a \in \Sigma$.

Example 1: Consider a Moore machine that we have already designed (refer Fig. 1.37) for finding residue mod - 3 for any binary number, redrawn in Fig. 1.42 (a).



(a) Moore Machine



(b) Equivalent Mealy Machine

Fig. 1.42

Solution: The state function ' δ ' for this Moore machine is as shown in the Table 1.29.

Table 1.29: Transition table, $\delta: Q \times \Sigma \rightarrow Q$

$\Sigma \backslash Q$	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Similarly, the machine function ' λ ' is as shown in the following Table.

Machine function, $\lambda: Q \rightarrow \Delta$

Q	q_0	q_1	q_2
Δ	0	1	2

We have to convert this Moore machine to equivalent Mealy machine with δ (state function) same as in the Table 1.29, but changed λ' i.e. machine function. According to the conversion rule,

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

$$\text{We can write, } \lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_0) = 0$$

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

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_2) = 2$$

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

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

$$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_2) = 2$$

From this information, we can now create a table for MAF as shown in the Table 1.30.

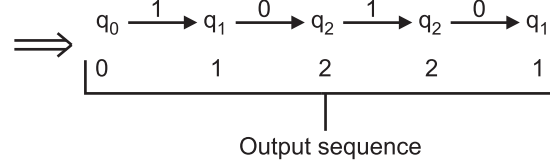
Table 1.30: MAF, $\lambda': Q \times \Sigma \rightarrow \Delta$ for Mealy machine

$\Sigma \backslash Q$	0	1
q_0	0	1
q_1	2	0
q_2	1	2

The transition diagram for the equivalent Mealy machine can be shown in Fig. 1.32 (b). Here, the output is associated with the transition i.e. it depends on the state making the transition as well as on what input the transition is.

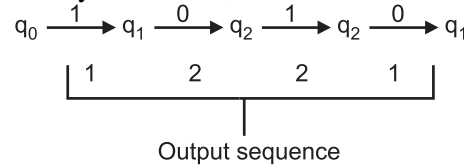
Note: Consider both Moore and Mealy machines shown in Fig. 1.42, also consider the input sequence "1010" of length = 4 i.e., length (input string) = $n = 4$.

(i) Output sequence for Moore machine is,



length (output sequence) = 5 = $n + 1$

(ii) Output sequence for Mealy machine is,



length (output sequence) = 4 = n .

- Thus, for Moore machine, the output sequence contains one more than the symbols in input sequence while in Mealy machine, the output sequence contains exactly the same number of symbols as in the input sequence.
- Mealy to Moore:** To converting the Mealy machine to Moore machine, we will create a separate state for every new output symbol and according to incoming and outgoing edges are distributed.
- If given Mealy machine is,

$$M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0) \text{ then,}$$

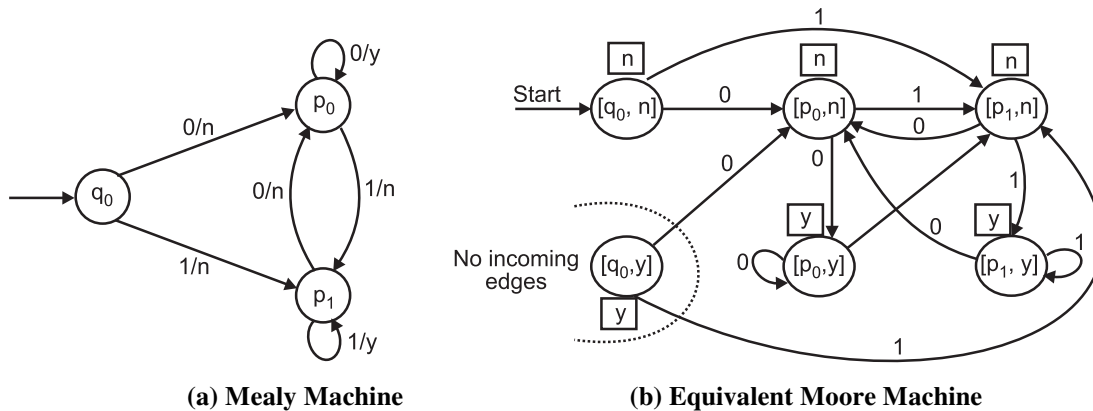
an equivalent Moore machine will be,

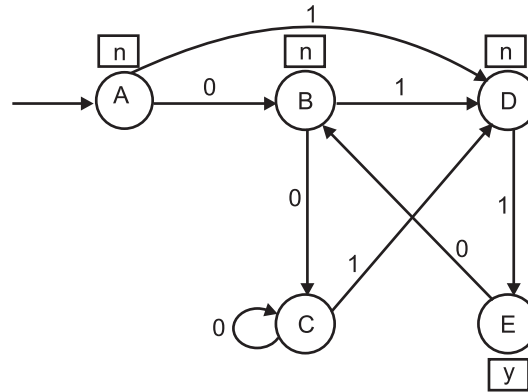
$$M_2 = ([Q \times \Delta], \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$$

where, ' b_0 ' is an arbitrarily selected member of Δ .

and $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$ and $\lambda'([q, b]) = b$

Example 2: Consider the Mealy machine that we have designed accepting the strings ending with '00' and '11' given in Fig. 1.40 and redrawn in Fig. 1.43 (a). The state tables for this machine are as shown in the Table 1.31.





(c) Moore Machine

Fig. 1.43

Table 1.31: Mealy machine state tables

(a) $\delta : Q \times \Sigma \rightarrow Q$

$\Sigma \backslash Q$	0	1
q ₀	p ₀	p ₁
p ₀	p ₀	p ₁
p ₁	p ₀	p ₁

(b) $\lambda : Q \times \Sigma \rightarrow \Delta$

$\Sigma \backslash Q$	0	1
q ₀	n	n
p ₀	y	n
p ₁	n	y

According to the rule of conversion, the resultant Moore machine consists of,

$$Q' = Q \times \Delta = \{[q_0, n], [q_0, y], [p_0, n], [p_0, y], [p_1, n], [p_1, y]\}$$

Also δ' and λ' functions can be found out as,

$$\begin{aligned} \delta'([q_0, n], 0) &= [\delta(q_0, 0), \lambda(q_0, 0)] \\ &= [p_0, n] \end{aligned}$$

$$\begin{aligned} \delta'([q_0, n], 1) &= [\delta(q_0, 1), \lambda(q_0, 1)] \\ &= [p_1, n] \end{aligned}$$

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

$$\begin{aligned} \delta'([q_0, y], 0) &= [\delta(q_0, 0), \lambda(q_0, 0)] \\ &= [p_0, n] \end{aligned}$$

$$\begin{aligned} \delta'([q_0, y], 1) &= [\delta(q_0, 1), \lambda(q_0, 1)] \\ &= [p_1, n] \end{aligned}$$

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

$$\begin{aligned} \delta'([p_0, n], 0) &= [\delta(p_0, 0), \lambda(p_0, 0)] \\ &= [p_0, y] \end{aligned}$$

$$\begin{aligned} \delta'([p_0, n], 1) &= [\delta(p_0, 1), \lambda(p_0, 1)] \\ &= [p_1, n] \end{aligned}$$

$$\lambda'([p_0, n]) = n$$

Similarly,

$$\begin{aligned}
 (d) \quad & \delta'([p_0, y], 0) = [p_0, y] \\
 & \delta'([p_0, y], 1) = [p_1, n] \\
 & \lambda'([p_0, y]) = y \\
 (e) \quad & \delta'([p_1, n], 0) = [\delta(p_1, 0), \lambda(p_1, 0)] \\
 & \quad = [p_0, n] \\
 & \delta'([p_1, n], 1) = [\delta(p_1, 1), \lambda(p_1, 1)] \\
 & \quad = [p_1, y] \\
 & \lambda'([p_1, n]) = n
 \end{aligned}$$

Similarly,

$$\begin{aligned}
 (f) \quad & \delta'([p_1, y], 0) = [p_0, n] \\
 & \delta'([p_1, y], 1) = [p_1, y] \\
 & \lambda'([p_1, y]) = y
 \end{aligned}$$

- The transition graph for equivalent Moore machine is as shown in the Fig. 1.43 (b). We can re-label the states if we want. Out of $[q_0, n]$ and $[q_0, y]$, we have to arbitrarily select the state as initial state and here we have chosen $[q_0, n]$ as a initial state.
- We can remove state $[q_0, y]$ as there are no incoming edges to this state. After removing this state and renaming the remaining states, we get the final Moore machine as shown in the Fig. 1.43 (c).

Mealy Machine vs. Moore Machine:

[April 17]

Sr. No.	Mealy Machine	Moore Machine
1.	In Mealy machine output depends on present state as well as present input.	In Moore machine output depends only upon present state.
2.	Generally, it has fewer states than Moore machine.	Generally, it has fewer states than Moore machine.
3.	In Mealy machine, if input changes, output also changes.	In Moore machine If input changes, output does not change.
4.	They react faster to inputs.	They react slower to inputs (one clock cycle later).
5.	It is difficult to design.	It is easy to design.
6.	In Mealy machine the output is placed on transitions.	In Moore machine the output is placed on states.

1.9 MINIMIZATION OF DFA

[April 16, 17, 18, 19, 20]

- The number of states used in DFA directly depends upon the size of the automaton that we have used. So, it is important to reduce the number of states.
- Minimization of DFA means reducing the number of states from given FA. Thus, we get the FSM with redundant states after minimizing the FSM.

Myhill-Nerode Theorem:

- John Myhill and Anil Nerode of the University of Chicago proposed a theorem in 1958, used to minimization of a DFA.
- Theorem:** For any regular language L , there exists a DFA with total transition function, which accepts L and has a number of states equal to the index of R_L , the relation associated with L .
- Any state in DFA constructed by the Myhill-Nerode theorem, which does not lie on a path from the start state to a final state, can be omitted together with all arcs to or from that state. The remaining DFA is called minimal DFA.
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be any DFA accepting L . We define relations $D_0, D_1 \dots$ on Q as follows:
 - A state 'q' distinguishable from state 'p' by a string of length 0 is written as qD_0p , iff either $q \in F$ and $p \notin F$ or $q \notin F$ and $p \in F$.
 - For $i > 0$ we define $qD_i p$ i.e. q is distinguishable from p by a string of length $\leq i$ iff $qD_{i-1}p$ or there exists $a \in \Sigma$ such that $\delta(q, a)D_{i-1}\delta(p, a)$.
 - State q is distinguishable from p iff qDp .

Example 1: Construct minimal DFA for DFA given in the Fig. 1.44.

[April 16]

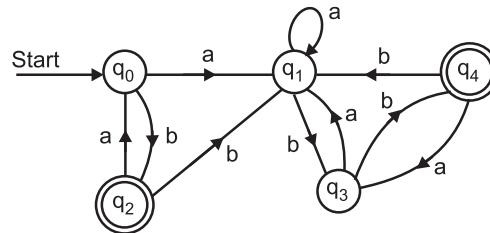


Fig. 1.44

Solution:

δ	a	b
q ₀	q ₁	q ₂
q ₁	q ₁	q ₃
q ₂	q ₀	q ₁
q ₃	q ₁	q ₄
q ₄	q ₃	q ₁

Here q_2 and q_4 are final state. Therefore, mark X at entry (q_2, q_0) (q_2, q_1) (q_4, q_0) (q_4, q_1) (q_4, q_3) as shown in Table 1.32.

Table 1.32

q ₀	—				
q ₁		—			
q ₂	×	×			
q ₃			×	—	
q ₄	×	×		×	—
	q ₀	q ₁	q ₂	q ₃	q ₄

$$\left. \begin{aligned}
 \delta(q_2, a) &= q_0 \notin F \quad \delta(q_2, b) = q_1 \notin F \\
 \delta(q_0, a) &= q_1 \notin F \quad \delta(q_0, b) = q_2 \in F
 \end{aligned} \right\} \therefore q_2, q_0 \text{ are distinguishable}$$

Once we mark entry X for final with nonfinal state i.e. (final, nonfinal); then

First we find D_0 . Table 1.33 shows D_0 . In this we mark entry (p, q) as X if $pD_0 q$ i.e. either $q \in F$ and $p \notin F$ or $q \notin F$ and $p \in F$.

Check for those states which are not distinguishable using D_1 . First we copy distinguishable states i.e. those marked as X from D_0 for Table 1.33. Now q_0 is not distinguishable from q_1 , so we find $q_0 D_1 q_1$. As $\delta(q_0, b) \in F$ and $\delta(q_1, b) \notin F$, q_0 is distinguishable from q_1 with string if length is 1 i.e. $q_0 D_1 q_1$. Also, $\delta(q_1, b) \notin F$ and $\delta(q_3, b) \in F$, so $q_1 D_1 q_3$ as shown in the Table 2.34.

Table 1.33

q ₀	—				
q ₁	×	—			
q ₂	×	×	—		
q ₃		×	×	—	
q ₄	×	×		×	—
	q ₀	q ₁	q ₂	q ₃	q ₄

We do not get any other distinguishable pair with D_1 . We must repeat this till table $D_{i-1} = \text{table } D_i$. As $D_0 \notin D_1$, we find D_2 i.e. with string of length 2. As $\Sigma = \{a, b\}$ possible strings are aa, ab, bb and ba. Now, again copy distinguishable states from D_1 . The table D_2 remains same as D_1 . Hence, we get $[q_0 q_3]$ and $[q_2 q_4]$ as equivalent states. Thus minimal DFA is as shown in the Fig. 1.35.

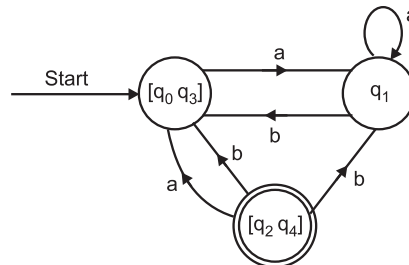


Fig. 1.45

Table 1.34: Transition table

δ	a	b
$[q_0 q_3]$	q_1	$[q_2 q_4]$
q_1	q_1	$[q_0 q_3]$
$[q_2 q_4]$	$[q_0 q_3]$	q_1

Example 2: Construct minimal DFA for the DFA shown in the Fig. 1.46.

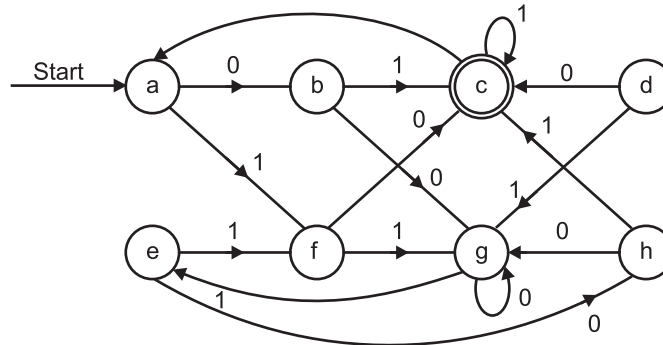


Fig. 1.46: DFA

Table 1.35: Transition table

δ	0	1
a	b	f
b	g	c
c	a	c
d	c	g
e	h	f
f	c	g
g	g	e
h	g	c

Solution: First we mark nonfinal Vs final entries as X. Here c is the final state. So all entries with respect to c (row and column) are mark as X i.e. (c, a) (c, b) (c, d) (c, e) (c, f) and (c, g) and (c, h) are mark as X as shown in Table 1.36.

Then we check for remaining entries in the table.

1. First check (b, a)

$$\left. \begin{array}{l} \delta(b, 0) = g \notin F \quad \delta(b, 1) = c \in F \\ \delta(a, 0) = b \notin F \quad \delta(a, 1) = f \notin F \end{array} \right\} \therefore b \text{ and } a \text{ are distinguishable}$$

2. Check for (d, a)

$$\left. \begin{array}{l} \delta(d, 0) = c \in F \\ \delta(a, 0) = b \notin F \end{array} \right\} \therefore d \text{ and } a \text{ are distinguishable}$$
3. Check for (e, a)

$$\left. \begin{array}{l} \delta(e, 0) = h \notin F \quad \delta(e, 1) = f \\ \delta(a, 0) = b \notin F \quad \delta(a, 1) = f \end{array} \right\} \text{not distinguishable}$$
4. Check for (b, e)

$$\left. \begin{array}{l} \delta(b, 0) = g \quad \delta(b, 1) = c \in F \\ \delta(e, 0) = h \quad \delta(e, 1) = f \notin F \end{array} \right\} \text{distinguishable}$$
5. Check for (f, d)

$$\left. \begin{array}{l} \delta(f, 0) = c \quad \delta(f, 1) = g \\ \delta(d, 0) = c \quad \delta(d, 1) = g \end{array} \right\} \text{not distinguishable}$$
6. Check for (a, h)

$$\left. \begin{array}{l} \delta(a, 0) = b \quad \delta(a, 1) = f \notin F \\ \delta(h, 0) = g \quad \delta(h, 1) = c \in F \end{array} \right\} \text{distinguishable}$$

We continue this for all entries in the table and mark X if they are distinguishable. Otherwise mark as '='.

Table 1.36

a	—						
b	×	—					
c	×	×	—				
d	×	×	×	—			
e		×	×	×	—		
f	×	×	×		×	—	
g	×	×	×	×	×	×	—
h	×		×	×	×	×	×
	a	b	c	d	e	f	g

We conclude that the equivalent states are $a = e$, $b = h$ and $d = f$. Since (a, e), (b, h) and (d, f) are not distinguishable. Therefore they are equivalent. Hence, the minimal finite automaton is as shown in the Fig. 1.47.

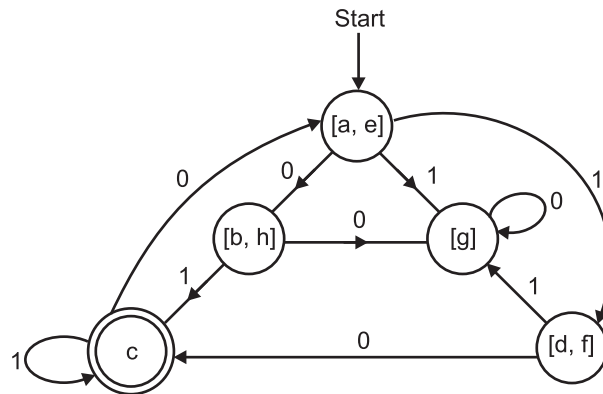


Fig. 1.47: Minimal DFA

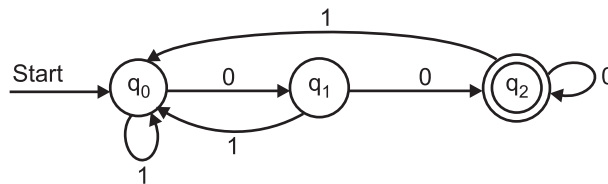
EXAMPLES**[Oct. 17]****Example 1:** Design DFA for language containing strings ending in '00' over alphabet $\{0, 1\}$.**Solution:**

Fig. 1.48: DFA

Table 1.37: Transition table

δ	0	1
start \rightarrow	q ₁	q ₀
	q ₂	q ₀
final \rightarrow	q ₂	q ₀

Example 2: Construct DFA for language whose strings are binary representation of numbers, multiples of 5 and starting with 1 to be accepted.**Solution:** Example. \rightarrow Strings are,

$5 \rightarrow 101,$ $10 \rightarrow 1010,$ $15 \rightarrow 1111,$
 $20 \rightarrow 10100,$ $25 \rightarrow 11001,$ $30 \rightarrow 11110$

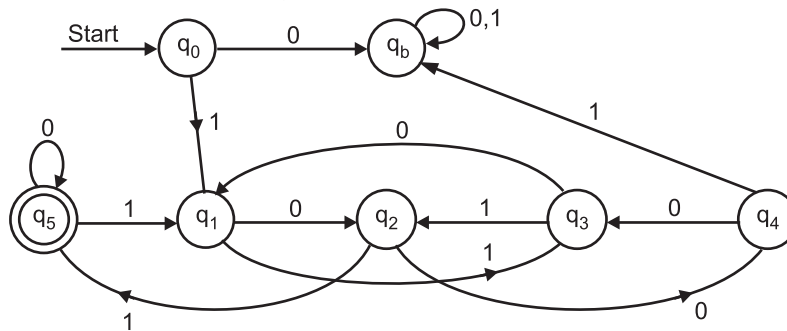


Fig. 1.49: DFA

Table 1.38: Transition table

	δ	0	1
start \rightarrow	q_0	q_6	q_1
	q_1	q_2	q_3
	q_2	q_4	q_5
	q_3	q_1	q_2
final \rightarrow	q_4	q_3	q_6
	q_5	q_5	q_1
	q_6	q_6	q_6

\leftarrow rejected state

Example 3: Construct NFA for language containing all strings over $\{0, 1\}$ such that some two zero's are separated by a string, whose length is $4i$, $i \geq 0$.

Solution:

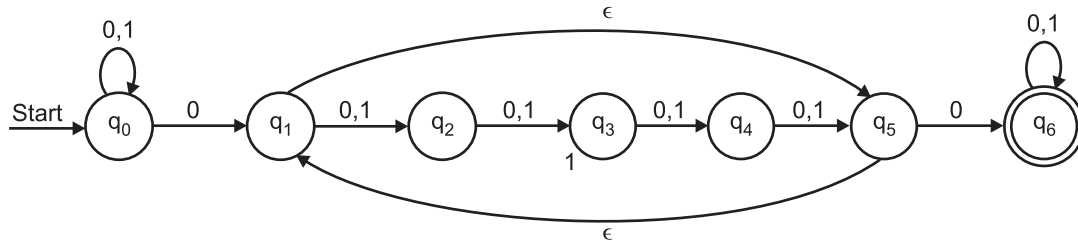


Fig. 1.50: NFA

Example 4: Construct NFA for language containing set of all strings over $\{0, 1\}$ such that 10th symbol from right end is 1.

Solution:

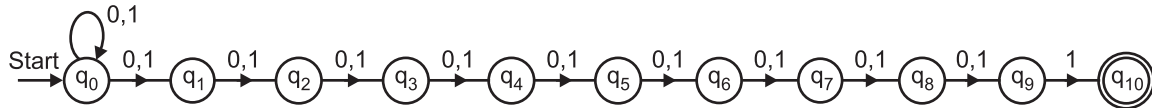


Fig. 1.51: NFA

Example 5: Design Moore and Mealy machine for a binary input sequence such that if it has a substring "101", the machine outputs A, if it has a substring "110" the machine outputs B, otherwise it outputs C.

Solution: To design such machines, it should be checked for sequences 101 and 110. Both these sequences are of length three and starting with the symbol 1, so start state has two paths, one will be on substring 101 and other on 110.

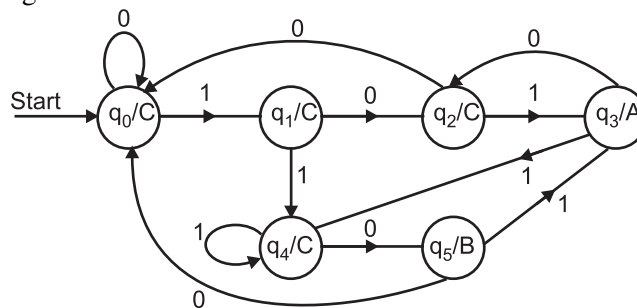


Fig. 1.52 (a): Moore Machine

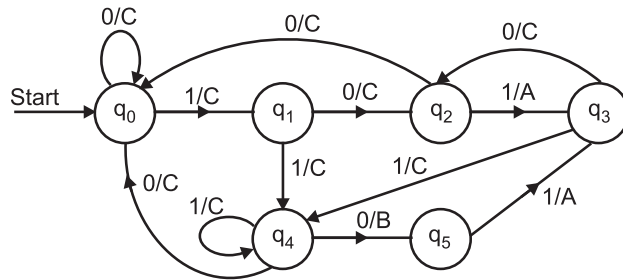


Fig. 1.52 (b): Mealy machine

Example 6: Design a Mealy machine which outputs 1's complement of input string treated as a binary number.

Solution:

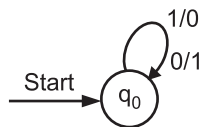


Fig. 1.53: Mealy Machine

Example 7: Construct a Moore machine equivalent to a Mealy machine in Fig. 1.54.

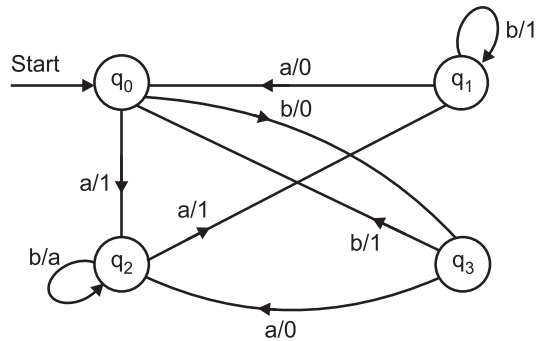


Fig. 1.54: Moore Machine

Solution:

Table 1.39: Transition table

	δ'		Output
	a	b	λ
$[q_0, 0]$	$[q_2, 1]$	$[q_3, 0]$	0
$[q_0, 1]$	$[q_2, 1]$	$[q_3, 0]$	1
$[q_1, 1]$	$[q_0, 0]$	$[q_1, 1]$	1
$[q_2, 0]$	$[q_1, 1]$	$[q_2, 0]$	0
$[q_2, 1]$	$[q_1, 1]$	$[q_2, 0]$	1
$[q_3, 0]$	$[q_2, 0]$	$[q_0, 1]$	0

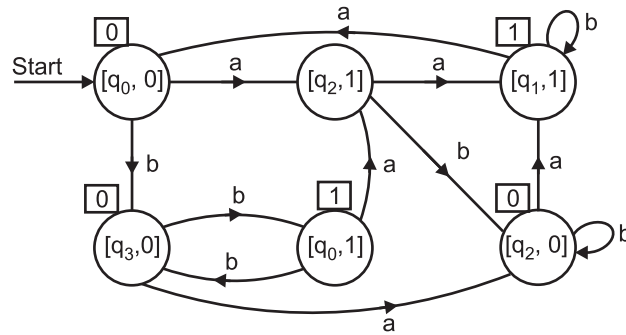


Fig. 1.55: Transition Diagram

Example 8: Draw a DFA for all strings starting with '10' and having substring 201 in it.

Solution:

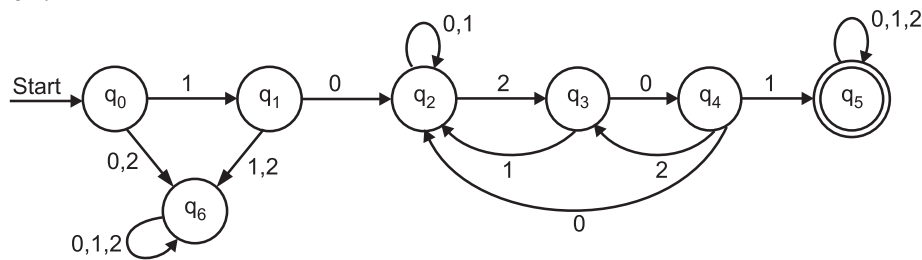


Fig. 1.56: DFA

Table 1.40: Transition table

	δ	0	1	2
Start \rightarrow	q_0	q_6	q_1	q_6
	q_1	q_2	q_6	q_6
	q_2	q_2	q_2	q_3
	q_3	q_4	q_2	q_3
	q_4	q_2	q_5	q_3
Final \rightarrow	q_5	q_5	q_5	q_5
	q_6	q_6	q_6	q_6

Example 9: Draw a DFA for $L = L_1 \cap L_2$ over $\{a, b\}$

where

L_1 = All strings starting with 'a'

L_2 = All strings not having 'ab' as substring.

Solution: L = all string starting with a and not having 'ab' as substring.

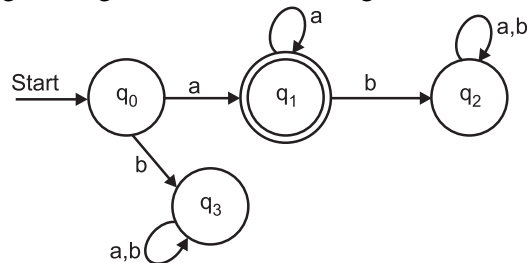


Fig. 1.57: DFA

Table 1.41: Transition table

	δ	a	b	
Start	q_0	q_1	q_3	
	q_1	q_1	q_2	
Final	q_2	q_1	q_2	
	q_3	q_3	q_3	\rightarrow rejected state

Example 10: Construct a DFA for a language over $\{0, 1, 2\}$ which accepts all strings containing atleast one occurrence of a double symbol.

$$L = \{0022, 0011, 0000, 01211, 12122, 2100 \dots\}$$

Solution:

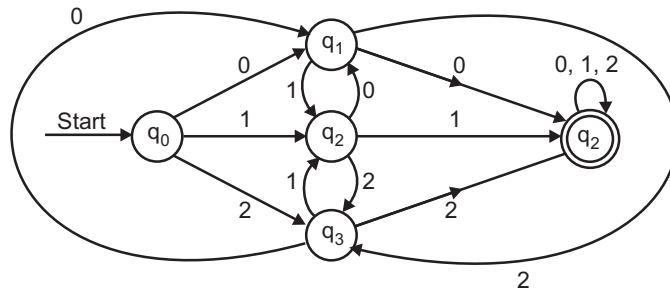


Fig. 1.58: DFA

Table 1.42: Transition table

δ	0	1	2
q_0	q_1	q_2	q_3
q_1	q_4	q_2	q_3
q_2	q_1	q_4	q_3
q_3	q_1	q_2	q_4
q_4	q_4	q_4	q_4

Example 11: Construct a DFA over $\{0, 1\}$ such that every even position is occupied by '0' and odd by '1'.

Solution:

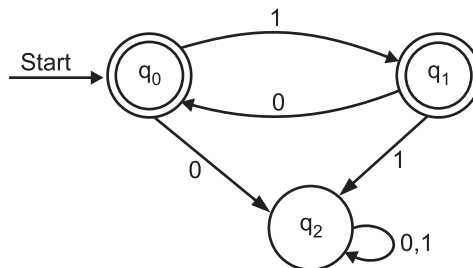


Fig. 1.59: DFA

Table 1.43: Transition table

δ	0	1
q_0	q_2	q_1
q_1	q_0	q_2
q_2	q_2	q_3

Example 12: Construct a DFA to accept the set of all strings over $\{0, 1\}$ such that every pair of adjacent 0's appear before any pair of adjacent 1's.

Solution:

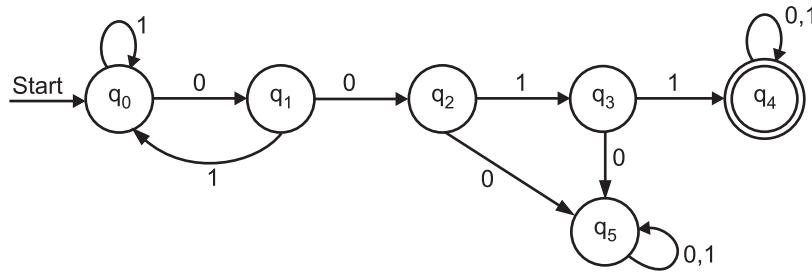


Fig. 1.60: DFA

Table 1.44: Transition table

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_5	q_3
q_3	q_5	q_4
q_4	q_4	q_4
q_5	q_5	q_5

Example 13: Construct a DFA containing all strings starting with 01 and having 012 as substring.

Solution:

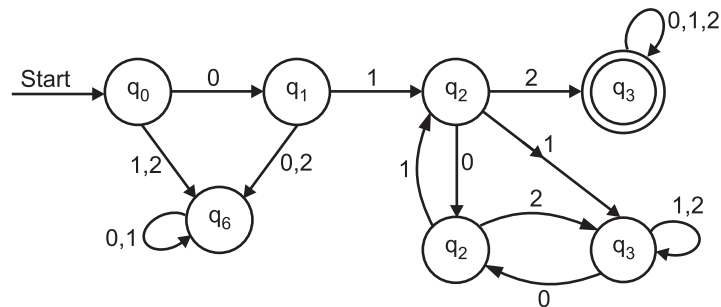


Fig. 1.61: DFA

Table 1.45: Transition table

δ	0	1	2
start \rightarrow			
q_0	q_1	q_6	q_6
q_1	q_6	q_2	q_6
q_2	q_4	q_5	q_3
final \rightarrow			
q_3	q_3	q_3	q_3
q_4	q_4	q_2	q_5
q_5	q_4	q_5	q_5
q_6	q_6	q_6	q_6

\rightarrow rejected state

Example 14: Construct a DFA for

$$L = L_1 \cap L_2 \text{ over } \{a, b\}$$

where

L_1 = all strings starting with a and having even number of b's

L_2 = all strings having odd number of b's

Solution:

$$L = L_1 \cap L_2 = \emptyset.$$

$$L_1 = \{a, abab, abb, abbabab, \dots\}$$

$$L_2 = \{b, ba, babb, abbb, ab, \dots\}$$

No strings are common.

Example 15: Construct a DFA which accepts all strings ends with 1 over $\{0, 1, 2\}$.

Solution:

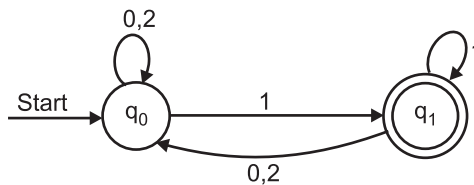


Fig. 1.62: DFA

Table 1.46: Transition table

δ	0	1	2
q_0	q_0	q_1	q_0
q_1	q_0	q_1	q_0

q_1 is a final state.

Example 16: Construct a DFA which accepts all strings having substring 'aba' in it over $\{a, b\}$.

Solution:

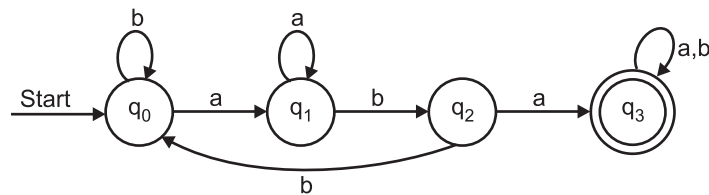


Fig. 1.63: DFA

Table 1.47: Transition table

	δ	a	b
start \rightarrow	q_0	q_1	q_0
	q_1	q_1	q_2
	q_2	q_3	q_0
final \rightarrow	q_3	q_3	q_3

Example 17: Construct DFA to accept all those strings starting with 00 but 00 does not occur anywhere else in the string over $\{0, 1\}$.

Solution:

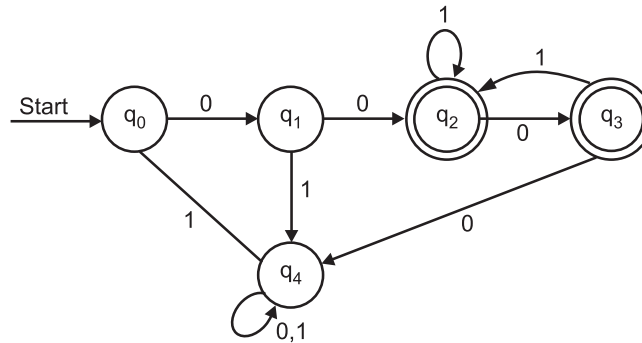


Fig. 1.64: DFA

Table 1.48: Transition table

δ	0	1
q_0	q_1	q_4
q_1	q_2	q_4
q_2	q_3	q_2
q_3	q_4	q_2
q_4	q_4	q_4

Example 18: Construct a NFA to accept string which contains substring 'abbc' in it over $\{a, b, c\}$.

Solution:

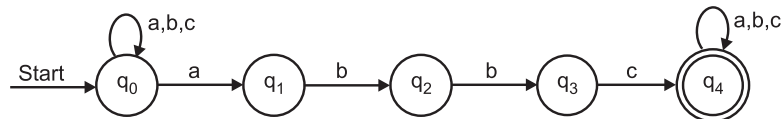


Fig. 1.65: NFA

Table 1.49: Transition table

δ	a	b	c
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$	ϕ
q_2	ϕ	$\{q_3\}$	ϕ
q_3	ϕ	ϕ	$\{q_4\}$
q_4	$\{q_4\}$	$\{q_4\}$	$\{q_4\}$

q_4 is final state.

Example 19: Design a DFA which checks whether a given decimal no. is even or not using binary representation.

Solution: The given decimal number is even when its binary representation contains only '0' at LSB. e.g. 100, 110, etc.

Now when number is entered from LSB, we can have DFA as shown below.

$$\begin{aligned} M &= (Q, \Sigma, \delta, S, F_0) \text{ such that} \\ Q &= \{q_0, q_1\}, \Sigma = \{0, 1\} \\ S &= \{q_0\}, F_0 = \{q_1\} \\ L(M) &= \{\epsilon, 0, 01, 0101, 0111, 01001, \dots\} \end{aligned}$$

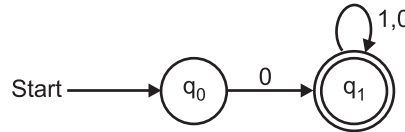


Fig. 1.66: DFA

Table 1.50: Transition table

δ	0	1
q_0	q_1	ϕ
q_1	q_1	q_1

For example, for number 14, we have binary representation as 1110, thus we can have it as from LSB

$\delta(q_0, 1110)$

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

$$\therefore \delta(q_0, 01) = \delta(\delta(q_0, 0), 1) = \delta(q_1, 1) = q_1$$

$$\delta(q_0, 011) = \delta(\delta(q_0, 01), 1) = \delta(q_1, 1) = q_1$$

$$\delta(q_0, 0111) = \delta(\delta(q_0, 011), 1) = \delta(q_1, 1) = q_1$$

As we get final state string is accepted.

Example 20: Design a DFA which checks whether given number is (decimal) divisible by 3.

Solution:

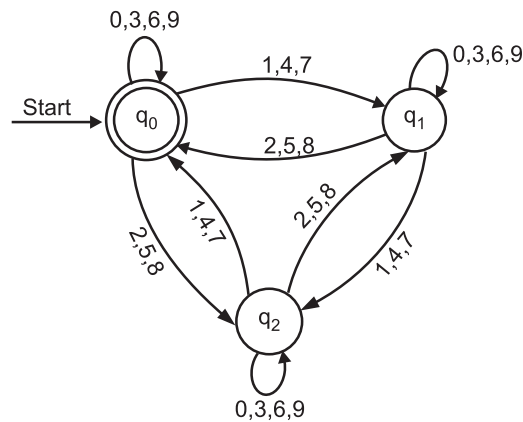


Fig. 1.67: DFA

where

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2\}, \\
 \Sigma &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
 \delta &= Q \times \Sigma \rightarrow Q \\
 q_0 &= q_0 \\
 F &= \{q_0\}
 \end{aligned}$$

Table 1.51: Transition table

δ	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0
q_1	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1
q_2	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2

$$\hat{\delta}(q, \epsilon) = q \text{ and } \delta(q, \omega a) = \delta(\delta(q, \omega), a)$$

Consider 873,

$$\begin{aligned}
 \hat{\delta}(q_0, 8) &= q_2 \\
 \delta(q_0, 87) &= \delta(\delta(q_0, 8), 7) = \delta(q_2, 7) = q_0 \\
 \delta(q_0, 873) &= \hat{\delta}(\delta(q_0, 87), 3) = \delta(q_0, 3) = q_0
 \end{aligned}$$

$\therefore q_0 \in F \therefore$ String 873 is accepted by DFA.

Example 21: Design a DFA which checks whether a binary number is divisible by 5 or not and also show the string 110011 is accepted.

Solution:

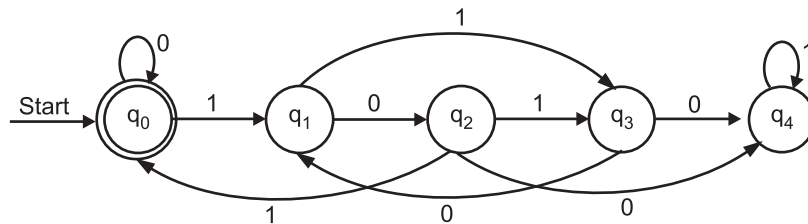


Fig. 1.68: DFA

where

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2, q_3, q_4\}, \Sigma = \{0, 1\} \\
 q_0 &= q_0, F = \{q_0\} \\
 \delta &= Q \times \Sigma \rightarrow Q \text{ as}
 \end{aligned}$$

and

Table 1.52: Transition table

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

$$\hat{\delta}(q_0, a) = q$$

and $\delta(q, wa) = \delta(\delta(q, w), a)$

Consider string 1110011.

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

$$\delta(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_1, 1) = q_3$$

$$\delta(q_0, 111) = \delta(\delta(q_0, 11), 1) = \delta(q_3, 1) = q_2$$

$$\delta(q_0, 1110) = \delta(\delta(q_0, 111), 0) = \delta(q_2, 0) = q_4$$

$$\delta(q_0, 11100) = \delta(\delta(q_0, 1110), 0) = \delta(q_4, 0) = q_3$$

$$\delta(q_0, 111001) = \delta(\delta(q_0, 11100), 1) = \delta(q_3, 1) = q_2$$

$$\delta(q_0, 1110011) = \delta(\delta(q_0, 111001), 1) = \delta(q_2, 1) = q_0$$

q_0 is final state. \therefore String is accepted.

Example 22: Design a DFA which accepts odd number of 1's and even number of 0's over $\{0, 1\}$.

Solution:

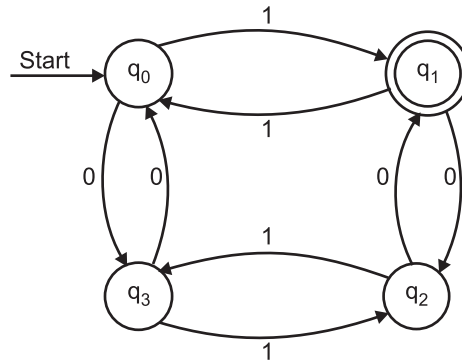


Fig. 1.69: DFA

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

where $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}$

$$q_0 = q_0$$

$$F = \{q_1\}$$

$$\delta = Q \times \Sigma \rightarrow Q$$

$$L(M) = \{11, 101, 011, 00001, 0010101, \dots\}$$

Table 1.53: Transition table

δ	0	1
q_0	q_3	q_1
q_1	q_2	q_0
q_2	q_1	q_3
q_3	q_0	q_2

final state \rightarrow

We have, $\hat{\delta}(q, \epsilon) = q$ and $\delta(q, a) = q'$

$$\delta(q, \omega a) = \delta(\delta(q, \omega), a)$$

Consider a string 0110100.

$$\delta(q_0, 0) = q_3$$

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1) = \delta(q_3, 1) = q_2$$

$$\delta(q_0, 011) = \delta(\delta(q_0, 01), 1) = \delta(q_2, 1) = q_3$$

$$\delta(q_0, 0110) = \delta(\delta(q_0, 011), 0) = \delta(q_3, 0) = q_0$$

$$\delta(q_0, 01101) = \delta(\delta(q_0, 0110), 1) = \delta(q_0, 1) = q_1$$

$$\delta(q_0, 011010) = \delta(\delta(q_0, 01101), 0) = \delta(q_1, 0) = q_2$$

$$\delta(q_0, 0110100) = \delta(\delta(q_0, 011010), 0) = \delta(q_2, 0) = q_1$$

Hence we get $q_1 \in F$

\therefore String is accepted by DFA.

Example 23: Design a DFA which accepts a language

$$L = \{x \mid x \text{ has neither "aa" nor "bb" as a substring}\}$$

Solution:

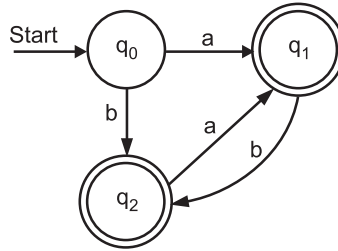


Fig. 1.70: DFA

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

where

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}$$

$$\delta = Q \times \Sigma \rightarrow Q$$

$$q_0 = q_0$$

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

$$L(M) = \{a, b, ab, ba, aba, bab, abab, \dots\}$$

Table 1.54: Transition table

δ	a	b
q_0	q_1	q_2
q_1	ϕ	q_2
q_2	q_1	ϕ

Consider string babab.

$$\begin{aligned}
 \delta(q_0, b) &= q_2 \\
 \delta(q_0, ba) &= \delta(\delta(q_0, b), a) = \delta(q_2, a) = q_1 \\
 \delta(q_0, bab) &= \delta(\delta(q_0, ba), b) = \delta(q_1, b) = q_2 \\
 \delta(q_0, baba) &= \delta(\delta(q_0, bab), a) = \delta(q_2, a) = q_1 \\
 \delta(q_1, babab) &= \delta(\delta(q_0, baba), b) = \delta(q_1, b) = q_2
 \end{aligned}$$

which is a final state.

\therefore String is accepted by DFA.

Example 24: Design a DFA in which every 'b' is preceded by 'a' over a, b, c.

Solution:

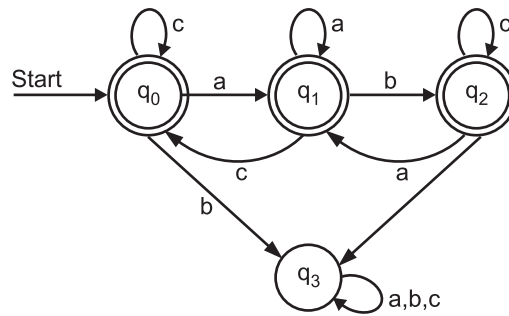


Fig. 1.71: DFA

where

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F_0) \\
 Q &= \{q_0, q_1, q_2\} \\
 \Sigma &= \{a, b, c\} \\
 \delta &= Q \times \Sigma \rightarrow Q \\
 q_0 &= q_0 \\
 F_0 &= \{q_0, q_1, q_2\}
 \end{aligned}$$

Table 1.55: Transition table

δ	a	b	c
q ₀	q ₁	q ₃	q ₀
q ₁	q ₁	q ₂	q ₀
q ₂	q ₁	q ₃	q ₂
q ₃	q ₃	q ₃	q ₃

$$L(M) = \{a, c, ab, ac, cc, aa, abc, ababc, \dots\}$$

Consider a string abcab.

$$\begin{aligned}
 \delta(q_0, a) &= q_1 \\
 \delta(q_0, ab) &= \delta(\delta(q_0, a), b) = \delta(q_1, b) = q_2 \\
 \delta(q_0, abc) &= \delta(\delta(q_0, ab), c) = \delta(q_2, c) = q_2 \\
 \delta(q_0, abca) &= \delta(\delta(q_0, abc), a) = \delta(q_2, a) = q_1 \\
 \delta(q_0, abcab) &= \delta(\delta(q_0, abca), b) = \delta(q_1, b) = q_2 \in F
 \end{aligned}$$

which is a final state.

\therefore String abcab is accepted by this DFA.

Example 25: Design a DFA which accepts all strings over a, b, c such that if it starts with a, then it should contain even number of 'b's. Else if it starts with 'c', then it should contain substring 'cba' in it.

Solution:

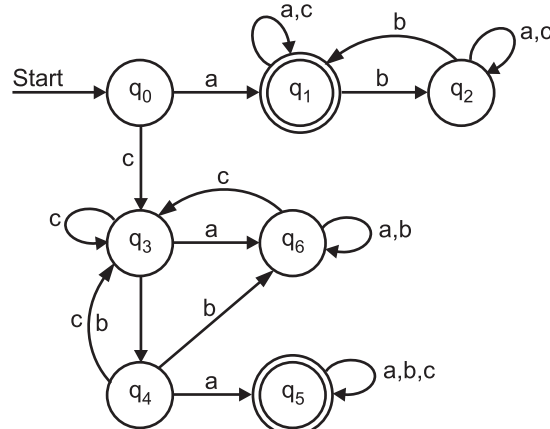


Fig. 1.72: DFA

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\} \\
 \Sigma &= \{a, b, c\} \\
 \delta &= Q \times \Sigma \rightarrow Q \\
 q_0 &= q_0 \\
 F_0 &= \{q_1, q_5\} \\
 L(M) &= \{ab, cba, abcb, acbb, cabcb, abcabc\}
 \end{aligned}$$

Table 1.56: Transition table

δ	a	b	c
q_0	q_1	ϕ	q_3
q_1	q_1	q_2	q_1
q_2	q_2	q_1	q_2
q_3	q_6	q_4	q_3
q_4	q_5	q_6	q_3
q_5	q_5	q_5	q_5
q_6	q_6	q_6	q_3

Consider string 'cbcbca'

$$\begin{aligned}
 \delta(q_0, c) &= q_3 \\
 \delta(q_0, cb) &= \delta(\delta(q_0, c), b) = \delta(q_3, b) = q_4 \\
 \delta(q_0, cbb) &= \delta(\delta(q_0, cb), b) = \delta(q_4, b) = q_6 \\
 \delta(q_0, cbbc) &= \delta(\delta(q_0, cbb), c) = \delta(q_6, c) = q_3 \\
 \delta(q_0, cbbcb) &= \delta(\delta(q_0, cbbc), b) = \delta(q_3, b) = q_4 \\
 \delta(q_0, cbbcba) &= \delta(\delta(q_0, cbbcb), a) = \delta(q_4, a) = q_5
 \end{aligned}$$

$q_5 \in F$

\therefore String is accepted by DFA.

Example 26: Design a DFA which accepts all strings over $\{0, 1, 2\}$ such that if it starts with '0' then contains substring '01' else it should contain substring '112' in it.

Solution:

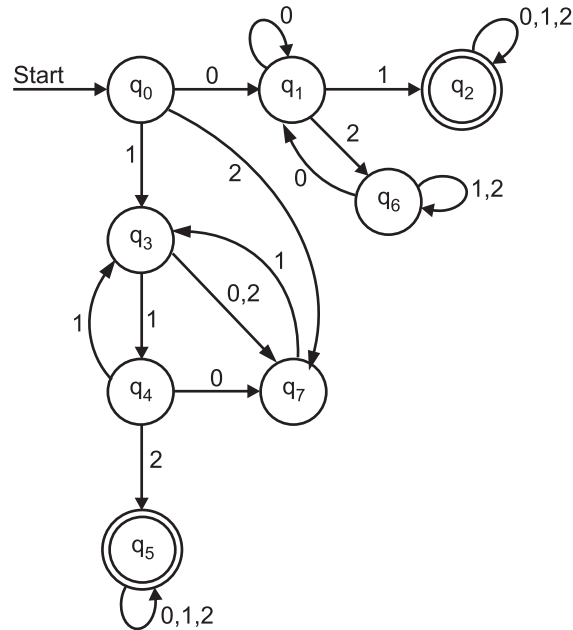


Fig. 1.73: DFA

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

where

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$$

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

$$F = \{q_2, q_5\}$$

$$q_0 = q_0$$

Table 1.57: Transition table

δ	0	1	2
q_0	q_1	q_3	q_7
q_1	q_1	q_2	q_6
q_2	q_2	q_2	q_2
q_3	q_7	q_4	q_7
q_4	q_7	q_3	q_5
q_5	q_5	q_5	q_5
q_6	q_1	q_6	q_6
q_7	q_7	q_3	q_7

Consider the string 001101.

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

$$\delta(q_0, 00) = \delta(\delta(q_0, 0), 0) = \delta(q_1, 0) = q_1$$

$$\delta(q_0, 001) = \delta(\delta(q_0, 01), 1) = \delta(q_1, 1) = q_2$$

$$\begin{aligned}\delta(q_0, 0011) &= \delta(\delta(q_0, 001), 1) = \delta(q_2, 1) = q_2 \\ \delta(q_0, 00110) &= \delta(\delta(q_0, 0011), 0) = \delta(q_2, 0) = q_2 \\ \delta(q_0, 001101) &= \delta(\delta(q_0, 00110), 1) = \delta(q_2, 1) = q_2 \in F\end{aligned}$$

\therefore String is accepted.

Example 27: Design a DFA which accepts all strings not having abc as substring over {a, b, c}.

Solution:

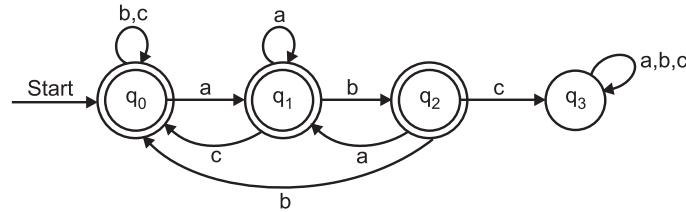


Fig. 1.74: DFA

where

$$\begin{aligned}M &= (Q, \Sigma, \delta, q_0, F) \\ Q &= \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b, c\} \\ q_0 &= q_0 \\ F &= \{q_0, q_1, q_2\} \\ \delta &= Q \times \Sigma \rightarrow Q\end{aligned}$$

Table 1.58: Transition table

δ	a	b	c
q_0	q_1	q_0	q_0
q_1	q_1	q_2	q_0
q_2	q_1	q_0	q_3
q_3	q_3	q_3	q_3

Consider the string acabc.

$$\begin{aligned}\delta(q_0, a) &= q_1 \\ \delta(q_0, ac) &= \delta(\delta(q_0, a), c) = \delta(q_1, c) = q_0 \\ \delta(q_0, aca) &= \delta(\delta(q_0, ac), a) = \delta(q_0, a) = q_1 \\ \delta(q_0, acab) &= \delta(\delta(q_0, aca), b) = \delta(q_1, b) = q_2 \\ \delta(q_0, acabc) &= \delta(\delta(q_0, acab), c) = \delta(q_2, c) = q_3\end{aligned}$$

As $q_3 \notin F$, \therefore String is not accepted.

Example 28: Design a FA which reads strings made up of letters in the word CHAPTER and recognize those strings containing word CAP as substring.

Solution:

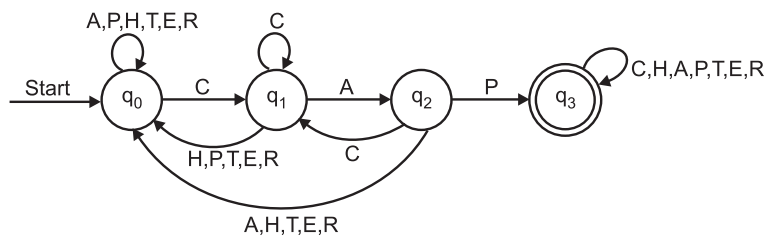


Fig. 1.75: FA

where

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2, q_3\} \\
 \Sigma &= \{C, H, A, P, T, E, R\} \\
 \delta &= Q \times \Sigma \rightarrow Q \\
 q_0 &= q_0 \\
 F &= \{q_3\} \\
 L(M) &= \{CAP, ACAP, ACAPTER, CAHTCPCACAPR\}
 \end{aligned}$$

Table 1.59: Transition table

δ	C	H	A	P	T	E	R
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_3	q_0	q_0	q_0
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

Consider a string HCPCAPR.

$$\begin{aligned}
 \delta(q_0, H) &= q_0 \\
 \delta(q_0, HC) &= \delta(\delta(q_0, H), C) = \delta(q_0, C) = q_1 \\
 \delta(q_0, HCP) &= \delta(\delta(q_0, HC), P) = \delta(q_1, P) = q_0 \\
 \delta(q_0, HCPC) &= \delta(\delta(q_0, HCP), C) = \delta(q_0, C) = q_1 \\
 \delta(q_0, HCPCA) &= \delta(\delta(q_0, HCPC), A) = \delta(q_1, A) = q_2 \\
 \delta(q_0, HCACAP) &= \delta(\delta(q_0, HCPCA), P) = \delta(q_0, P) = q_3 \\
 \therefore \delta(q_0, HCPCAPR) &= \delta(\delta(q_0, HCPCAP), R) = \delta(q_3, R) = q_3
 \end{aligned}$$

As $q_3 \in F$, \therefore String is accepted by DFA.

Example 29: Design a NFA which accepts all strings containing even number of 'a's over $\{a, b\}$.

Solution:

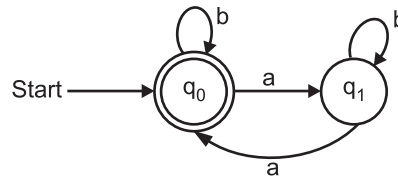


Fig. 1.76: NFA

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1\} \\
 \Sigma &= \{a, b\} \\
 \delta &= Q \times \Sigma \rightarrow 2^Q \\
 q_0 &= q_0 \\
 F &= \{q_0\} \\
 L(M) &= \{\epsilon, b, aa, baba, bbaba, aaaa, \dots\}
 \end{aligned}$$

Table 1.60

δ	a	b
q_0	q_1	q_0
q_1	q_0	q_1

Consider string 'babab'.

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

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

$$\delta(q_0, bab) = \delta(\delta(q_0, ba), b) = \delta(q_1, b) = q_1$$

$$\delta(q_0, baba) = \delta(\delta(q_0, bab), b) = \delta(q_1, a) = q_0$$

$$\therefore \delta(q_0, babab) = \delta(\delta(q_0, baba), b) = \delta(q_0, b) = q_0$$

As $q_0 \in F$, \therefore String is accepted by FA.

Example 30: Design a NFA which accepts the language with strings having even number of 'a's and odd number of 'b's over $\{a, b\}$.

Solution:

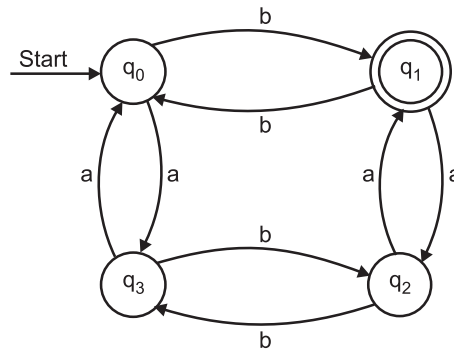


Fig. 1.77: NFA

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

where

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

$$\Sigma = \{a, b\}$$

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

$$q_0 = q_0$$

$$F = \{q_1\}$$

$$L(M) = \{b, aab, aba, baa, baabb, \dots\}$$

Table 1.61: Transition table

δ	a	b
q_0	q_3	q_1
q_1	q_2	q_0
q_2	q_1	q_3
q_3	q_0	q_2

Consider string 'abab'.

$$\delta(q_0, a) = q_3$$

$$\delta(q_0, ab) = \delta(\delta(q_0, a), b) = \delta(q_3, b) = q_2$$

$$\delta(q_0, aba) = \delta(\delta(q_0, ab), a) = \delta(q_2, a) = q_1$$

$$\therefore \delta(q_0, abab) = \delta(\delta(q_0, aba), b) = \delta(q_1, b) = q_0$$

As $q_0 \notin F$, \therefore String is not accepted by FA.

Example 31: Design a NFA to accept string containing 'aa' but not 'bb' over $\{a, b\}$.

Solution:

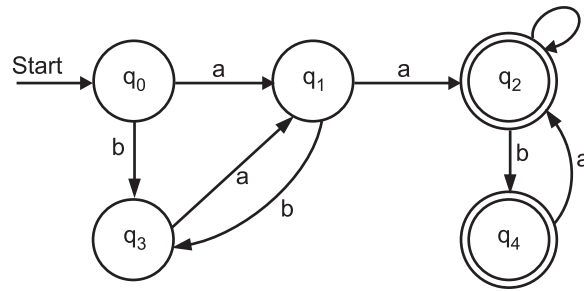


Fig. 1.78: NFA

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

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

$$\Sigma = \{a, b\}$$

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

$$q_0 = q_0$$

$$F = \{q_2, q_4\}$$

$$L(M) = \{aa, aab, aaba, babaa, \dots\}$$

Table 1.62: Transition table

δ	a	b
q_0	q_1	q_3
q_1	q_2	q_3
q_2	q_2	q_4
q_3	q_1	ϕ
q_4	q_2	ϕ

Consider a string 'abaab'.

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

$$\delta(q_0, ab) = \delta(\delta(q_0, a), b) = \delta(q_1, b) = q_3$$

$$\delta(q_0, aba) = \delta(\delta(q_0, ab), a) = \delta(q_3, a) = q_1$$

$$\delta(q_0, abaa) = \delta(\delta(q_0, aba), a) = \delta(q_1, a) = q_2$$

$$\therefore \delta(q_0, abaab) = \delta(\delta(q_0, abaa), b) = \delta(q_2, b) = q_4$$

As $q_4 \in F$, \therefore String is accepted by FA.

Example 32: Design a NFA with second last symbol is 'a' over {a, b}.

Solution:

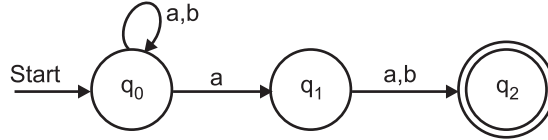


Fig. 1.79: NFA

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

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

$$\Sigma = \{a, b\}$$

$$\delta = Q \times \Sigma \rightarrow 2^Q$$

$$q_0 = q_0$$

$$F = \{q_2\}$$

$$L(M) = \{aa, ab, aab, abab, \dots\}$$

Table 1.63: Transition table

δ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ

Consider string 'abaa'.

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

$$\begin{aligned} \delta(q_0, ab) &= \delta(\delta(q_0, a), b) = \delta(q_0, a) \cup \delta(q_1, b) \\ &= \{q_0, q_2\} \end{aligned}$$

$$\begin{aligned} \delta(q_0, aba) &= \delta(\delta(q_0, ab), a) = \delta(q_0, a) \cup \delta(q_2, a) \\ &= \{q_0, q_1\} \end{aligned}$$

$$\begin{aligned} \delta(q_0, abaa) &= \delta(\delta(q_0, aba), a) = \delta(q_0, a) \cup \delta(q_1, a) \\ &= \{q_0, q_1, q_2\} \end{aligned}$$

But $q_0, q_1 \notin F, q_2 \in F$

\therefore String is accepted.

Example 33: Design NFA which accepts all strings containing every symbol in alphabetical order over $\Sigma = \{p, q, a, b\}$.

Solution:

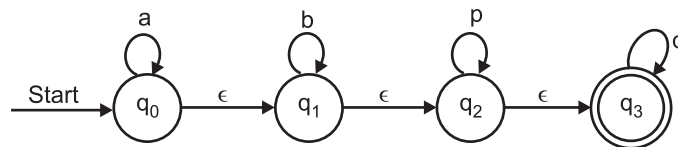


Fig. 1.80: NFA

$$\begin{aligned}
 M &= (Q, \Sigma, \delta, q_0, F) \\
 Q &= \{q_0, q_1, q_2, q_3\} \\
 \Sigma &= \{a, b, p, q\} \\
 q_0 &= q_0 \\
 F &= \{q_3\} \\
 L(M) &= \{a, b, p, q, abbpp, qq\}
 \end{aligned}$$

Table 1.64: Transition table

δ	a	b	p	q
q_0	q_0	ϕ	ϕ	ϕ
q_1	ϕ	q_1	ϕ	ϕ
q_2	ϕ	ϕ	q_2	ϕ
q_3	ϕ	ϕ	ϕ	q_3

Example 34: Construct a DFA to accept the language

$$L = \{\omega \mid \omega \text{ is of even length and begins with } 01\}$$

Solution:

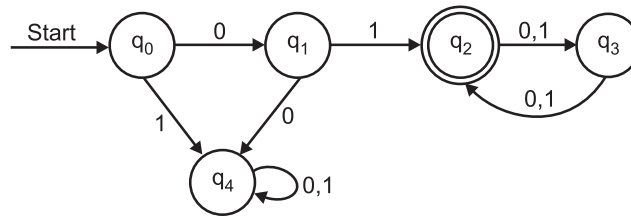


Fig. 1.81: DFA

Example 35: Construct DFA for the languages which checks whether a given decimal number is even.

Solution: A number is even means it is a multiple of 2. i.e. it should be divisible by 2.

\therefore The strings accepted by this language are: 2, 4, 6, 8, 10, 12, 14,

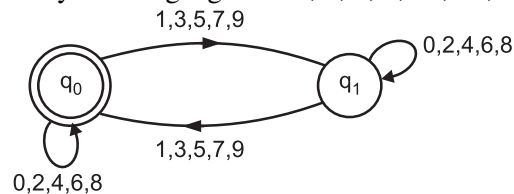


Fig. 1.82: DFA

Table 1.65: Transition table

δ	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1
q_1	q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0	q_1	q_0

Let us take decimal number which is even i.e. 14.

$$\therefore 14 = 7 + 7$$

$$\therefore \delta(q_0, 7) = q_1$$

$$\delta(q_0, 14) = \delta(\delta(q_0, 7), 7) = \delta(q_1, 7) = q_0$$

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

where $Q =$ Finite set of states i.e. $\{q_0, q_1\}$

$$\Sigma = \{0, 1, 2, \dots, 9\}$$

$$\delta = Q \times \Sigma \rightarrow Q$$

$q_0 =$ Initial state

Final state (F) = $\{q_0\}$

Example 36: Construct a DFA for a languages over alphabets $\{a, b, c\}$ which accepts all strings containing 'a' at every even position in the string.

Solution:

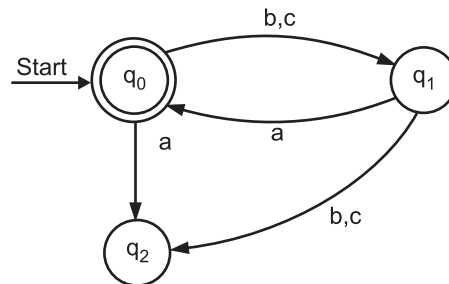


Fig. 1.83: DFA

Table 1.66: Transition table

δ	a	b	c
q_0	q_2	q_1	q_1
q_1	q_0	q_2	q_2
q_2	q_2	q_2	q_2

EXAMPLES (MINIMIZATION OF DFA)

Example 1: Construct the minimize DFA for the following DFA.

	δ	0	1
Start	A	B	F
	B	G	C
Final	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

Solution:

Table 1.67

A	-							
B	x	-						
C	x	x	-					
D	x	x	x	-				
E	=	x	x	x	-			
F	x	x	x	=	x	-		
G	x	x	x	x	x	x	-	
H	x	=	x	x	x	x	x	-
	A	B	C	D	E	F	G	H

1. Draw grid.
2. Marked the entries with hyphen '-' as shown above.
3. From transition table we can directly find $q \not\sim p$ such that either $q \in F$ and $p \notin F$ or $q \notin F$ and $p \in F$. Mark the entries 'X'. This process is continue for length of string more than $(1 \geq 1)$ i.e. D_1, D_2, \dots

First we marked some entries with Do from transition table by checking non-final and final states. The remaining entries we can proceed further with more length as follows:

(i) $E D_2 A$

$$\begin{array}{lcl} E \xrightarrow{0} H \xrightarrow{0} G \xrightarrow{0} G & & GE \xrightarrow{1} F \xrightarrow{1} G \xrightarrow{1} E \\ A \xrightarrow{0} B \xrightarrow{0} G \xrightarrow{0} G & & GA \xrightarrow{1} F \xrightarrow{1} G \xrightarrow{1} E \end{array}$$

Both states shown with circle are non-final and same, so we can't distinguish A and E for $i = 2$.

(ii) $G D_2 A$

$$\begin{array}{lcl} G \xrightarrow{0} G \xrightarrow{0} G \xrightarrow{0} G & & G \xrightarrow{1} E \xrightarrow{1} F \xrightarrow{1} G \\ A \xrightarrow{0} B \xrightarrow{0} G \xrightarrow{0} G & & A \xrightarrow{1} F \xrightarrow{1} G \xrightarrow{1} E \end{array}$$

distinguishable

(iii) $H D_2 B$

$$\begin{array}{lcl} H \xrightarrow{0} G \xrightarrow{0} G \xrightarrow{0} G & & H \xrightarrow{1} C \xrightarrow{1} C \xrightarrow{1} C \\ B \xrightarrow{0} G \xrightarrow{0} G \xrightarrow{0} G & & B \xrightarrow{1} C \xrightarrow{1} C \xrightarrow{1} C \end{array}$$

can't distinguish

(iv) $F D_2 D$

$$\begin{array}{l} F \xrightarrow{0} C \xrightarrow{0} A \xrightarrow{0} B \\ D \xrightarrow{0} C \xrightarrow{0} A \xrightarrow{0} B \end{array}$$

$$\begin{array}{l} F \xrightarrow{1} G \xrightarrow{1} E \xrightarrow{1} F \\ D \xrightarrow{1} G \xrightarrow{1} E \xrightarrow{1} F \end{array}$$

can't distinguish

(v) $G D_2 E$

$$\begin{array}{l} G \xrightarrow{0} G \xrightarrow{0} G \xrightarrow{0} G \\ E \xrightarrow{0} H \xrightarrow{0} G \xrightarrow{0} G \end{array}$$

$$\begin{array}{l} G \xrightarrow{1} E \xrightarrow{1} F \xrightarrow{1} G \\ E \xrightarrow{1} F \xrightarrow{1} G \xrightarrow{1} E \end{array}$$

distinguishable

Therefore we will combine the following states which are non-distinguishable.

(a) A and E

(b) B and H

(c) F and D

C is final state.

Transition table for minimized DFA is

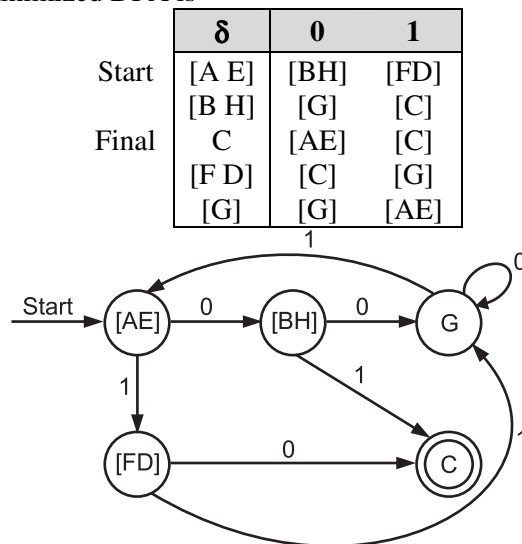


Fig. 1.84

Example 2: Construct the minimize DFA for the following DFA.

Table 1.68

δ	0	1
Start	a	b
	b	a
	c	d
Final	* d	b
	e	d
	f	d
	g	f
	h	g

Solution:

Table 1.69

a	-							
b	×	-						
c	×	×	-					
d	×	×	×	-				
e	×	×	×	×	-			
f	×	×	×	×	×	-		
g	×	×	×	×	×	×	-	
h	×		×	×	×	×	×	-
	a	b	c	d	e	f	g	h

We direct proceed for D_2 entries.

(i) $b D_2 a$

$$\begin{array}{c} 0 \quad 0 \quad 0 \\ b \longrightarrow a \longrightarrow b \longrightarrow a \\ 0 \quad 0 \quad 0 \\ a \longrightarrow b \longrightarrow a \longrightarrow b \end{array}$$

$$\begin{array}{c} 1 \quad 1 \quad 1 \\ b \longrightarrow c \longrightarrow b \longrightarrow c \\ 1 \quad 1 \quad 1 \\ a \longrightarrow a \longrightarrow a \longrightarrow a \end{array}$$

(ii) $f D_2 a$

$$\begin{array}{c} 0 \quad 0 \quad 0 \\ f \longrightarrow g \longrightarrow f \longrightarrow g \\ 0 \quad 0 \quad 0 \\ a \longrightarrow b \longrightarrow a \longrightarrow b \end{array}$$

$$\begin{array}{c} 1 \quad 1 \quad 1 \\ f \longrightarrow e \longrightarrow f \longrightarrow e \\ 1 \quad 1 \quad 1 \\ a \longrightarrow a \longrightarrow a \longrightarrow a \end{array}$$

(iii) $g D_2 a$

$$\begin{array}{c} 0 \quad 0 \quad 0 \\ g \longrightarrow f \longrightarrow g \longrightarrow f \\ 0 \quad 0 \quad 0 \\ a \longrightarrow b \longrightarrow a \longrightarrow b \end{array}$$

$$\begin{array}{c} 1 \quad 1 \quad 1 \\ g \longrightarrow g \longrightarrow g \longrightarrow g \\ 1 \quad 1 \quad 1 \\ a \longrightarrow a \longrightarrow a \longrightarrow a \end{array}$$

(iv) $f D_2 b$

$$\begin{array}{c} 0 \quad 0 \quad 0 \\ f \longrightarrow g \longrightarrow f \longrightarrow g \\ 0 \quad 0 \quad 0 \\ b \longrightarrow a \longrightarrow b \longrightarrow a \end{array}$$

$$\begin{array}{c} 1 \quad 1 \quad 1 \\ f \longrightarrow e \longrightarrow f \longrightarrow e \\ 1 \quad 1 \quad 1 \\ b \longrightarrow c \longrightarrow b \longrightarrow c \end{array}$$

(v) $g D_2 b$

$$\begin{array}{c} 0 \quad 0 \quad 0 \\ g \longrightarrow f \longrightarrow g \longrightarrow f \\ 0 \quad 0 \quad 0 \\ b \longrightarrow a \longrightarrow b \longrightarrow a \end{array}$$

$$\begin{array}{c} 1 \quad 1 \quad 1 \\ g \longrightarrow g \longrightarrow g \longrightarrow g \\ 1 \quad 1 \quad 1 \\ b \longrightarrow c \longrightarrow b \longrightarrow c \end{array}$$

(vi) $g D_2 f$

$$\begin{array}{ccccc} & 0 & & 0 & & 0 \\ g & \xrightarrow{\quad} & f & \xrightarrow{\quad} & g & \xrightarrow{\quad} & f \\ & 0 & & 0 & & 0 \\ f & \xrightarrow{\quad} & g & \xrightarrow{\quad} & f & \xrightarrow{\quad} & g \end{array}$$

$$\begin{array}{ccccccc} & 1 & & 1 & & 1 \\ g & \xrightarrow{\quad} & g & \xrightarrow{\quad} & g & \xrightarrow{\quad} & g \\ & 1 & & 1 & & 1 \\ f & \xrightarrow{\quad} & e & \xrightarrow{\quad} & f & \xrightarrow{\quad} & e \end{array}$$

Similarly $e D_2 c$ and $c D_2 b$ are also distinguishable. Like this if we proceed for more length we will not get here any non-distinguishable states. Therefore this DFA cannot be minimize.

Example 3: Construct minimal DFA for the following DFA.

	δ	0	1
Start	q_0	q_4	q_0
Final	q_1	q_1	q_0
	q_2	q_1	q_3
	q_3	q_7	q_2
	q_4	q_0	q_5
	q_5	q_1	q_4
	q_6	q_7	q_1
	q_7	q_3	q_7

Solution: $q_0 D_0 q_1$

$$q_0 \xrightarrow{0} q_4 \notin F$$

$$q_1 \xrightarrow{0} q_1 \in F \text{ distinguishable mark X.}$$

$q_0 D_0 q_2$

$$q_0 \xrightarrow{0} q_4 \notin F$$

$$q_0 \xrightarrow{1} q_0$$

$$q_2 \xrightarrow{0} q_1 \in F$$

$$q_2 \xrightarrow{1} q_3$$

Mark X.

Similarly make all D_0 entries then D_1 entries

e.g. $q_3 D_1 q_0$

$$q_3 \xrightarrow{0} q_7 \xrightarrow{0} q_3$$

$$q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_3$$

$$q_0 \xrightarrow{0} q_4 \xrightarrow{0} q_0$$

$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0$$

Both reachable states are different. Hence, proceed with D_2 entries.

(i) $q_3 D_2 q_0$

$$q_3 \xrightarrow{0} q_7 \xrightarrow{0} q_3 \xrightarrow{0} q_7$$

$$q_3 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_2$$

$$q_0 \xrightarrow{0} q_4 \xrightarrow{0} q_0 \xrightarrow{0} q_4$$

$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0$$

Again distinguishable

(ii) $q_4 D_2 q_0$

$$\begin{array}{l} q_4 \xrightarrow{0} q_0 \xrightarrow{0} q_4 \xrightarrow{0} q_0 \\ q_0 \xrightarrow{0} q_4 \xrightarrow{0} q_0 \xrightarrow{0} q_4 \end{array}$$

$$\begin{array}{l} q_4 \xrightarrow{1} q_5 \xrightarrow{1} q_4 \xrightarrow{1} q_5 \\ q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \end{array}$$

(iii) $q_7 D_2 q_0$

$$\begin{array}{l} q_7 \xrightarrow{0} q_3 \xrightarrow{0} q_7 \xrightarrow{0} q_3 \\ q_0 \xrightarrow{0} q_4 \xrightarrow{0} q_0 \xrightarrow{0} q_4 \end{array}$$

$$\begin{array}{l} q_7 \xrightarrow{1} q_7 \xrightarrow{1} q_7 \xrightarrow{1} q_7 \\ q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \end{array}$$

Similarly,

$$q_5 D_2 q_2$$

$$q_3 D_2 q_4$$

$$q_3 D_2 q_7$$

$$q_4 D_2 q_7$$

All these remaining entries also we get as distinguishable as shown below.

Table 1.70

a	—							
b	×	—						
c	×	×	—					
d	×	×	×	—				
e	×	×	×	×	—			
f	×	×	×	×	×	—		
g	×	×	×	×	×	×	—	
h	×		×	×	×	×	×	—

Since, all entries are distinguishable. We cannot minimize the DFA.

EXAMPLES (FA WITH OUTPUT)

Example 1: Design a Mealy machine which outputs even or odd according to number of 1's encountered is even or odd over $\{0, 1\}$.

Solution:

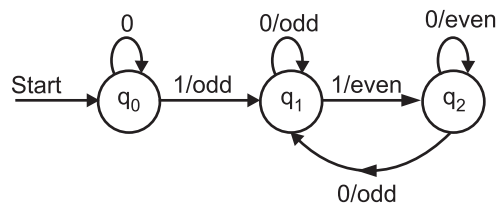


Fig. 1.85

Example 2: Construct a Moore machine that outputs valid or invalid for a language

$$L = a(a+b)^*b.$$

Solution:

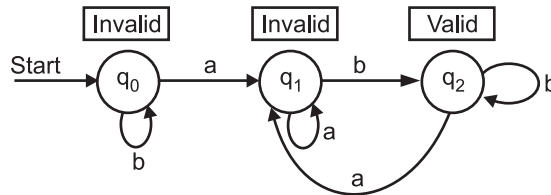


Fig. 1.86

Example 3: Design a Mealy machine for input $(a + b + c)^*$ if input ends with 'bac', print A else print B.

Solution:

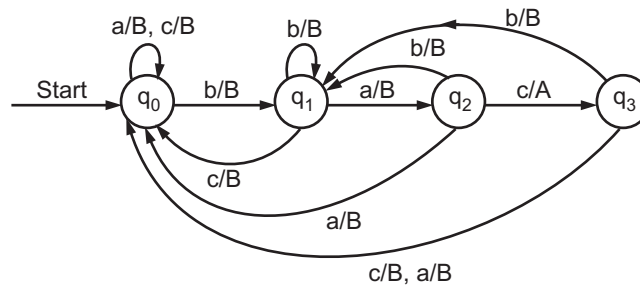


Fig. 1.87

Example 4: Design a Moore machine for binary input sequence such that if it has a substring 101, the machine output A, if it has substring 110 then outputs B else outputs C.

Solution:

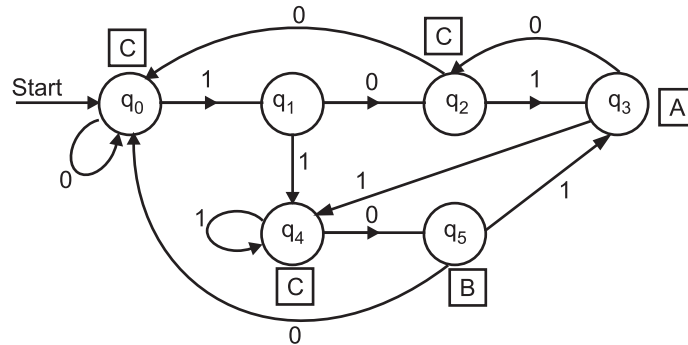


Fig. 1.88

Example 5: Construct a Moore machine to convert each occurrence of substring 100 by 101.

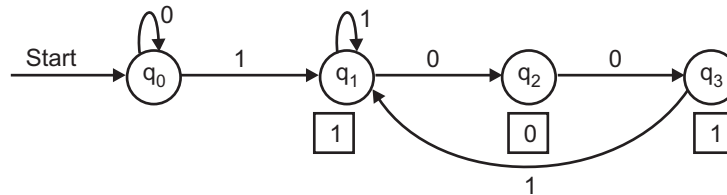


Fig. 1.89

Example 6: Design Mealy machine for a binary input sequence such that if it has a substring 101, the machine outputs A, if it has a substring 110, the machine outputs B otherwise it outputs C.

Solution:

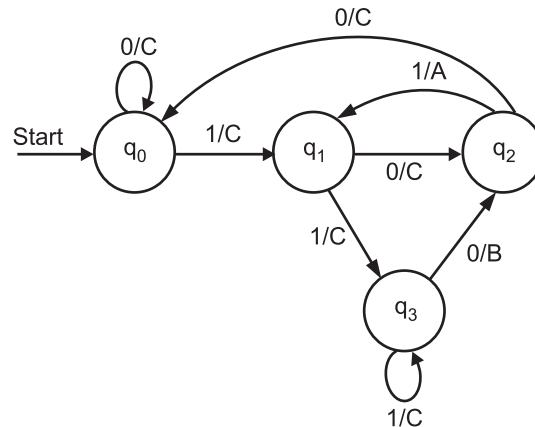


Fig. 1.90

EXAMPLES

Example 1: Construct NFA for language L, where $L = \{a(a+b)^*b\}$.

Solution:

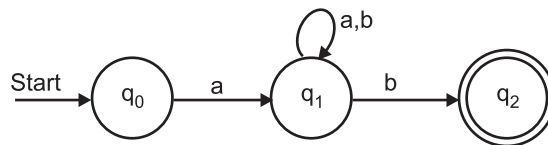


Fig. 1.91

Example 2: Construct DFA equivalent to NFA

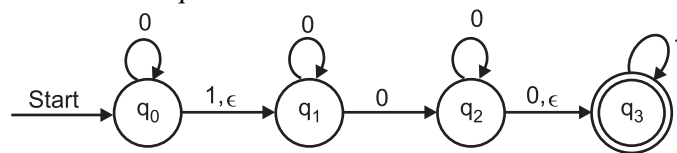


Fig. 1.92

Solution: (1) To find initial state, find ϵ -closure of (q_0) .

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1\} = A \text{ (say)}$$

From A on 0

$$\epsilon\text{-closure}\{q_0, q_1, q_2\} = \{q_0, q_1, q_2, q_3\} = B$$

A on input 1

$$\epsilon\text{-closure}\{q_1\} = \{q_1\} = C$$

B on 0

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

B on 1

$$\in\text{-closure}\{q_1\} = C$$

C on 0

$$\in\text{-closure}\{q_1, q_2\} = \{q_1, q_2, q_3\} = D$$

C on 1

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

D on 0

$$\in\text{-closure}\{q_1, q_2\} = D$$

D on 1

$$\in\text{-closure}\{q_3\} = q_3 = E$$

E on 0

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

E on 1

$$\in\text{-closure}\{q_3\} = E$$

Equivalent DFA is

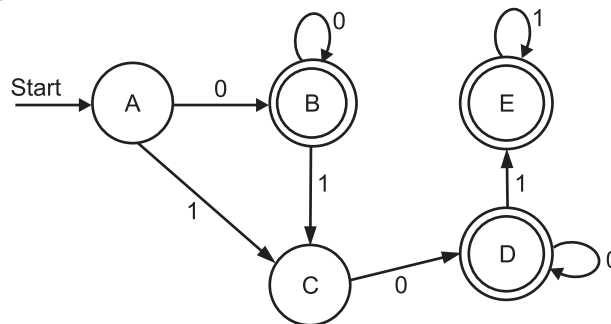


Fig. 1.93

Example 3: Construct Mealy machine over $\{0, 1\}$ which toggles its input.

Solution:

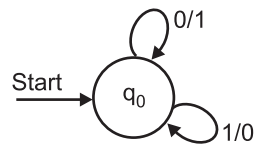


Fig. 1.94

Table 1.71

δ	0	1	
q_0	1	0	$\lambda(q_0, 0) = 1$ $\lambda(q_0, 1) = 0$

Example 4: Construct NFA without ϵ for language L , where $L = (0 + 1)^* 01$.

Solution:

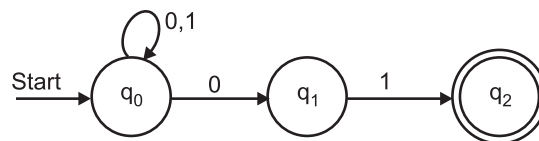


Fig. 1.95

Table 1.72

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

Example 5: Construct DFA equivalent to given NFA.

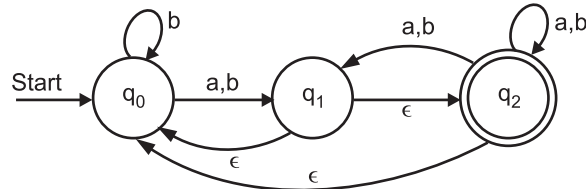


Fig. 1.96

Solution: ϵ -closure (q_0) = $q_0 \rightarrow$ Initial state of DFA (say A)

From A on input a i.e. q_0 on a is

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

From A on input b is

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

B on input a

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

B on input b is

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

No more states are added.

\therefore DFA is

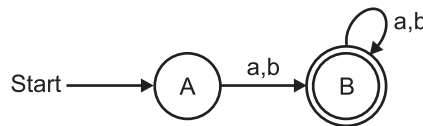


Fig. 1.97

Example 6: Construct Moore machine to generate 1's complement of binary number.

Solution: $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

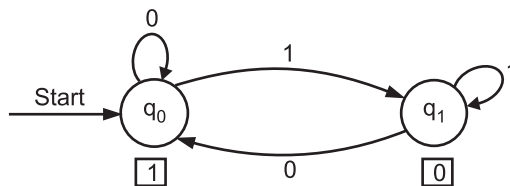


Fig. 1.98

Example 7: Construct FA for following regular expression

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

Solution: Let $r_1 = ab(a+b)^*$ and $r_2 = ba(a+b)^*$

FA for r_1 is M_1 .

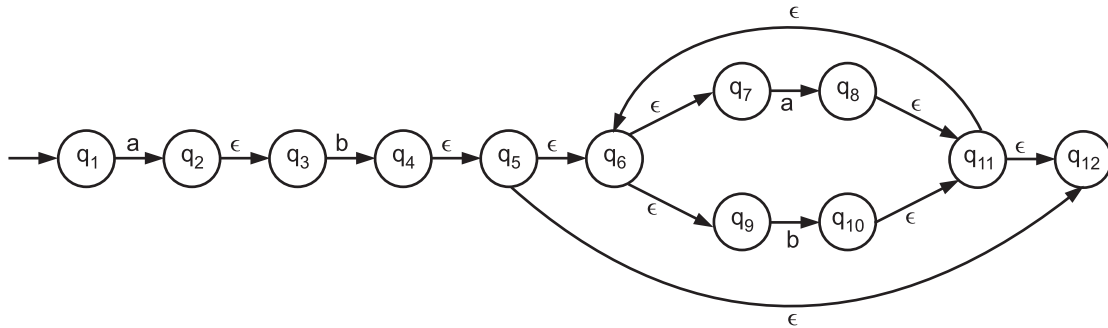


Fig. 1.99 (a): FA M_1 for r_1

FA M_2 for r_2 is

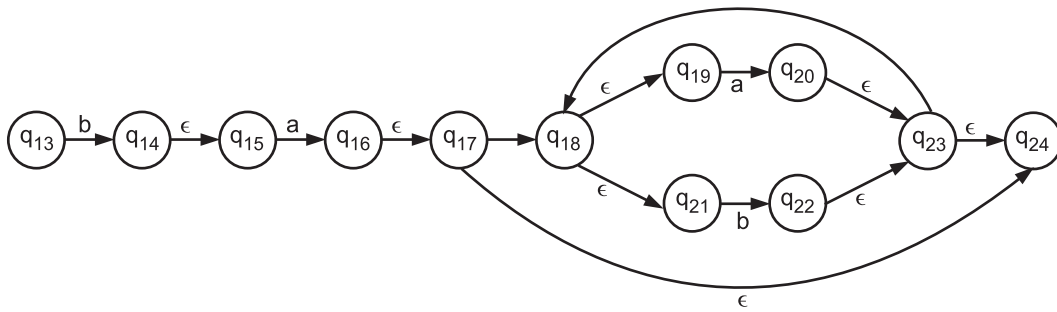


Fig. 1.99 (b)

\therefore FA M is

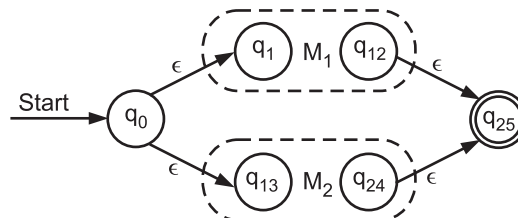


Fig. 1.99 (c)

Example 8: Construct NFA for language L where $L = 00(0+1)^*1$.

Solution:

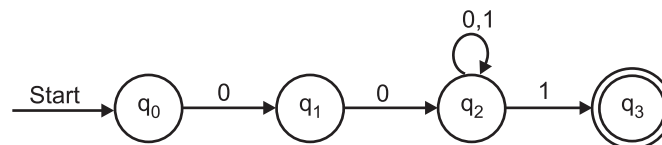


Fig. 1.100

Example 9: Construct Moore machine which outputs even or odd according to number of a's encountered is even or odd.

Solution:

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

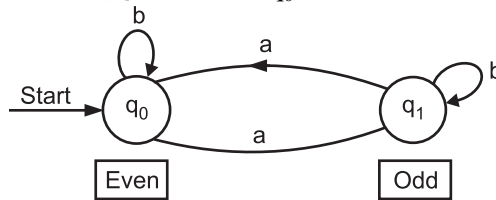


Fig. 1.101

Table 1.73

δ	a	b	λ
q ₀	q ₁	q ₀	Even
q ₁	q ₀	q ₁	Odd

Example 10: Construct DFA for accepting string over {a, b} such that it starts with a and not having substring "bac" in it.

Solution:

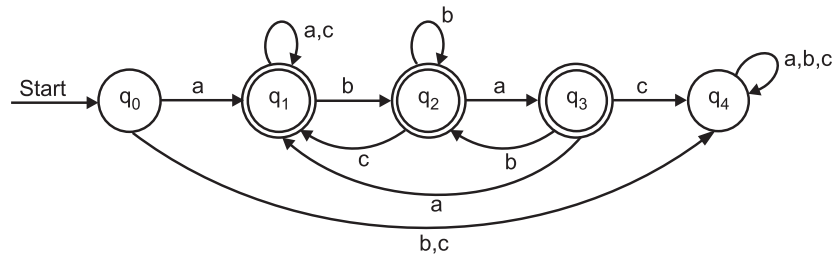


Fig. 1.102

Example 11: Construct DFA equivalent to NFA

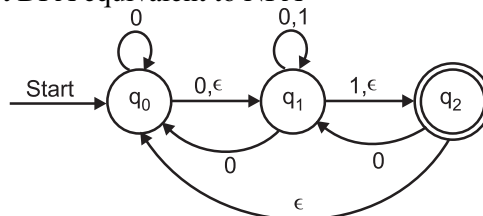


Fig. 1.103

Solution: ϵ -closure (q_0) = { q_0, q_1, q_2 } = Initial state = A

From A on input

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

From A on input 1

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

No more states are added. Therefore, DFA is

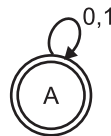


Fig. 1.104

Example 12: Construct NFA for language $L = 01^*(01)^* + 1$.

Solution:

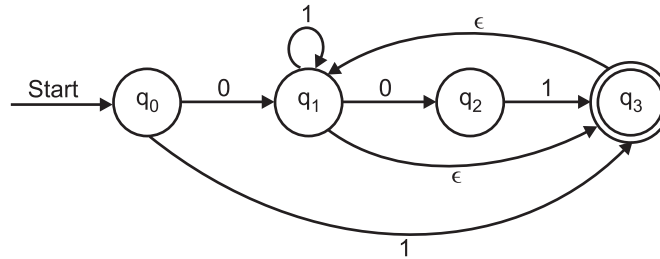


Fig. 1.105

Example 13: Construct Mealy machine to convert each occurrence of substring 101 by 100 over alphabet $\{0, 1\}$.

Solution:

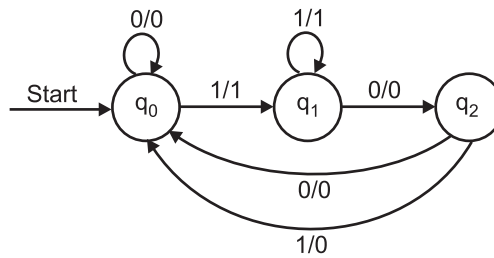


Fig. 1.106

Table 1.74

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_0

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

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

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

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

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

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

Example 14: Construct NFA for $ab^* + ba^*$.

Solution:

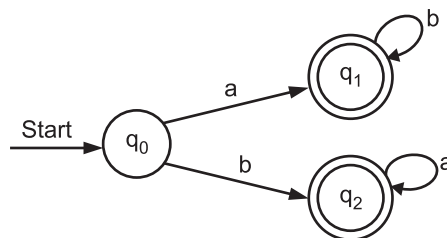


Fig. 1.107

Example 15: Construct DFA for a language over $\{0, 1, 2\}$ which starts with “00”, ends with “22” and having substring “11” in it.

Solution:

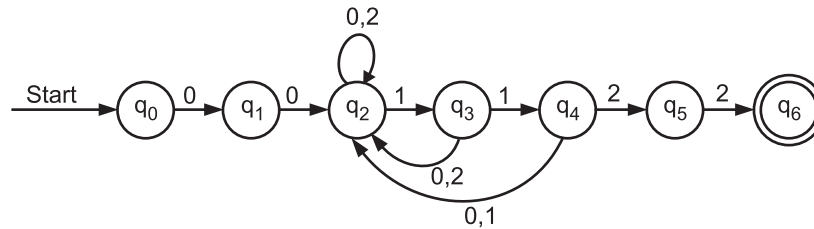


Fig. 1.108

Example 16: Minimize the following DFA.

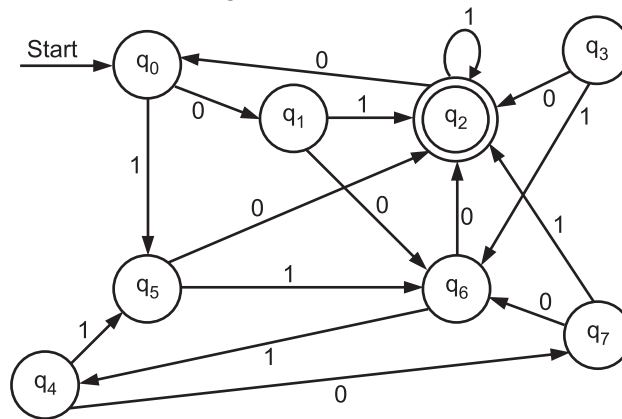


Fig. 1.109

Table 1.75

Solution: Step 1:

q_0	—							
q_1		—						
q_2	×	×	—					
q_3			×	—				
q_4			×		—			
q_5			×			—		
q_6			×				—	
q_7			×					—
	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7

$$F = \{q_2\}$$

So put mark X at entries q₂ Vs all non-final states.

Step 2: Find out distance D_1 in between remaining states.

$$\left. \begin{array}{ll} \delta(q_1, 0) = q_6 & \delta(q_1, 1) = q_2 \in F \\ \delta(q_0, 0) = q_1 & \delta(q_0, 1) = q_5 \notin F \end{array} \right\} \text{distinguishable}$$

Put mark X at entry (q_0, q_1) .

Similarly, we find out distance for all remaining states. We get states q_0 and q_4 equivalent, q_1 and q_7 are equivalent, q_3 and q_5 are equivalent.

\therefore Minimal DFA is

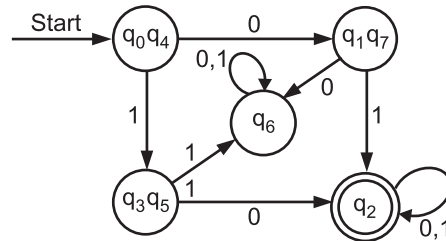


Fig. 1.110

Example 17: Construct a Mealy machine equivalent to the following Moore machine.

Table 1.76

States	0	1	Output
q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

Solution:

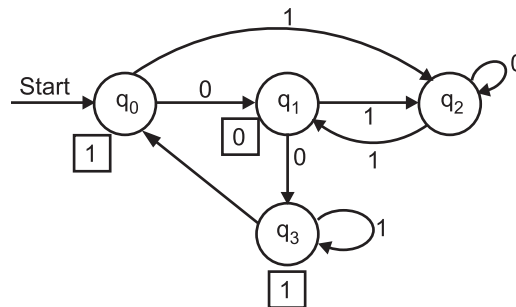


Fig. 1.111: Moore machine

The equivalent Moore machine is:

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

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

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

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

\therefore Equivalent Moore machine is:

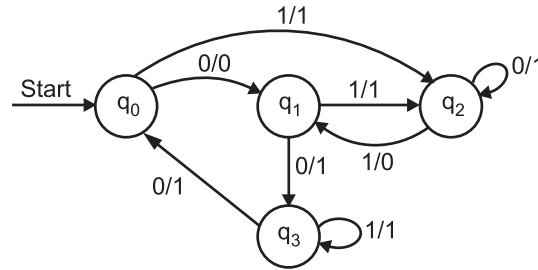


Fig. 1.112

Example 18: Construct FA for $L = \{\epsilon\}$.

Solution:

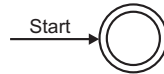


Fig. 1.113

Example 19: Define Moore machine. Design a Moore machine to change all vowels to '\$' and rest of the 21 alphabets changes to '#'.
Solution:

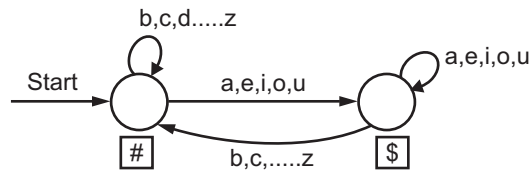


Fig. 1.114

Example 20: Construct a DFA which accepts odd number of 1's and even number of 0's over $\{0, 1\}$.

Solution:

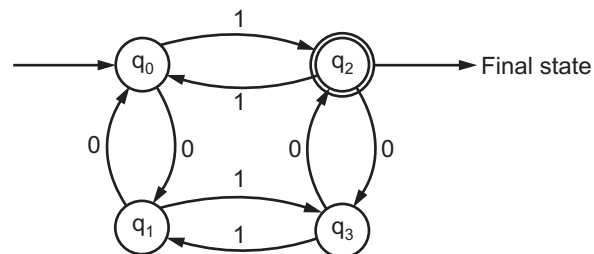


Fig. 1.115

Example 21: Construct FA for regular expression: $(ab)^* = (a + b)^* a * b$.

Solution:

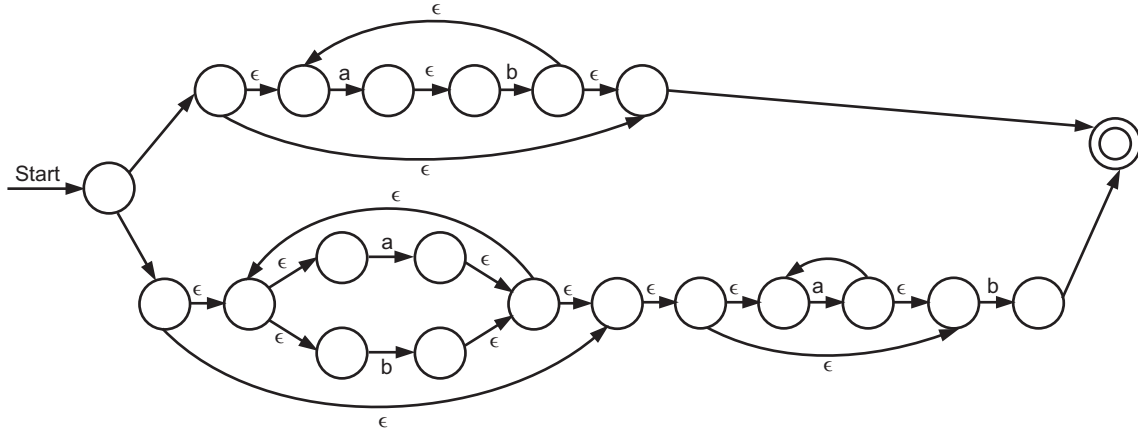


Fig. 1.116

Example 22: Design Mealy machine to determine the residue (remainder) mod 3 for a decimal number.

Solution:

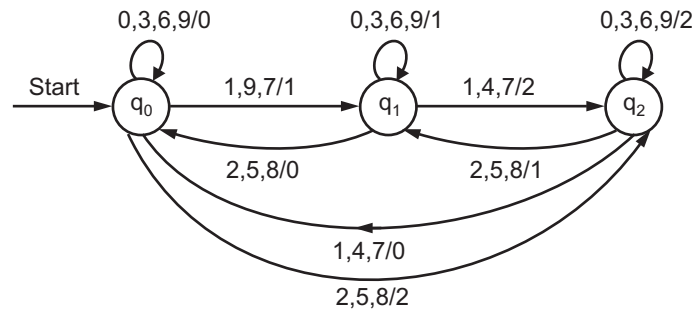


Fig. 1.117

Example 23: Construct DFA for the following NFA with ϵ -moves.

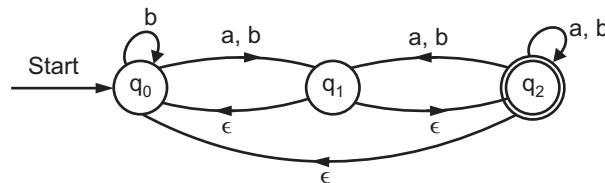


Fig. 1.118

Solution:

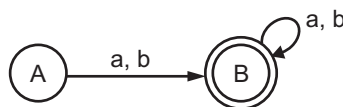


Fig. 1.119

PRACTICE QUESTIONS**Q.I Multiple Choice Questions:**

1. Which uses mathematical and logical methods to understand the nature of computation and to solve fundamental problems arising through the everyday practical use of computer systems?
(a) Computer Science (CS) (b) Theoretical Computer Science (TCS)
(c) Science Theory (ST) (d) Computer Science Theory (CST)
2. The word 'automata' (plural and the singular is 'automaton'), a _____ provides the simplest model of a computing device.
(a) Finite Automaton (FA) (b) Finite Grammar (FG)
(c) Finite Language (GL) (d) None of the mentioned
3. Which is a finite, non empty set of symbols?
(a) a language (b) a grammar
(c) an alphabet (d) All of the mentioned
4. Which is an entity or individual objects, which can be any letter, alphabet or any picture like 1, x, y, #?
(a) symbols (b) alphabets
(c) strings (d) All of the mentioned
5. Which is a finite collection of symbols from the alphabet.
(a) symbol (b) alphabet
(c) string (d) language
6. Which is a subset of Σ^* for some alphabet Σ ?
(a) symbol (b) alphabet
(c) string (d) language
7. Which is a set of rules to define a valid sentence in any language?
(a) grammar (b) language
(c) symbol (d) string
8. Noam Chomsky classified (Chomsky hierarchy) the grammar into following types depending on the production rules:
(a) **Type 0:** Type 0 grammar is a phase structure grammar without any restriction. All grammars are type 0 grammar. For example, Turing machine.
(b) **Type 1:** Type 1 grammar is called context-sensitive grammar. For example, Linear Bounded Automata (LBA).
(c) **Type 2:** Type 2 grammar is called context-free grammar. In the LHS of the production, there will no left or right context. For example, Push down automata (PDA).
(d) **Type 3:** Type 3 grammar is called regular grammar. For example, Finite Automata (FA).
(e) All of the mentioned

9. Which is generated from the rules of a grammar?
 - (a) a language
 - (b) a symbol
 - (c) a grammar
 - (d) an string
10. Finite automata M is represented as, $M = \{Q, \Sigma, \delta, q_0, F\}$.
 - (a) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
 - (b) $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \varnothing, \$, F)$
 - (c) $M = \{Q, \Sigma, \delta, q_0, F\}$
 - (d) None of the mentioned
11. In which FA, there is one and only one move from a given state to the next state of any input symbol.
 - (a) NFA
 - (b) DFA
 - (c) MFA
 - (d) None of the mentioned
12. In which FA, there may be more than one move or no move from a given state to the next state of any input symbol.
 - (a) NFA
 - (b) DFA
 - (c) MFA
 - (d) None of the mentioned
13. The construction of finite automata with a minimum number of states, which is equivalent to the given Finite Automata (FA) is called as,
 - (a) Maximization of FA
 - (b) Context-freeing of FA
 - (c) Minimization of FA
 - (d) None of the mentioned
14. The _____ machine is one type of finite automata where output depends on the present state only, but the output is independent of the present input.
 - (a) Moore
 - (b) Mealy
 - (c) transition machine
 - (d) None of the mentioned
15. The _____ machine is one type of finite automata with output, where the output depends on the present state and the present input.
 - (a) Moore
 - (b) Mealy
 - (c) transition machine
 - (d) None of the mentioned
16. A transition diagram also called as,
 - (a) transition graph
 - (b) transition system
 - (c) Both (a) and (b)
 - (d) None of the mentioned
17. Consider the regular expression $(0 + 1)(0 + 1) \dots N$ times. The minimum state FA that recognizes the language represented by this regular expression contains,
 - (a) $(n + 1)$ states
 - (b) $(n + 2)$ states
 - (c) n states
 - (d) None of the mentioned
18. Which is the string with zero occurrences of symbols?
 - (a) full
 - (b) finite
 - (c) null
 - (d) symbolic

Answers

1. (b)	2. (a)	3. (c)	4. (a)	5. (c)	6. (d)	7. (a)	8. (e)	9. (a)	10. (c)
11. (b)	12. (a)	13. (c)	14. (a)	15. (b)	16. (c)	17. (a)	18. (c)		

Q.II Fill in the Blanks:

1. The term "Automata" is derived from the Greek word "automatos" which means "_____".
2. An _____ (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.
3. The _____ is the smallest building block in the theory of computation and can be any letter, number or even picture.
4. _____ (Σ) are set of symbols, which are always finite.
5. FA is called finite because it has a finite number of _____.
6. _____ is an automaton that computes a Boolean function is called an acceptor.
7. _____ is a finite sequence of symbols from some alphabet denoted by ω .
8. _____ of a string is the number of symbols present in a string.
9. A _____ is a set of strings, chosen from some Σ^* or we can say, a language is a subset of Σ^* .
10. The finite automata are called _____ when there exist many paths for specific input from the current state to the next state.
11. _____ string is the string with zero occurrence of symbols, represented as ϵ .
12. Non-terminal symbols are those symbols which can be replaced multiple times and _____ symbols are those symbols which cannot be replaced further.
13. An automaton with a finite number of states is called a _____ or Finite State Machine (FSM).
14. If any FA contains ϵ transaction or move, the finite automata is called _____.
15. The production rules of a grammar consist of two parts namely, _____ [(LHS) of a production rule mainly contains the non-terminal symbols to be replaced and may contain terminal and non-terminal both for some cases but at least one non-terminal] and _____ [(RHS) of a production rule may contain terminal, non-terminal or any combination of terminal and non-terminal even null].
16. The finite automata are called _____ finite automata if the machine is read an input string one symbol at a time.
17. The _____ machine was proposed by George H. Mealy at the Bell Labs in 1960.
18. John Myhill and Anil Nerode of the University of Chicago proposed a theorem in 1958 which provides a necessary and sufficient condition for a language to be regular and theorem can also be used to _____ a FA.
19. A _____ M is a 5-tuple, $M_{DFA} = (Q, \Sigma, \delta, q_0, F)$.
20. A transition diagram (also called transition graph or transition system) is a finite directed labeled graph in which each _____ represents a state and directed edges indicate the transition from one state to another.
21. A _____ diagram or state transition diagram is a directed graph for FA.
22. In a Mealy machine, the output depends on the _____ present state and the present input.

23. The transition _____ is basically a tabular representation of the transition function which takes two arguments (a state and a symbol) and returns a state (the "next state").
24. The set of rules for constructing a language is called the _____ for that language.
25. The Moore machine, the output depends only on the present _____.
26. A _____ of a string is formed by taking any number of symbols from the end of the string.
27. A _____ language is a set of strings of symbols drawn from a finite alphabet.

Answers

1. self-acting	2. automaton	3. symbol	4. Alphabets
5. states	6. Recognizer	7. String	8. Length
9. language	10. NFA	11. Empty	12. Terminal
13. Finite Automaton (FA)	14. NFA with ϵ move	15. Left Hand Side and Right Hand Side	16. deterministic
17. Mealy	18. minimization	19. DFA	20. vertex
21. transition	22. present	23. table	24. grammar
25. state	26. suffix	27. formal	

Q.III State True or False:

1. The word "Automata" is the plural form of the word "Automaton" means an object that is capable of doing something on its own or itself.
2. An automaton is a system where materials, energy, or information are transformed and transmitted for performing some operation without the direct participation of a human.
3. A language which can be formed over ' Σ ' is any subset Σ^* and can be Finite or Infinite.
4. Any non empty finite set is called as an alphabet (Σ).
5. In NFA, there is a finite set of states, a finite set of input symbols, and a finite set of transitions from one state to another state that occur on input symbol chosen to form an alphabet; there is exactly one transition out of each state.
6. NFA can be converted to an equivalent DFA.
7. In FA input tape (divided into squares/cells) contains a string of symbols, with one symbol in each tape square. The finite, control is the main part of the machine whose internal status can be specified as one of a finite number of distinct states. Using a movable reading head, the finite control can sense the symbol written at any position on the input tape.
8. A language is not complete without grammar.
9. A finite automaton has a finite set of states with which it accepts or rejects strings.
10. DFA can have any number of transitions to the next state from a given state on a given input symbol.

11. The Moore machine was proposed by Edward F. Moore in IBM around 1960.
12. By minimizing, we can get a minimized DFA with minimum number of states and transitions which produces that particular language.
13. A DFA with minimized states needs less time to manipulate a regular expression.
14. The Myhill–Nerode theorem is used to minimize finite automata.
15. A transition graph or a transition system is a finite directed labeled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labeled with input/output.
16. An FA has a infinite number of states.
17. A transition diagram or state transition diagram is a directed graph for FA.
18. NFA with ϵ can be converted to NFA without ϵ , and this NFA without ϵ can be converted to DFA.
19. Maximization of FA means reducing the number of states from given FA.
20. In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol.
21. In DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.
22. In Mealy Machine the output depends both on the current state and the current input and in Moore Machine the output depends only on the current state.
23. A finite automaton recognizes regular language.
24. A prefix of a string is the string formed by taking any number of symbols of the string.

Answers

1. (T)	2. (T)	3. (T)	4. (T)	5. (F)	6. (T)	7. (T)	8. (T)	9. (T)	10. (T)
11. (F)	12. (T)	13. (T)	14. (T)	15. (T)	16. (F)	17. (T)	18. (T)	19. (F)	20. (T)
21. (T)	22. (T)	23. (T)	24. (T)						

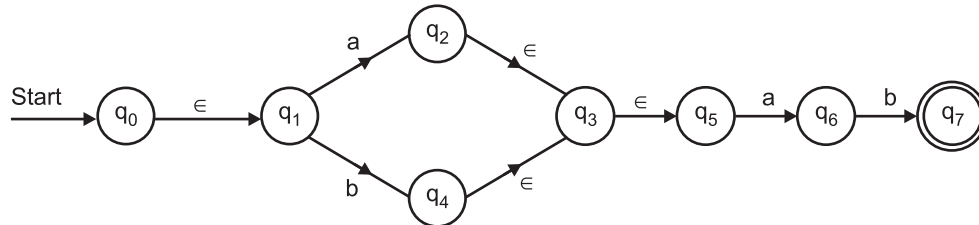
Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. What is symbol?
2. Define alphabet.
3. Define FA.
4. What is prefix and suffix?
5. Define formal language.
6. List operations on languages.
7. Define grammar.
8. What are DFA and NFA?
9. Define minimization of FA.
10. Give the purpose of Mealy and Moore machines.
11. Compare DFA and NFA (any two points).

(B) Long Answer Questions:

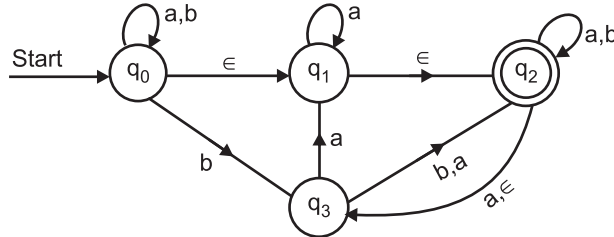
1. What is FA? Enlist its types.
2. What is the necessity of converting an NFA to a DFA?
3. What is null string and length of string? Explain with example.
4. Write a short note on: NFA with ϵ -Transitions to DFA.
5. For the following NFA, find an equivalent DFA.



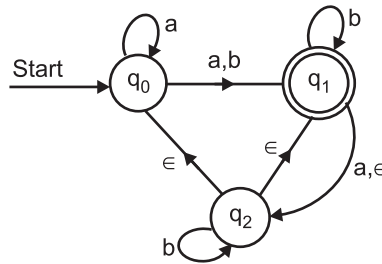
6. Describe DFA as pattern recognizer in detail.
7. With the help of example describe Myhill-Nerode method for NFA to DFA.
8. Describe NFA with ϵ - transitions with example.
9. Construct a DFA to accept the set of all strings over $\{0, 1\}$ such that every pair of adjacent 0's appears before any pair of adjacent 1's.
10. Construct DFA for language over alphabet $\{a, b\}$ which accepts all strings containing 'a' at every even position in the string.
11. Construct NFA for language over alphabet $\{a, b\}$ which accepts all strings starting with 'a' and not having substring 'bbb' in it.
12. Construct DFA for $L = L_1 \cap L_2$, where
 L_1 = all strings starting with 'a' containing even number of b's over $\{a, b\}$.
 L_2 = all strings containing odd number of b's over $\{a, b\}$.
13. Construct DFA to accept following languages:
 - (i) Set of all strings which contain exactly two c's anywhere over $\{a, b, c\}$.
 - (ii) Set of strings over $\{0, 1\}$ whose second and second last symbol is 1.
 - (iii) $\{x \in \{0, 1\}^* \mid x \text{ has neither } 00 \text{ nor } 11 \text{ as substring}\}$.
 - (iv) All strings over $\{0, 1\}$ in which every even position in string is occupied by 0 and odd position by 1.
 - (v) Which accepts all strings starting with 01 and having substring 012 in it.
 - (vi) All strings starting with '10' and having substring '201' in it.
 - (vii) All strings over $\{0, 1\}$ such that if it starts with 0 then it contains even number of 0's and if it starts with 1 then it contains odd number of 1's.
 - (viii) Language over alphabet $\{0, 1, 2\}$ which accepts all strings containing at least one occurrence of double symbol.

14. Construct DFA for the following NFA with ϵ -transitions:

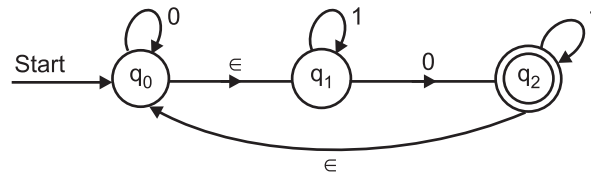
(i)



(ii)



(iii)

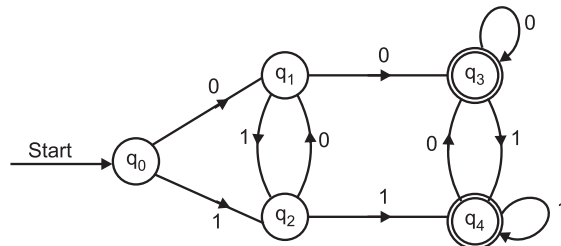


15. (i) Construct minimal DFA for the following:

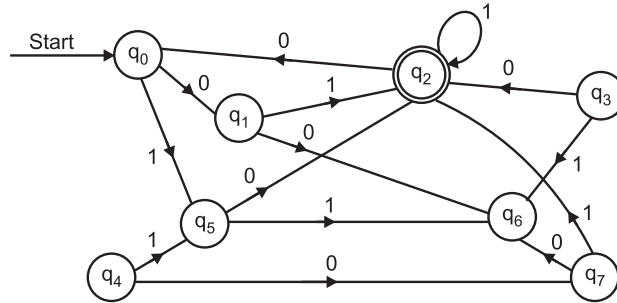
$M = (\{A, B, C, D, E, F, G, H\}, \{0, 1\}, \delta, A, \{C\})$, where δ is

δ	0	1
A	B	F
B	G	C
C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

(ii)



(iii)



(iv)

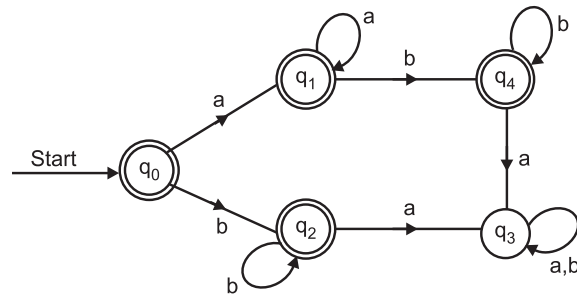
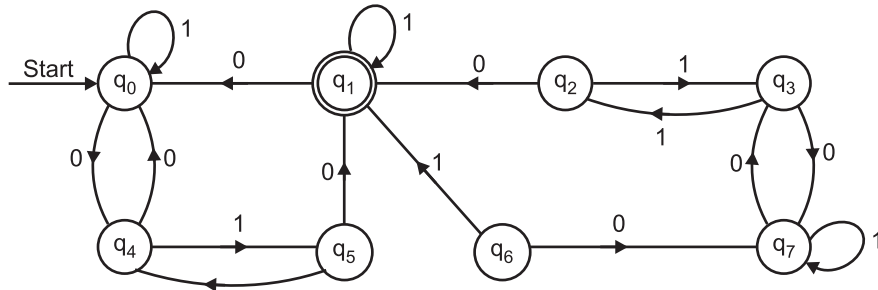
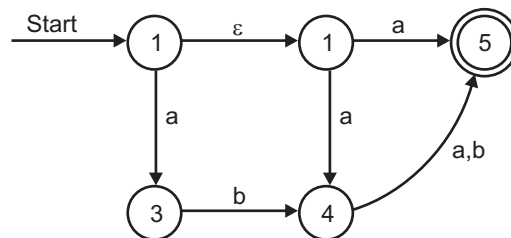


Fig. 1.87

(v)

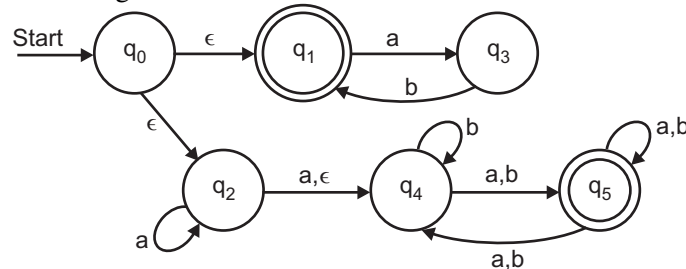


(vi)

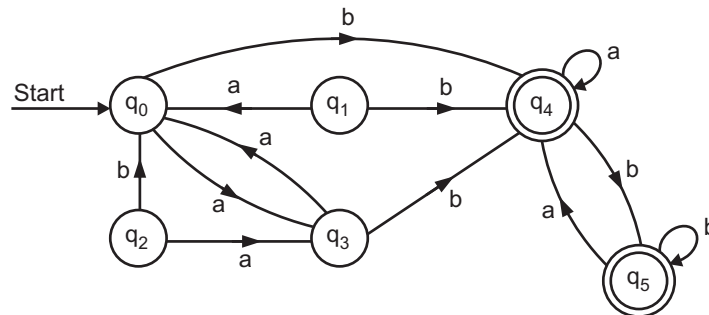


16. Design a Mealy machine for the following: for input from $(a + b + c)^*$ if input ends in "bac" then print A, else print B.
17. Design a Mealy machine to get 1's complement of a given binary string.
18. Design a Moore machine to get 1's complement of a given binary string.

19. Design a Moore machine for binary input sequence such that if it has a substring 101, the machine outputs A, if it has a substring 110, the machine outputs B, otherwise it outputs C.
20. Design a Mealy machine which outputs EVEN or ODD according to number of 1's encountered is even or odd over $\{0, 1\}$.
21. Write a mapping of δ in case of NFA and DFA.
22. Differentiate between Moore and Mealy machine.
23. Convert the following NFA with ϵ -moves to DFA.



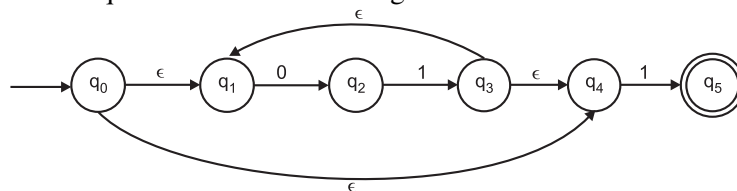
24. Find minimum state FA equivalent to the following DFA, $M = (\{q_0, \dots, q_5\}, \{a, b\}, \delta, q_0, \{q_4, q_5\})$.



UNIVERSITY QUESTIONS AND ANSWERS

April 2016

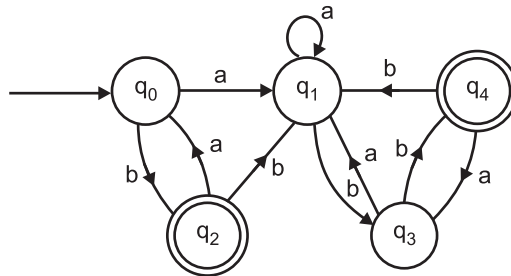
1. What are the proper prefixes and proper suffixes of the string "India"? [1 M]
- Ans. Refer to Page 1.2.
2. Compare 'λ' function of Melay and Moore machine. [1 M]
- Ans. Refer to Section 1.8.
3. Construct a DFA to accept all decimal numbers divisible by 3. [5 M]
- Ans. Refer to Section 1.3.
4. Construct DFA equivalent to the following NFA: [5 M]



Ans. Refer to Section 1.5.

5. Minimize the following DFA:

[4 M]



Ans. Refer to Section 1.9.

October 2016

1. Define NFA.

[1 M]

Ans. Refer to Section 1.4.

2. Define proper suffix with the help of an example.

[1 M]

Ans. Refer to Page 1.2.

3. Write the mapping of λ function in Mealy Machine.

[1 M]

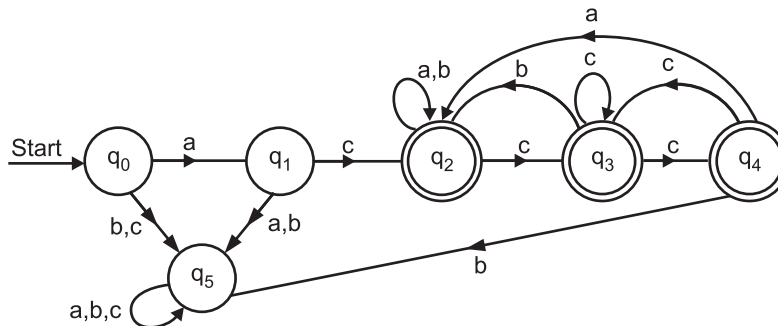
Ans. Refer to Section 1.8.

4. Construct a DFA to accept the set of all strings over $\Sigma = \{a, b, c\}$ such that the string starts with 'ac' and not having 'cab' as substring in it.

[5 M]

Ans. $L = \{ac, aca, acb, acc, \dots\}$

The TD is as follows.



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

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{a, b, c\}$$

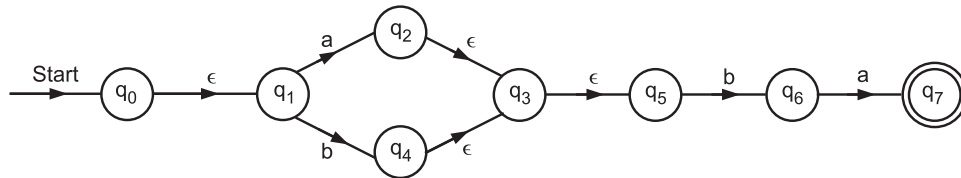
$$q_0 = \{q_0\}$$

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

$\delta Q \backslash \Sigma$	a	b	c
q_0	q_1	q_5	q_5
q_1	q_5	q_5	q_2
q_2	q_2	q_2	q_3
q_3	q_4	q_2	q_3
q_4	q_2	q_5	q_3
q_5	q_5	q_5	q_5

5. Convert the following NFA with ϵ moves to DFA.

[5 M]



Ans. Initial state is q_0 , we take

ϵ -closure (q_0) = { q_0, q_1 } – A – initial state of DFA

1. From A, on input 'a', we get q_2 , so

ϵ -closure (q_2) = { q_2, q_3, q_5 } = B

From A, on input 'b' we get q_4 , so

ϵ -closure (q_4) = { q_4, q_3, q_5 } = C

2. From B, on input 'a' we get only state q_6

ϵ -closure (q_6) = { q_6 } = D

From B, on input 'b', we get only set ϕ .

3. From C, on input 'a' we get state q_6

ϵ -closure (q_6) = { q_6 } = D

From c, on input 's', we get set ϕ .

4. From D, on input 'a' we get only set { ϕ }

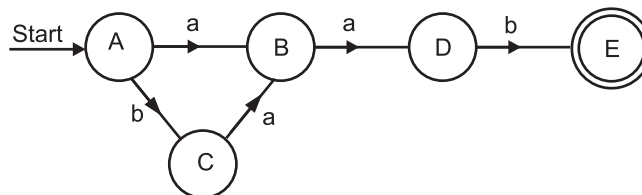
From D, on input 'b' we get only state q_7

ϵ -closure (q_7) = { q_7 } = E

5. From E on input 'ab', we get set { ϕ }

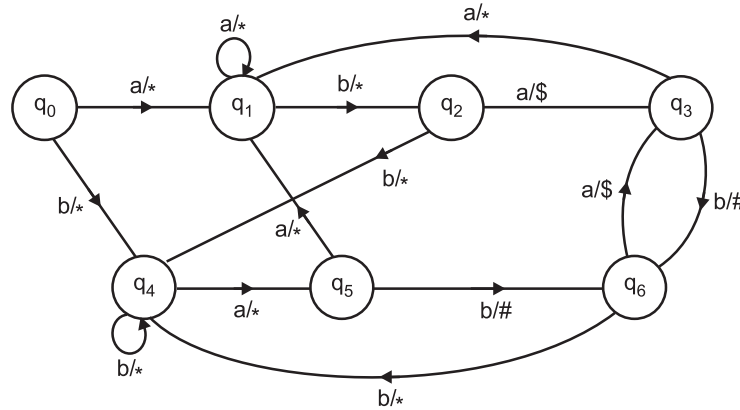
Also from E on input 'b' we get set { ϕ }

Equivalent DFA



6. Construct a Mealy machine of a language L over $\Sigma = \{0, 1\}$ which outputs '\$' if string ends with 'aba', outputs '#' if string ends with 'bab', otherwise outputs '*'. [5 M]

Ans. Mealy Machine



δ $Q \backslash \Sigma$	a	b	ϕ $Q \backslash \epsilon$	a	b
q_0	q_1	q_4	q_0	*	*
q_1	q_1	q_2	q_1	*	*
q_2	q_3	q_4	q_2	\$	*
q_3	q_1	q_6	q_3	*	#
q_4	q_5	q_4	q_4	*	*
q_5	q_1	q_6	q_5	*	#
q_6	q_3	q_4	q_6	\$	*

7. Differentiate between DFA and NFA.

[5M]

Ans. Refer to Section 1.21.

April 2017

1. Give the mapping of ' δ ' function of NFA with ϵ moves.

[1 M]

Ans. Refer to Section 1.6.

2. If $A = \{\epsilon\}$. Find the value of $|A|$.

[1 M]

Ans. Value of $A = 1$.

3. Differentiate between Moore and Mealy machine.

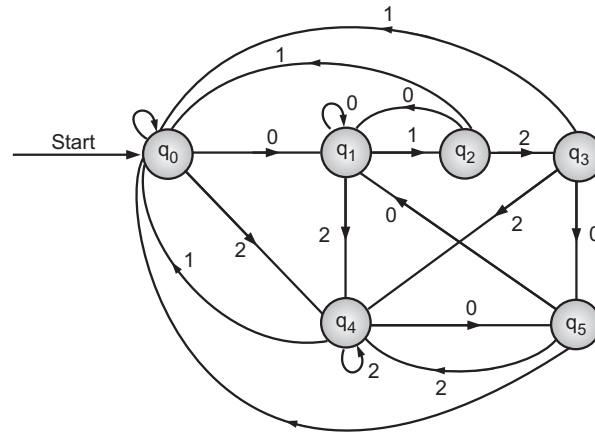
[1 M]

Ans. Refer to Page 1.43.

4. Construct a DFA to accept the set of all strings over $\Sigma = \{0, 1, 2\}$ such that the string ends with '012' or '20'.

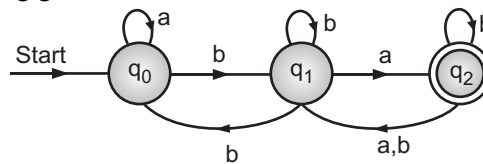
[5 M]

Ans. $L = \{012, 20, 0012, 020 \dots\}$



5. Convert the following given NFA to DFA:

[5 M]



Ans. Refer to Section 1.5.

6. Construct a Moore machine for a language L over $\Sigma = \{0, 1\}$ which outputs '\$' if string ends with '100', outputs '#' if string ends with '001', otherwise outputs '*'. [5 M]

Ans. Refer to Section 1.8.

7. Minimize the following DFA:

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{0, 1\}, \delta, q_0, \{q_1\})$ where, δ is given by:

[5 M]

δ	a	b
$\rightarrow q_0$	q_4	q_0
$*q_1$	q_1	q_0
q_2	q_1	q_3
q_3	q_7	q_2
q_4	q_0	q_5
q_5	q_1	q_4
q_6	q_7	q_1
q_7	q_3	q_7

Ans. Refer to Section 1.9.

October 2017

1. Define suffix of a string. Give one example.

[1 M]

Ans. Refer to Page 1.2.

2. "DFA cannot have more than one final states". Justify.

[1 M]

Ans. Refer to 1.2.

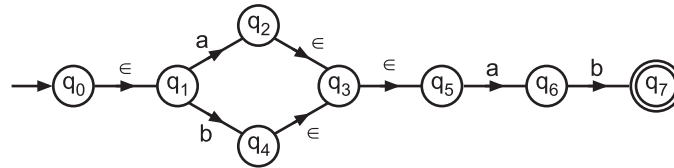
3. Write output function λ of Moore and Mealy machines. [1 M]

Ans. Refer to Section 1.8.

4. Construct DFA to accept substrings having both aa and bb over $\Sigma = \{a, b\}$. [5 M]

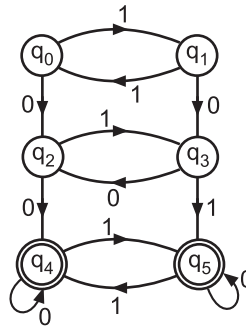
Ans. Refer to Section 1.3.

5. Convert the following NFA to DFA: [5 M]



Ans. Refer to Section 1.5.

6. Minimize the following DFA using Myhill-Nerode theorem: [5 M]



Ans. Refer to Section 1.9.

7. Construct Moore and Mealy machines which outputs valid for valid strings and invalid for invalid strings for language $L = a(a + b)^* b$. [4 M]

Ans. Refer to Section 1.8.

April 2018

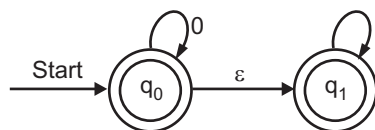
1. What are the proper prefix and proper suffix of the string "India"? [1 M]

Ans. Refer to Page 1.2.

2. Define DFA. [1 M]

Ans. Refer to Section 1.3.

3. Write down the ϵ -closure of each state from the following FA. [1 M]

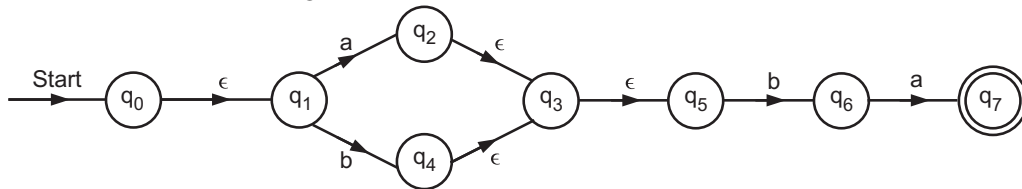


Ans. Refer to Section 1.6.

4. Construct DFA containing all string starting with 01 and having 012 as substring. [5 M]

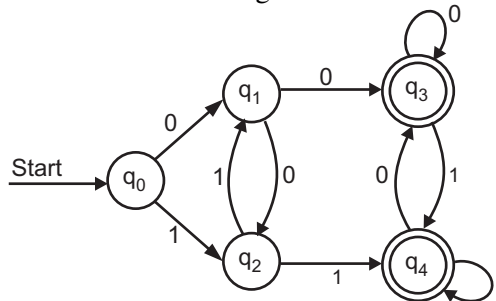
Ans. Refer to Section 1.3.

5. Construct the following NFA with ϵ -moves to DFA: [5 M]



Ans. Refer to Section 1.6.

6. Construct minimal DFA for the following: [2 M]



Ans. Refer to Section 1.9.

7. Construct Mealy machine to convert each occurrence of substring 101 by 100 over alphabet $\{0, 1\}$. [4 M]

Ans. Refer to Section 1.8.

October 2018

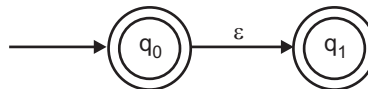
1. Define suffix of a string. Give one example. [1 M]

Ans. Refer to Page 1.2.

2. Compare ' λ ' function of Melay and Moore machine. [1 M]

Ans. Refer to Section 1.8.

3. Write down the ' ϵ -closure' of each state from the following FA: [3 M]

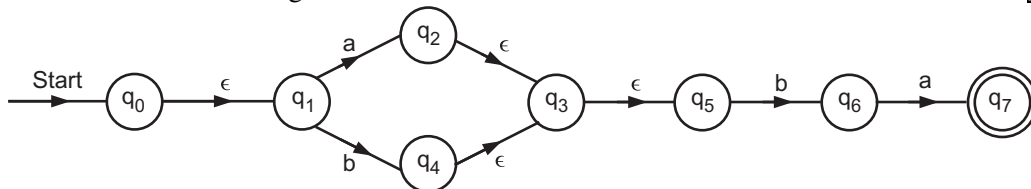


Ans. Refer to Section 1.6.

4. Construct DFA for $L = L_1 \cap L_2$ where
 L_1 = all strings starting with 'b' over $\{a, b\}$
 L_2 = all strings not having 'ba' as substring over $\{a, b\}$. [5 M]

Ans. Refer to Section 1.3.

5. Convert the following NFA to DFA: [5 M]



Ans. Refer to Section 1.5.

6. Construct the minimize DFA for the following DFA: [4 M]

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

	δ	a	b
Start \rightarrow	q_0	q_1	q_2
	q_1	q_1	q_3
Final	$\leftarrow q_2$	q_0	q_1
	q_3	q_1	q_4
Final	$\leftarrow q_4$	q_3	q_1

(Use Myhill-Nerode Theorem)

Ans. Refer to Section 1.9.

7. Construct Moore machines for binary input sequence such that if it has a substring 101, the machine output 'A', if it has substring 110 then output 'B' else output 'C'. [4 M]

Ans. Refer to Section 1.8.

8. DFA may have many final states. Comment. [1 M]

Ans. Refer to Section 1.3.

April 2019

1. What are the proper prefix and proper suffix of the string "Computer"? [1 M]

Ans. Refer to Page 1.2.

2. Write down the ϵ -closure of each state from the following FA: [1 M]



Ans. Refer to Section 1.6.

3. Finite Automata has more than one Final states (True or False) Justify. [1 M]

Ans. Refer to Section 1.3.

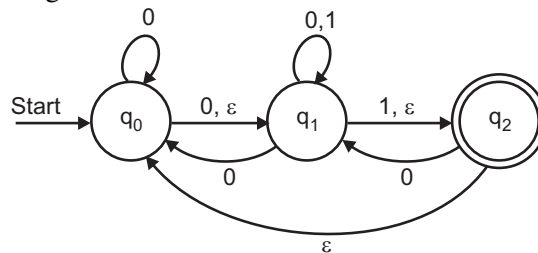
4. State two differences between NFA and DFA. [1 M]

Ans. Refer to Page 1.21.

5. Construct a DFA for a language $L = \{x \mid x \text{ has neither "aa" nor "bb" as a substring}\}$ over $\Sigma = \{a, b\}$. [5 M]

Ans. Refer to Section 1.2.

6. Convert the following NFA with ϵ moves to DFA. [5 M]



Ans. Refer to Section 1.6.

7. Construct minimal DFA for the following:

[5 M]

$M = (\{A, B, C, D, E\}, \{0, 1\}, \delta, A, \{E\})$

when δ is given by,

δ	0	1
\rightarrow A	B	C
B	B	D
C	B	C
D	B	E
* E	B	C

Ans. Refer to Section 1.9.

8. Construct a mealy machine to convert each occurrence of substring 101 by 100 over alphabet $\{0, 1\}$.

[4 M]

Ans. Refer to Section 1.8.



Regular Expressions and Languages

Objectives ...

- To study Basic Concepts in Regular Expressions
 - To learn Regular Languages
-

2.0 INTRODUCTION

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages. A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

2.1 REGULAR EXPRESSION (RE)

[April 16, 17, 18, 19, Oct. 17, 18]

- The languages accepted by finite automata are described or represented by simple expressions called regular expressions.
- The languages that are associated with these regular expressions are called regular and are also said to be defined by finite automata.
- Regular expressions are also referred as rational expressions. Regular expression is generally a sequence of characters that is used to find a string in language.

Operations of Sets of Strings:

- Regular expressions represent simple language-denoting formulas, based on the operations of concatenation, union and closure.
- Let, Σ be a finite set of symbols and let L_1 , L_2 and L be sets of strings from Σ^* . Then we can define following operations on these sets.

1. **Concatenation:** Concatenation of L_1 and L_2 is denoted by,

$$L_1 L_2 = \{xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2\}$$

2. **Kleene Closure:** It is denoted by L^* and is defined as,

[April 17, 19]

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

We define $L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$

3. **Positive Closure:** It is denoted by L^+ and is defined as,

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Example: Let, $L_1 = \{10, 1\}$ and $L_2 = \{011, 11\}$ then,

$$L_1 L_2 = \{10011, 1011, 111\}$$

$$L_1^* = \{\epsilon, 10, 1, 1010, 101, 110 \dots\}$$

$$L_1^+ = \{10, 1, 1010, 101, 110 \dots\}$$

- Regular expressions play an important role in computer science applications, which involves:
 - It is used by many text editors and utilities to search block of text for certain pattern.
 - It is also used to design the compilers for programming languages.

2.1.1 Formal Definition of Regular Expression

- Let Σ be an alphabet. The regular expression over Σ and the sets they denote are defined recursively as follows :
 - ϕ is a regular expression and denotes the empty set.
 - ϵ is a regular expression and denotes the set $\{\epsilon\}$.
 - For each a in Σ , a is a regular expression and denotes the set $\{a\}$.
 - If r_1 and r_2 are regular expressions denoting languages R_1 and R_2 respectively then $(r_1 + r_2)$, $(r_1 r_2)$ and r_1^* are regular expressions denoting the sets $R_1 \cup R_2$, $R_1 R_2$ and R_1^* respectively.
- If 'r' is a regular expression, then the language represented by 'r' is denoted by $L(r)$.

- If $r = a + b$
 $L(r) = \{a, b\}$
- If $r = ab$
 $L(r) = \{ab\}$
- If $r = a^*$
 $L(r) = \{\epsilon, a, aa, aaa, aaaa, \dots\}$

Here, ϵ stands for zero occurrences of 'a'. Hence, 'a' denotes an infinite set of strings.

- If $r = (ab)^* = \{\epsilon, ab, abab, ababab, \dots\}$
- If $r = a^* b^* = \{\epsilon, a, b, ab, aab, abb, aabb, \dots\}$
- If $r = (a + b)^* = \{\epsilon, a, b, ba, ab, baa, abb, \dots\}$

2.1.2 Examples

- In this section we will study some example of regular expressions.

Example 1: Define the language such that all words begin and end with 'a' and in between any word using 'b', using regular expression.

Solution: $\Sigma = \{a, b\}$, start symbol and end symbol = 'a'. The regular expression is $ab^*a + a$.

Example 2: Describe the language consisting of all strings over $\Sigma = \{0, 1\}$ with atleast two consecutive 0's, using regular expression.

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

Here, we want atleast one occurrence of "00" and any number of trailing 1's and 0's and any number of leading 1's and 0's.

Therefore, the regular expression is,

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

Example 3: If $L(r)$ = the set of all strings over $\{0, 1\}$ ending with "011", find regular expression r .

Solution: The regular expression is,

$$r = (0 + 1)^* 011$$

Example 4: Define language over $\{0, 1\}$ containing all possible combinations of 0's and 1's but not having two consecutive 0's.

Solution: $(1 + 10)^*$ is language containing the strings starting with 1 and not having two consecutive 0's. Similarly, the language containing the strings starting with '0' and not having two consecutive 0's is represented by $0 \cdot (1 + 10)^*$.

If we combine both, we get the required regular expression as

$$(1 + 10)^* + 0 (1 + 10)^* \text{ i.e. } (0 + \epsilon) (1 + 10)^*.$$

Example 5: Define language containing all strings of a's and b's containing atleast one combination of double letters using regular expression.

Solution: Regular expression, $r = (a + b)^* (aa + bb) (a + b)^*$.

Example 6: Define language containing all strings over $\{0, 1\}$ having atleast one pair of 0's or at most one pair of 1's.

Solution: Here we have four combinations

1. Consecutive 0's and 1's not present.
2. Only one pair of consecutive 0's present.
3. Only one pair of consecutive 1's present.
4. One pair of consecutive 0's and 1's present.

The regular expression is,

$$r = (01)^* 00 (10)^* + (10)^* 11 (01)^* + (01)^* + (10)^* + (10)^* 0 (10)^* + (01)^* 1 (01)^*$$

Example 7: If $r = b^*ab^*ab^*$, describe $L(r)$ in simple English.

Solution: $L(r)$ is the language over $\Sigma = \{a, b\}$ containing exactly two a's.

Example 8: Show that $(a^*b^*)^* = (a+b)^*$.

Solution: Let $r_1 = (a^*b^*)^*$ and $r_2 = (a+b)^*$

$$\begin{aligned} L(r_1) &= (\{\epsilon, a, aa, \dots\} \cup \{\epsilon, b, bb, \dots\}) \\ &= \{\epsilon, a, b, aa, bb, ab, ba, \dots\} \end{aligned} \quad \dots (1)$$

$$L(r_2) = \{\epsilon, a, b, aa, ab, bb, ba, \dots\} \quad \dots (2)$$

From (1) and (2), we get,

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

Example 9: Is a^*b^* is equal to $(ab)^*$?

Solution: No. a^*b^* is not the same as $(ab)^*$ because a^*b^* does not contain substring "ba" whereas $(ab)^*$ contains strings like abab, etc.

Example 10: Write a regular expression for a language containing set of all strings of a's and b's ending in bb.

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

Example 11: Write a regular expression for language containing string with any number of a's followed by any number of b's followed by any number of c's.

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

Example 12: Write a regular expression to denote a language over $\Sigma = \{a, b\}$ such that every string begin and end with either aa or bb.

Solution: $r = (aa+bb)(a+b)^*(aa+bb)+aa+bb$

Example 13: Write a regular expression for language contains atleast one a and atleast one b.

Solution: $r = [(a+b)^*a(a+b)^*b(a+b)^*] + [(a+b)^*b(a+b)^*a(a+b)^*]$

Example 14: Write a regular expression to denote a language containing string in which the 4th character from the right end of string is always a over $\Sigma = \{a, b\}$.

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

Example 15: Write a regular expression for language containing each string of even length over $\{0\}$.

Solution: $L = (\epsilon, 00, 0000, \dots)$

$$r = (00)^*$$

Example 16: Write a regular expression to denote a language having strings not containing substring “01” over $\Sigma = \{0, 1\}$.

Solution:
$$r = 1^* 0^*$$
$$L = \{\epsilon, 0, 1, 10, 100, 000, \dots\}$$

Example 17: Find regular expression for a language consists of strings over $\{a, b\}$ which are either all b’s or strings begin with a and followed with any number of b’s only or an empty string.

Solution:
$$L = \{\epsilon, b, bb, bbb, a, ab, abb, abbb, \dots\}$$
$$r = b^* + ab^*$$
$$= (\epsilon + a) b^*$$

Example 18: Find regular expression for a language consists of string over $\{0, 1\}$ whose 3rd digit from right end is always 1.

Solution:
$$r = (0 + 1)^* 1 (0 + 1) (0 + 1)$$

Example 19: Write regular expression for a language consists of string having length divisible by 3 over $\{a\}$.

Solution:
$$r = (aaa)^*$$

Example 20: Write regular expression for a language containing strings do not have either “aa” or “bb” as a substring in it over $\{a, b\}$.

Solution:
$$L = \{\epsilon, a, b, ab, bab, aba, \dots\}$$
$$r = (b + \epsilon) (ab)^* (a + \epsilon)$$

Example 21: Write regular expression for a language consisting of string such that total number of b’s in each string is divisible by 3 over $\{a, b\}$.

Solution:
$$L = \{\epsilon, a, aa, abbb, bbba, ababab, \dots\}$$
$$r = (a^* b a^* b a^* b a^*)^* + a^*$$

Example 22: Write a regular expression for a language of strings with total number of 0’s are even over $\{0, 1\}$.

Solution:
$$r = 1^* + (1^* 0 1^* 0 1^*)^*$$

Example 23: Write regular expression to denote a language L over $\{a, b\}$ such that all the strings do not have a substring “ab”.

Solution:
$$r = b^* a^* (L = \{\epsilon, a, b, bb, ba, baa, \dots\})$$

Example 24: Write regular expression for language containing strings without a substring “abb” and “bba”.

Solution:
$$r = b^* + (b + \epsilon) (a + ab)^*$$
$$\therefore L = \{\epsilon, a, b, ab, ba, abab, aab, bb, \dots\}$$

Example 25: Write regular expression for language containing strings in which every block of 4 consecutive symbols contain atleast two a's over {a, b}.

Solution: For block of 4, atleast two a's are required, among 4 positions any two can be a's. There are 4C_2 possible ways = 6.

$$\therefore R = r_1 + r_2 + r_3 + r_4 + r_5 + r_6$$

$$\text{where } r_1 = (a + b)(a + b)(a)(a)$$

$$r_2 = (a)(a)(a + b)(a + b)$$

$$r_3 = (a)(a + b)(a)(a + b)$$

$$r_4 = (a + b)(a)(a)(a + b)$$

$$r_5 = (a + b)(a)(a + b)(a)$$

$$r_6 = (a)(a + b)(a + b)(a)$$

$$\therefore R = [(a + b)(a + b)aa] + [aa(a + b)(a + b)] + [a(a + b)a(a + b)] \\ + [(a + b)aa(a + b)] + [(a + b)a(a + b)a] + [a(a + b)(a + b)a]$$

Example 26: Write regular expression for language containing strings without consecutive 0's or 1's or without consecutive 0's followed by without consecutive 1's over {0, 1}.

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

Example 27: Define the language of the following regular expression :

$$(a + b)^* a (\epsilon + bbb)$$

Solution: It defines the language made up of zero or more occurrences of a or b ending with "a" or "abbb".

Example 28: Define the language of following regular expression:

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

Solution: It defines the language containing set of strings of even length in which "0" is at even position.

Example 29: Find the regular expression for the set of strings not having "101" as a substring.

Solution: In this, we should take care that if 1 is followed by 0 (i.e. 10) then it should be followed by another 0 or nothing to avoid (101). Thus the regular expression is $0^*(1^*(00^+)^*)^*$.

Example 30: Find the regular expression for the set of all strings such that every block of four consecutive symbols contains atleast two zeros.

Solution: This means that always we have to search four symbols and see that atleast two zeros are there i.e. if A, B, C, D are four symbols then any of these two symbols should be zero. i.e. AB, or BC, or CD, or AC or AD or BD.

Thus the regular expression is,

$$[00(0+1)(0+1) \mid 0(0+1)0(0+1) \mid 0(0+1)(0+1)0 \mid (0+1)00(0+1) \mid (0+1)0(0+1)0 \mid (0+1)(0+1)00]^*$$

Example 31: Find the regular expression for the set of all strings such that the fifth symbol from the right end is 1.

Solution: The regular expression is $(0+1)^*1(0+1)(0+1)(0+1)(0+1)$.

2.1.3 Regular Expression Identities

[Oct. 16]

- Two regular expressions P and Q are equivalent and written as $P = Q$ if P and Q represent the same set of strings.
- Following are the identities for regular expressions which are useful for simplification of regular expression :
 1. $\phi + R = R$
 2. $\phi R = R\phi = R$
 3. $\epsilon R = R\epsilon = R$
 4. $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$
 5. $R + R = R$
 6. $R^* R^* = R^*$
 7. $RR^* = R^* R = R^+$
 8. $(R^*)^* = R^*$
 9. $\epsilon + RR^* = R^* = \epsilon + R^* R$
 10. $(PQ)^* P = P (QP)^*$
 11. $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)$
 12. $(P + Q) R = PR + QR$
 13. $R (P + Q) = RP + RQ$

Arden's Theorem:

- The Arden's theorem can be applied to find the regular expression. It states that, if P and Q are two regular expressions over Σ . If P does not contain ϵ , then $R = Q + RP$ has a unique solution given by, $R = QP^*$.

Example 1: Example show that $(a \cdot b)^* \neq a^* b^*$

Solution: Let $L_1 = (a \cdot b)^*$ and $L_2 = a^* b^*$

$$L_1 = \{\epsilon, ab, abab, \dots\}$$

$$L_2 = \{\epsilon, a, aa, b, bb, ab, aabb, \dots\}$$

The strings accepted by both languages are not same.

$$\therefore (ab)^* \neq a^* b^*$$

Example 2: Show that $(1 + 011)^* = \epsilon + 1^* (011)^* (1^* (011)^*)^*$

$$\begin{aligned} \text{Solution: } (1 + 011)^* &= \epsilon + 1^* (011)^* + (1^* (011)^*)^* \\ &= \epsilon + R \cdot R^* && \text{where } R = 1^* (011)^* \\ &= \epsilon + R^+ && \text{(using identity of regular expression)} \\ &= R^* \\ &= (1 + 011)^* \end{aligned}$$

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

Example 3: Justify true or false : $(01)^* 0 = 0 (10)^*$

Solution: True, since $(PQ)^* P = P (QP)^*$ (identity of regular expression)

Example 4: Show that regular expressions given below are equivalent.

$$(b + aa^* b) + (b + aa^* b) (a + bab)^* \delta (a + ba^* b) = a^* b (a + ba^* b)^*$$

$$\text{Solution: L.H.S.} = (b + aa^* b) + (b + aa^* b) (a + ba^* b)^* (a + ba^* b)$$

$$\text{Let } b + aa^* b = P$$

$$\text{and } a + ba^* b = Q$$

$$\begin{aligned} \therefore \text{L.H.S.} &= P + PQ^* Q \\ &= P + PQ^* Q \\ &= P (\epsilon + Q^* Q) = PQ^* (\epsilon + RR^* = R^*) \\ &= (b + aa^* b) (a + ba^* b)^* \\ &= (\epsilon + aa^*) b (a + ba^* b)^* \\ &= a^* b (a + ba^* b)^* \\ &= \text{R.H.S.} \end{aligned}$$

Example 5: Show that $\epsilon 0^* + (0^* 1) 1^* = 0^* 1^*$

Solution:

$$\begin{aligned}
 \text{L.H.S.} &= \epsilon 0^* + (0^* 1) 1^* \\
 &= 0^* + (0^* (11)^*) && \text{(identity property)} \\
 &= 0^* + (\epsilon + 11^*) \\
 &= 0^* (1^*) = 0^* 1^* \\
 &= \text{R.H.S.}
 \end{aligned}$$

2.2 REGULAR LANGUAGES

[April 17, 19, Oct. 17]

- A language is regular if it can be expressed in terms of regular expression.

2.2.1 Definition

- The regular languages are those languages that can be constructed from the three set operations viz., Union, Concatenation and Kleene closure.
- A regular language is defined as, let Σ be an alphabet. The class of "regular languages" over Σ is defined inductively as follows:
 1. ϕ is a regular language.
 2. For each $a \in E$, $\{a\}$ is a regular language.
 3. For any natural number $n \geq 2$ if L_1, L_2, \dots, L_n are regular languages, then $L_1 \cup L_2 \cup \dots \cup L_n$ is regular language.
 4. For any natural number $n \geq 2$, if L_1, L_2, \dots, L_n are regular languages, then $L_1 \cdot L_2 \cdot \dots \cdot L_n$ is regular language.
 5. If L is a regular language, then L^* is regular language.

2.2.2 Examples

- If $L = \phi$, ϕ is a regular language by rule (a).
- $L = \{a, ab\}$ is a language over $\Sigma = \{a, b\}$ because, both $\{a\}$ and $\{b\}$ are regular languages by rule (b). By rule (d) it follows that $\{a\} \cup \{b\} = \{ab\}$ is a regular language. Using rule (c), we see that $\{a\} \cup \{ab\} = L$ is a regular language.
- The language over the alphabet $\{0, 1\}$ where strings contain an even number of 0's can be constructed by $1^*((01)^*(01)^*)^*$ or simply $1^*(01^*01^*)^*$.

2.3

EQUIVALENCE OF FINITE AUTOMATA (FA)

AND REGULAR EXPRESSION (RE) [April 16, 17, 18, 19, Oct. 17, 18]

- In this section we will study how to transform any FA to an equivalent RE and conversion of any RE to an equivalent FA. Thereby, establish the equivalence between REs and FAs.

- **Theorem:** Let r be a regular expression. There exists an NFA with ϵ -transition that accepts $L(r)$.
- **Proof:** The languages accepted by FA are precisely the languages denoted by regular expression.

2.3.1 Conversion from RE to FA

- In this section we will study how to convert the RE to FA. Consider the pictorial representation in Fig. 2.1 of RE to FA.

Case 1:

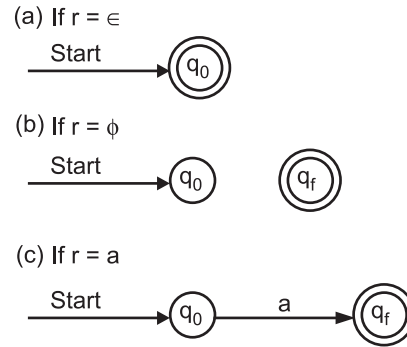


Fig. 2.1

Case 2: If

$$r = r_1 + r_2 \text{ (for union)}$$

there are NFAs,

$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$$

and

$$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\}) \text{ with}$$

$$L(M_1) = L(r_1) \text{ and } L(M_2) = L(r_2)$$

Construct,

$$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$$

$$q_0 = \text{new initial state, } f_0 = \text{new final state.}$$

$$L(M) = L(M_1) \cup L(M_2)$$

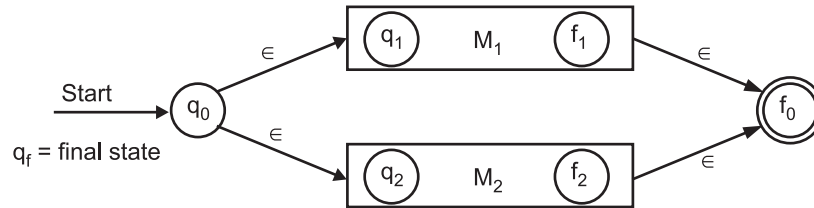


Fig. 2.2

Case 3: If

$$r = r_1 r_2 \text{ (for concatenation)}$$

M_1 and M_2 are same as above,

$$L(M) = L(M_1) L(M_2)$$



Fig. 2.3

Case 4: $r_1 = r_1^*$ (for closure)

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ and $L(M_1) = r_1$

Construct, $M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$

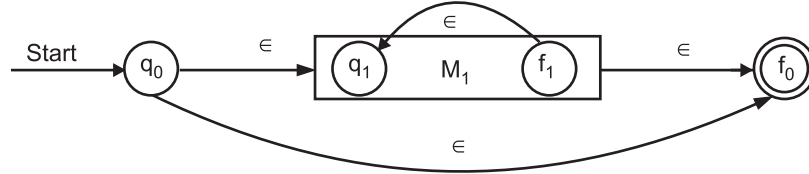


Fig. 2.4

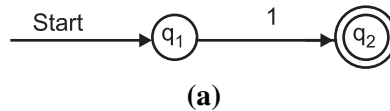
- Any path from q_0 to f_0 is either path from q_0 to f_0 on ϵ or a path from q_0 to q_1 on ϵ followed by some number of paths from q_1 to f_1 , then back to q_1 on ϵ , each labeled by a string in $L(M_1)$ followed by path q_1 to f_1 on string in $L(M_1)$ then to f_0 on ϵ .

Example 1: Construct NFA for regular expression $01^* + 1$.

Solution: $01^* + 1 = r_1 + r_2$

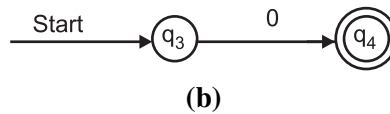
where, $r_1 = 01^*$ and $r_2 = 1$.

NFA for r_2 is,

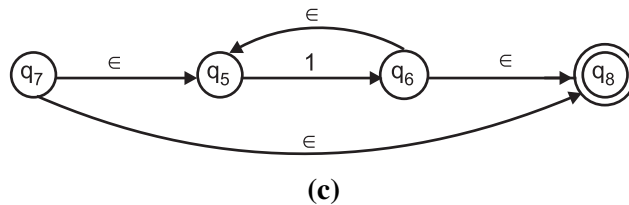


NFA for $r_1 = r_3 r_4$, where $r_3 = 0$ and $r_4 = 1^*$.

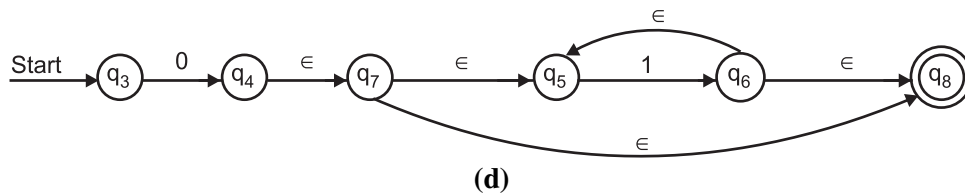
NFA for r_3 is,



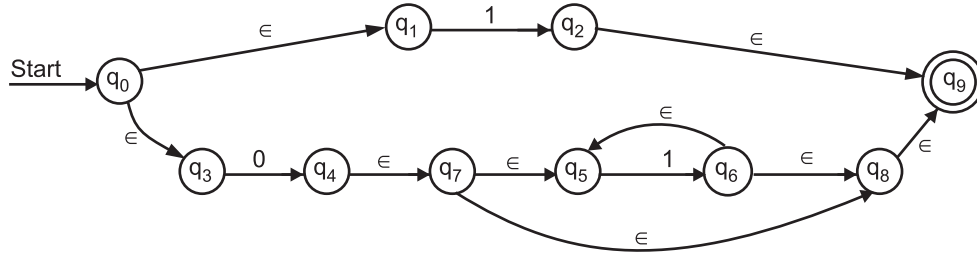
NFA for r_4 is,



NFA for $r_1 = 01^*$ is,



NFA for $01^* + 1$ is $r_1 + r_2$ as follows:



(e)

Fig. 2.5

Example 2: Construct a FA equivalent to regular expression $(1^* + 0)^*$.

Solution:

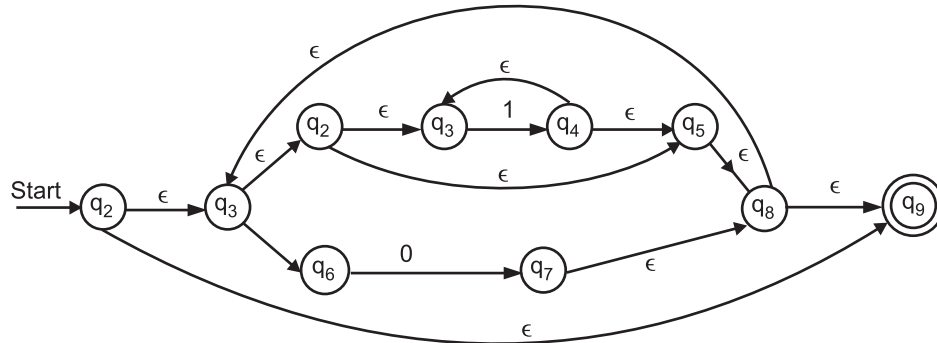


Fig. 2.6

Example 3: Construct a FA equivalent to regular expression $(01 + 10)^* + 11$.

Solution: $(01 + 10)^* + 11 = r_1 + r_2$, where $r_1 = (01 + 10)^*$ and $r_2 = 11$.

NFA for r_1 is M_1 as shown below.

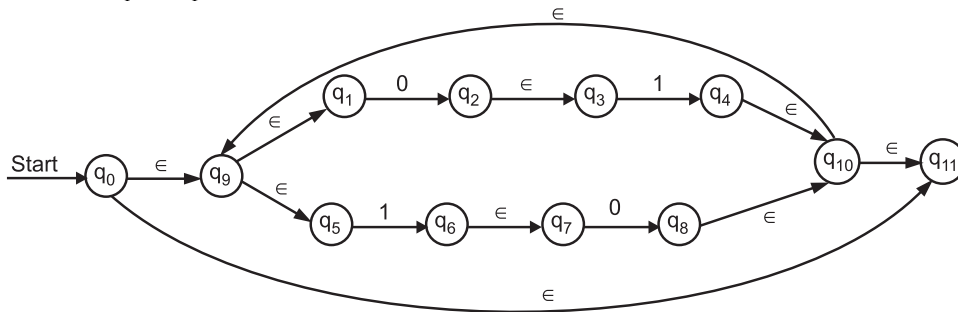


Fig. 2.7 (a)

NFA for r_2 is M_2 as shown in Fig. 3.7 (b).



Fig. 2.7 (b)

Therefore, equivalent NFA is as shown in the Fig. 3.7 (c).

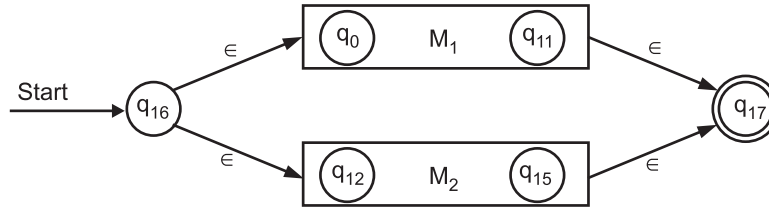


Fig. 2.7 (c)

Example 4: (Direct method) For DFA shown in the Fig. 2.8, find the regular expression.

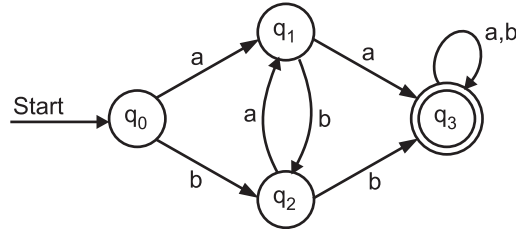


Fig. 2.8

Solution: If the start symbol is 'a', we can find different expressions to reach to final state q_3 from q_0 via q_1 and are as follows :

$a (ba)^* a (a + b)^*$ and $a (ba)^* bb (a + b)^*$.

Similarly, if the start symbol is 'b' then we have,

$b (ab)^* b (a + b)^*$ and $b (ab)^* aa (a + b)^*$

Therefore, the required regular expression is obtained by "ORing all above expressions,

i.e. $a (ba)^* a + a (ba)^* bb (a + b)^* + (ab)^* b (a + b)^* + b (ab)^* aa (a + b)^* a (a + b)^*$.

Example 5: For the DFA shown in the Fig. 3.9, find the regular expression.

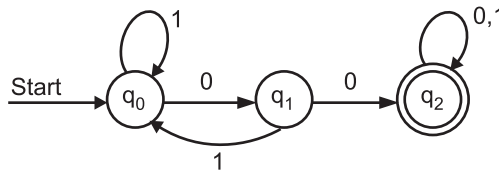


Fig. 2.9

Solution : The regular expression is $1^* 00 (0 + 1)^* + (1^* 01)^* 00 (0 + 1)^*$.

Example 6: For the DFA shown in the Fig. 3.10, find the regular expression.

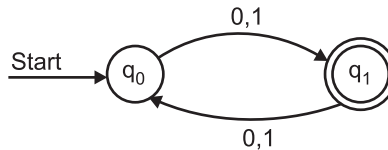


Fig. 2.10

Solution: The required regular expression is $(0 + 1) (0 + 1)^*$.

Example 7: Language L_1 consists of strings beginning with 'a' and language L_2 consists of strings ending with 'a'.

- What is L_1 intersection L_2 ?
- Write regular expression for L_1 and L_2 and $L_1 \cap L_2$.
- Draw FA.

Solution:

- L_1 intersection L_2 is all words starting with 'a' and ending with 'a'.
- $L_1 : r_1 = a (a + b)^*$
 $L_2 : r_2 = (a + b)^* a$
 $L_1 \cap L_2 : r_3 = a (a + b)^* a + a.$
-

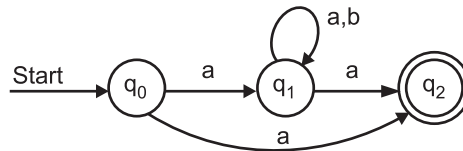


Fig. 2.11

Example 8: Give regular expression of $L_1 \cap L_2$ if

L_1 : all strings of even length

L_2 : starting with 'b'.

Solution: L_1 intersection L_2 is all strings of even length and starting with 'b'.

$L_1 = ((a + b) (a + b))^*$

$L_2 = b (a + b)^*$

Now, for intersection the string should start with 'b' and length is to be even. Hence, 'b' should be concatenated by a string of odd length.

$((a + b) (a + b))^*$ is even length, adding an extra symbol we get the odd length.

Hence, $((a + b) (a + b))^* (a + b)$

Therefore, the required regular expression is

$b ((a + b) (a + b))^* (a + b).$

2.4

PUMPING LEMMA FOR REGULAR LANGUAGES AND APPLICATIONS

[April 16, 17, Oct. 17]

- By using FA and RE, we have been able to define many languages, but language which cannot be defined by a regular expression is called a non-regular set.
- All languages are regular or non-regular. Pumping lemma, which is a powerful tool for proving certain languages, is non-regular.

- The term Pumping Lemma is made up of two words:
 1. **Pumping:** The word pumping refers to generate many input strings by pushing a symbol in an input string again and again.
 2. **Lemma:** The word Lemma refers to intermediate theorem in a proof.
- **Theorem:** Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$, we may write $z = uvw$ such that $|uv| \leq n$, $|v| \geq 1$ and for all $i \geq 0$, $uv^i w \in L$. Furthermore, n is a number greater than the number of states of the smallest FA accepting L .
- **Proof:** Let DFA $M = (Q, \Sigma, \delta, q_0, F)$ with some particular number of states, say n . Consider an input of n or more symbols $a_1, a_2 \dots a_m$, $m \geq n$ and for $i = 1, 2, \dots, m$, let $\delta(q_0, a_1, a_2 \dots a_i) = q_i$.
- It is not possible for each of the $n + 1$ states q_0, q_1, \dots, q_n to be distinct, since there are only n different states. Thus there are two integers j and k , $0 \leq j < k \leq n$, such that $q_j = q_k$.
- The path labelled $a_1, a_2 \dots a_m$ in the transition diagram of M is as shown in the Fig. 2.12.
- Since, $j < k$, the string $a_{j+1} \dots a_k$ is of length atleast 1 and since $k \leq n$, its length is no more than n .

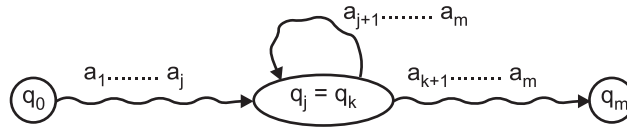


Fig. 2.12: Path in transition diagram of DFA M

- If q_m is in F , i.e. $a_1, a_2, \dots, a_m \in L(M)$.

$$\begin{aligned}
 \delta(q_0, a_1, a_2 \dots a_j a_{k+1} \dots a_m) &= \delta(\delta(q_0, a_1 \dots a_j), a_{k+1} \dots a_m) \\
 &= \delta(q_j, a_{k+1} \dots a_m) \\
 &= \delta(q_k, a_{k+1} \dots a_m) \\
 &= q_m
 \end{aligned}$$
- Similarly, we could go around the loop of Fig. 3.12 more than once. Thus $a_1 \dots a_j (a_{j+1} \dots a_k)^i a_{k+1} \dots a_m$ is in $L(M)$ for any $i \geq 0$.

2.4.1 Applications

- We will use above theorem to prove that given language is not regular.
 1. Select the language L which we wish to prove non-regular.
 2. Assume L is regular and let n be number of states in the corresponding FA.
 3. Select a string z such that $|z| \geq n$. Use pumping lemma to write $z = uvw$ with $|uv| \leq n$ and $|v| \geq 1$.
 4. Find a suitable integer i such that $uv^i w \notin L$.
- Above contradicts our assumption and hence L is non-regular.

Example 1: Show that the set $L = \{0^{i^2} \mid i \text{ is an integer, } i \geq 1\}$ which consists of all strings of 0's whose length is a perfect square, is not regular.

Solution:

Case 1:

1. Assume L is regular set and let n be the integer in the pumping lemma. Let $z = 0^{n^2}$.
2. By the pumping lemma, 0^{n^2} may be written as uvw where $1 \leq |v| \leq n$ and $uv^i w$ is in L for all i .
3. Let $i = 2$ (say).

The length of uv^2w is $|uvv w| > n^2$ always. Since, v is at least one and maximum n , therefore, the maximum length of string is $n^2 + n$.

Hence, $n^2 < |uv^2w| \leq n^2 + n < (n + 1)^2$ i.e. the length of uv^2w lies properly between n^2 and $(n + 1)^2$ and is thus not a perfect square. Thus uv^2w is not in L , a contradiction.

Therefore, L is not regular.

Case 2: $L = \{0, 0000, 000000000, \dots\}$

1. Assume L is regular set.
2. Let $z = 0000$ (any word from language)
3. Split z into uvw such that $|uv| \leq |z|$ and $|v| \geq 1$.
Suppose $u = 0, v = 00, w = 0$.
4. Let $i = 2$.

$$\begin{aligned} uv^i w &= uv^2 w \\ &= 0 (00)^2 0 \\ &= 000000 \notin L \end{aligned}$$

$$\therefore uv^i w \notin L$$

Hence, L is non-regular.

Example 2: Show that $L = \{a^p \mid p \text{ is prime}\}$ is non-regular.

Solution:

Case 1:

1. Let L be regular and n be number of states in FA accepting L .
2. Let p is a prime number and $p \geq n$.
 $u = a^m, v = a$ and $w = a^{p-(m+1)}, 1 \leq n$.

So, $|uv| \leq n$ and $|v| = 1$.

By pumping lemma, $uv^i w$ is in L .

Let, $i = p + 1$

$$\begin{aligned} \therefore a^m (a)^{p+1} a^{p-m-1} &\in L \\ a^{m+p+1+p-m-1} &= a^{2p} \in L \end{aligned}$$

but $2p$ is not a prime number.

So, $uv^{p+1} w \in L$ is a contradiction. Hence, L is not regular.

Case 2: $L = \{a, aa, aaa, aaaaa, \dots\}$

1. Assume L be a regular set.
2. Let $z = aaaaa$ be a word such that
 $z = uvw \in L$ (By pumping lemma)
 Let $u = aa, v = a, w = aa$
 Let $i = 2$

$$\begin{aligned} uv^i w &= uv^2 w \\ &= aa (a)^2 aa \\ &= aaaaaa \notin L \end{aligned}$$

$$\therefore uv^i w \notin L$$

Hence L is non-regular.

Example 3: Show that $L = \{ww \mid w \in (a, b)^*\}$ is not regular.

Solution:

1. Let L be regular. Let n be the number of states in FAM accepting L .
2. Let,
 $ww = a^m b a^m b \in L$
 Let,
 $|ww| = 2(m+1) = n$

We can have two different ways.

Case 1: v does not contain any b 's, so $k > 1$ and $k \neq m$, $u = a^k$, $v = a^{m-k}$, $w = ba^m b$.

$$\therefore |uv| < n, |v| \geq 1$$

By pumping lemma, $uw \in L$.

$$\text{But } uw = a^k b a^m b \in L$$

This is a contradiction. Hence, L is not regular.

Case 2: v contains only one b .

$$u = a^k, v = b, w = a^m b$$

$$\therefore |uv| < n \text{ and } |v| \geq 1$$

By pumping lemma $uw \in L$.

$$\text{But, } uw = a^k a^m b \in L.$$

This is a contradiction. Hence L is not regular.

Example 4: Determine whether following language is regular ? Justify.

$$L = L_1 \cap L_2$$

$$L_1 = \{a^n b^m \mid n \geq m \text{ and } n, m > 0\}$$

$$L_2 = \{b^m a^n \mid m > 0, n > 1\}$$

Solution: L_1 contains all strings starting with 'a' and L_2 contains all strings starting with 'b'.

$$\therefore L = L_1 \cap L_2 = \phi$$

We can construct FA for L as shown in the Fig. 2.13.

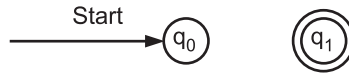


Fig. 2.13

Hence, L is a regular language.

Example 5: Show that $L = \{0^n 1^n \mid n \geq 1\}$ is not regular.

Solution:

1. Let L is regular.
2. Let $z = 0^n 1^n$, then $|z| = 2n > n$
By pumping lemma, split $z = uvw$ such that $|uv| \leq n$ and $|v| \geq 1$.
3. Let $z = 000111$ and $u = 00$, $v = 011$, $w = 1$
 $|uv| \leq 6$ satisfies the condition.
4. Let $i = 2$.

$$\begin{aligned}
 uv^i w &= uv^2 w \\
 &= 00 (011)^2 1 \\
 &= 000110111 \notin L
 \end{aligned}$$

\therefore L is not regular.

Example 6: Prove that $L = \{a^n b^{n+1} \mid n > 0\}$ is non-regular.

Solution: Let,

$$\begin{aligned}
 n &= 1 \\
 a^n b^{n+1} &= ab^2 \\
 |ab^2| &= 1 + 2 = 3 \\
 \text{Let } n &= 2 \quad |a^2 b^3| = 5
 \end{aligned}$$

All strings of this language are of odd length.

1. Assume L be regular.
2. $z = a^3 b^4$. $|z| = 7$
Split z into uvw such that $|uv| \leq n$, $|v| \geq 1$.
Let $u = aa$, $v = ab$, $w = bbb$.

$$\begin{aligned}
 z &= aa ab bbb \\
 &= u v w
 \end{aligned}$$

3. Let $i = 2$

$$\begin{aligned}
 uv^i w &= uv^2 w \\
 &= (aa) (ab)^2 bbb \\
 &= aa ab ab bbb \notin L
 \end{aligned}$$

\therefore Language L is not regular.

2.5

CLOSURE PROPERTIES OF REGULAR LANGUAGES

[April 16, 17, Oct. 17]

- In this section, we discuss the closure properties of regular sets under union, concatenation, Kleen closure, intersection, complementation, etc.
- These operations when we apply on regular sets result in regular set and if class of languages is closed under particular operation we call that fact a closure property of the class of languages.
- **Theorem 1:** The regular sets are closed under union, concatenation and kleen closure. i.e. if L_1 and L_2 are regular sets, then $L_1 \cup L_2$, $L_1 L_2$ and L_1^* are also regular sets.
- **Proof:** $L_1 + L_2$ means the language of all words in either L_1 or L_2 , say regular expressions for L_1 and L_2 are r_1 and r_2 respectively. Then we define,
 $r_1 + r_2$ is regular expression for $L_1 \cup L_2$,
 $r_1 r_2$ is regular expression for $L_1 L_2$ and r_1^* is regular expression for L_1^* .
- Therefore, all three of these sets of words are definable by regular expressions and so are themselves regular sets.

Example: Let, L_1 = Strings starts with and ends with same letter over $\{a, b\}$
 L_2 = Strings containing “ab” as substring over $\{a, b\}$

Regular expression for language L_1 is

$[a(a+b)^*a + b(a+b)^*b]$

Regular expression for language L_2 is

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

1. The language represented by regular expression $L_1 + L_2$ is
 $[a(a+b)^*a + b(a+b)^*b] + (a+b)^*ab(a+b)^* \quad \dots (1)$
 Therefore, language $L_1 \cup L_2$ is regular because it is represented by regular expression (1).
2. The language $L_1 L_2$ is represented by regular expression
 $[a(a+b)^*a + b(a+b)^*b] \cdot [(a+b)^*ab(a+b)^*]$
 Therefore, it is regular.
3. The language L_1^* is represented by regular expression
 $(a(a+b)^*a + b(a+b)^*b)^*$
 Therefore, it is regular.

- **Theorem 2:** The class of regular sets is closed under complementation.
- **Definition:** If L is language over alphabet Σ , we define its complement L' to be the language of all strings of letters from Σ that are not words in L (or L' or \bar{L}) i.e. L is a regular set and $L \subseteq \Sigma^*$ then $\Sigma^* - L$ is a regular set.

- **Proof:** Let L be $L(M)$. Some of states of this FA, M are final states and some are not. Let us reverse the status of each state i.e. if it was a final state, make it non-final and if it was non-final state, make it final state. If the input ends formerly with non-final state, it now ends in final state and vice-versa. This new FA accepts all strings that were not accepted by the original FA (L'). Therefore, the machine accepts the language L' . Therefore L' is a regular set.

Example: Construct DFA for a language over $\{a, b\}$ which contains all strings not having "bba" as a substring.

Solution: First construct DFA which accepts substring "bba".

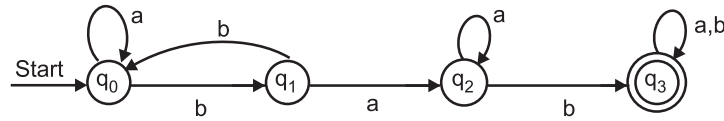


Fig. 2.14

Now, complement the final state. We get the required DFA as shown in the Fig. 3.15.

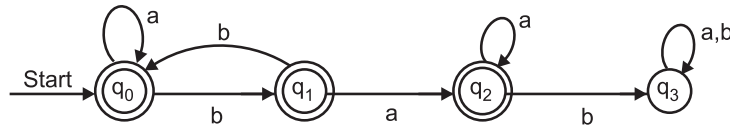


Fig. 2.15

- **Theorem:** The regular sets are closed under intersection (i.e. if L_1 and L_2 are regular sets, then $L_1 \cap L_2$ is also a regular set).
- **Proof:** By DeMorgan's law for sets of any kind

$$L_1 \cap L_2 = (L_1' + L_2')'$$

- This means the language $L_1 \cap L_2$ consists of all words that are not in either L_1' or L_2' . Since L_1 and L_2 are regular sets, then L_1' and L_2' are regular sets proved by theorem 3.4.2. By theorem 3.4.1, since L_1' and L_2' are regular then $L_1' + L_2'$ is also a regular set.

Again by theorem 3.4.2, $(L_1' + L_2')'$ is a regular set.

Therefore, $L_1 \cap L_2$ is a regular set.

Example: Construct DFA for $L = L_1 \cap L_2$ over $\{0, 1\}$, where

L_1 = starting with 0 and ending with 11

L_2 = containing substring "010" in it.

Solution: $L = L_1 \cap L_2 =$ All strings with 0 ending with 11 and having substring 010 in it.

DFA of L is as shown in the Fig. 2.16.

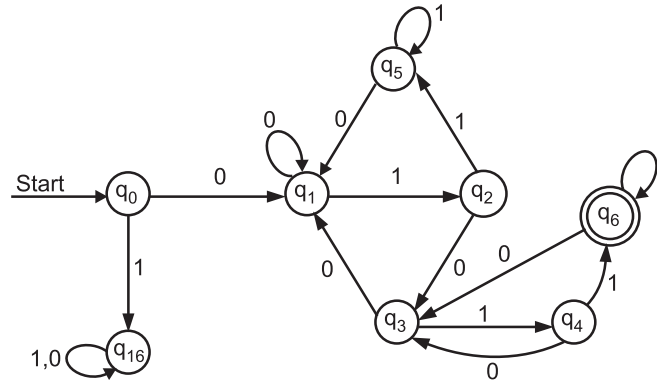


Fig. 2.16

2.6 OPERATIONS ON REGULAR LANGUAGES

- In this section we will study operations on regular languages.
- Union:** Union of languages L_1 and L_2 is the language (L) containing all strings of L_1 and all strings of L_2 .

$$L = L_1 \cup L_2$$

- Example:**

If $L_1 = a^*, L_2 = b^*$

then $L = L_1 \cup L_2 = a^* b^* + b^* a^*$

- Concatenation:** The concatenation of languages L_1 and L_2 is the language L containing all strings of L_1 followed by all strings of L_2 without space.

$$L = L_1 L_2$$

- Example:**

$L_1 = \{a, bb, bab\}$ and $L_2 = \{a, ab\}$

$L = L_1 L_2 = \{aa, aab, bba, bbab, baba, babab\}$

- Intersection:** Intersection of languages L_1 and L_2 is the language L containing common strings of L_1 and L_2 .

$$L = L_1 \cap L_2$$

- Example:**

If, $L_1 = a^*$ and $L_2 = b^*$ then

$L = L_1 \cap L_2 = \phi$ (No string common)

- **Kleene Closure:** Given an alphabet Σ , we wish to define a language in which any string of letters from Σ is a word, even the null string. This language we shall call the closure of the alphabet. It is denoted by writing a star (asterisk) after the name of the alphabet as a superscript.
- The Σ^* is a notation called Kleene star or Kleene closure.

- **Example:**

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

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

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

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 011, \dots\}$$

3. If $\Sigma = \{a, b, c\}$ then

$$\Sigma^* = \{\epsilon, a, b, c, aa, bb, cc, ac, ab, ba, bc, ca, cb, aaa, \dots\}$$

4. If $S = \{aa, b\}$ then

$$S^* = \{\epsilon, aa, b, aab, baa, bbb, aaaa, aabb, \dots\}$$

The string aabaaab is not in S^* since it has a's of length 3.

5. If $S = \{a, ab\}$ then

$$S^* = \{\epsilon, a, ab, aa, aab, aba, aaaa, aaab, aaba, abab, aaaab, abaaa, \dots\}$$

Here, for each word in S^* every b must have an 'a' immediately to its left.

6. If $S = \{10, 1\}$ then

$$S^* = \{\epsilon, 10, 1, 1010, 11, 101110, \dots\}$$

Therefore, Kleene closure of language is denoted by L^* and is defined as

$$L^* = \bigcup_{i=0}^{\infty} L_i$$

- **Positive Closure:** If S is a set of strings not including ϵ , then S^+ is the language S^* without the word ϵ . If S is a language that does contain ϵ , then $S^+ = S^*$.
- This plus operation is called positive closure. Therefore, positive closure of language is L^+ and is defined as

$$L^+ = \bigcup_{i=0}^{\infty} L_i$$

- **Example:**

1. If $S = \{10, 1\}$ then

$$S^+ = \{10, 1, 1010, 11, 101, 110, \dots\}$$

2. If $S = \{aa, b\}$ then

$$S^+ = \{aa, b, aab, baa, bbb, aaaa \dots\}$$

- Closure operator can be applied to infinite set or finite set.

EXAMPLES

Example 1: Find the regular expression for the set of strings not having "101" as a substring.

Solution: In this, we should take care that if 1 is followed by 0 (i.e. 10) then it should be followed by another 0 or nothing to avoid (101). Thus the regular expression is $0^*(1^*(00^+)^*)^*$.

Example 2: Find the regular expression for the set of all strings such that every block of four consecutive symbols contains atleast two zeros.

Solution: This means that always we have to search four symbols and see that atleast two zeros are there i.e. if A, B, C, D are four symbols then any of these two symbols should be zero. i.e. AB, or BC, or CD, or AC or AD or BD.

Thus the regular expression is

$$[00(0+1)(0+1) \mid 0(0+1)0(0+1) \mid 0(0+1)(0+1)0 \mid (0+1)00(0+1) \mid (0+1)0(0+1)0 \mid (0+1)(0+1)00]^*$$

Example 3: Find the regular expression for the set of all strings such that the fifth symbol from the right end is 1.

Solution: The regular expression is $(0+1)^*1(0+1)(0+1)(0+1)(0+1)$.

Example 4: Prove $L = \{a^n b^n a^n\}$ is not regular.

Solution: By pumping lemma, $w = a^n b^n a^n$ split into xyz (say)

$$w \in L \quad \therefore xyz \in L.$$

Case 1:

$$x = \epsilon$$

$$y = a$$

$$z = b^n a^n$$

$$\therefore w = a^i b^n a^n \quad \text{such that } xy^iz \in L$$

Thus we can show that this is not regular.

Case 2:

$$x = a^n$$

$$y = b$$

$$z = a^n$$

$$\therefore w = a^n b^i a^n, \text{ again for this } i \neq n, w \text{ generates a word which is not in } L.$$

Case 3:

$$x = a^n b^n$$

$$y = a$$

$$z = \epsilon$$

$$\therefore w = a^n b^n a^i$$

Thus, for any value of i, w will not generate valid words. Thus it is not regular.

Case 4: $x = \epsilon$
 $y = aba$
 $z = \epsilon$
 $\therefore w = (aba)^i$
 For $i = 1$, $w = aba \in L$
 For $i = 2$, $w = abaaba$ is not in L .
 Thus, the language is not regular from case 1 through case 4.
 $\therefore L = \{a^n b^n a^n\}$ is not regular.

Example 5: Prove $L = \{a^n \mid b^n\}$ valid words in $L = \{a, b, \dots\}$ is regular using pumping lemma.

Solution: Here again y can be either a or b .

Case 1: $x = \epsilon$
 $y = a$
 $z = \epsilon$

Thus, $w = a^i$ and for all i , w will generate a word which is in L . Thus, the language is regular.

Case 2: $x = \epsilon$
 $y = b$
 $z = \epsilon$
 $\therefore w = b^i$

Again for this, w generates words which are in L .

Thus the language is regular. Thus $L = \{a^n \mid b^n\}$ is regular.

Example 6: Draw a FA for a language that will not accept strings "aba" and "abb" over alphabet $\Sigma = \{a, b\}$.

Solution: The FA that accepts only the strings "aba" and "abb" is shown below.

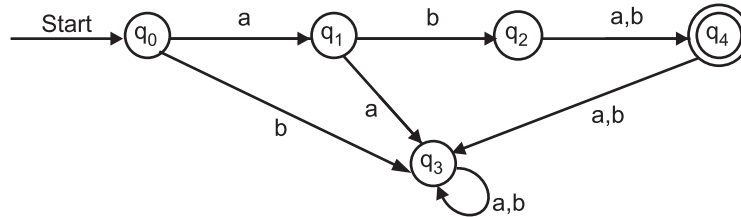


Fig. 2.17

An FA that accepts all strings other than "aba" and "abb" is complement of above language as shown below.

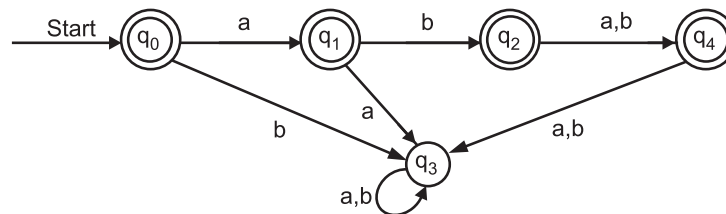


Fig. 2.18

Here, we have to reverse the final / non-final status of the states.

Example 7: Construct a NFA for the following regular expression :

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

Solution: Let

$$r_1 = (ab + ba)^*$$

$$r_2 = aa$$

$$R = r_1 r_2 r_1$$

NFA for r_1 is

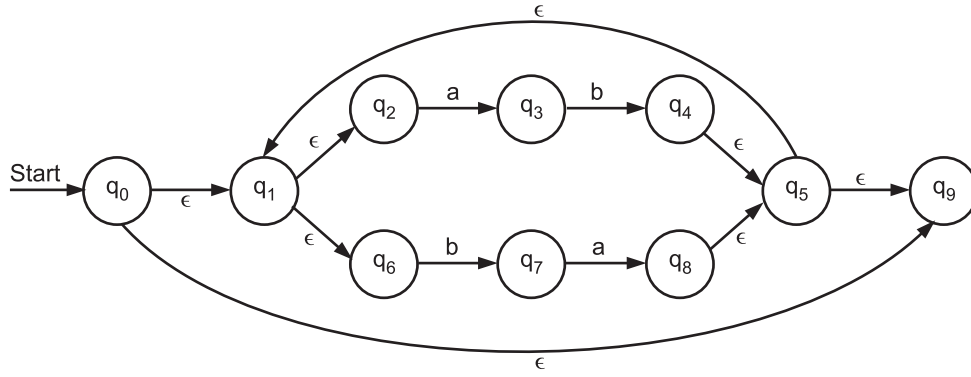


Fig. 2.19 (a)

NFA for r_2 is

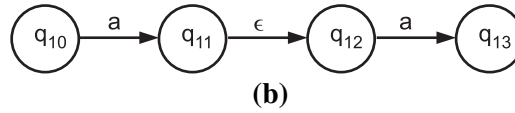


Fig. 2.19 (b)

\therefore NFA for R is

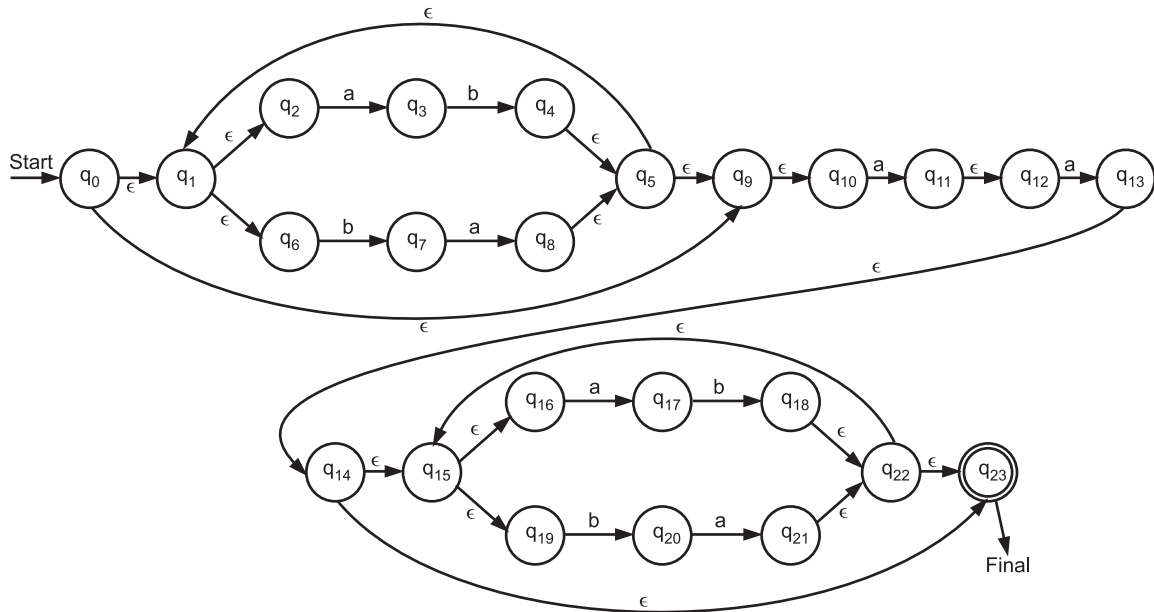


Fig. 2.19 (c)

Example 8: Construct NFA for the following regular expression:

$$(0(11)^*0 + 01^*0)^*$$

Solution:

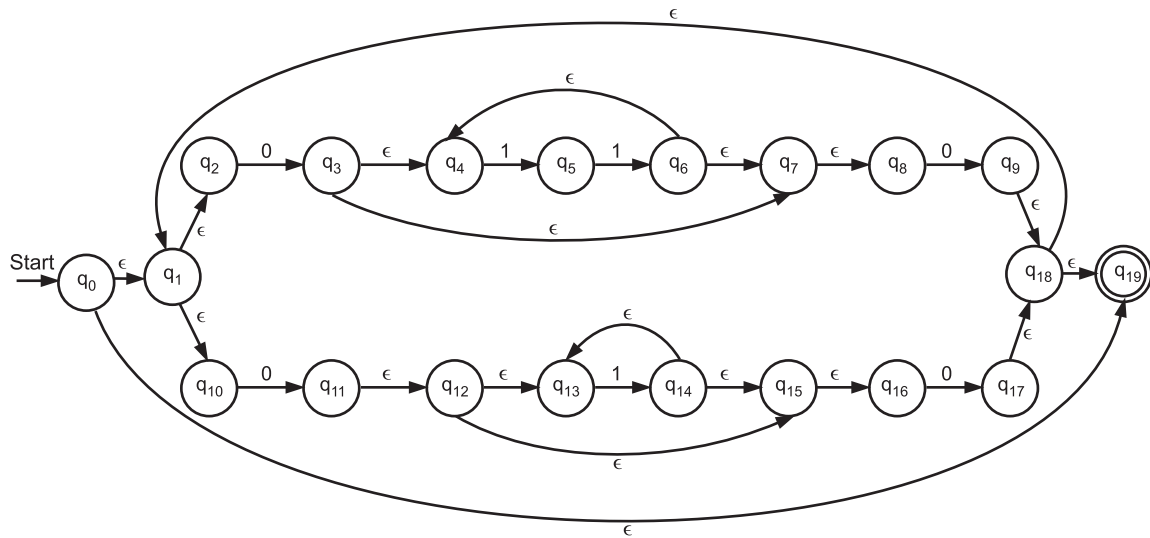


Fig. 2.20

EXAMPLES

Example 1: Describe the language $L = \{a^n b^n \mid n \geq 1\}$.

Solution: $\epsilon = \{a, b\}$

$$L = \{ab, aabb, aaabb, \dots\}$$

The language contains equal number of a's followed by equal no. of b's.

Example 2: If $L_1 = (a^*, b^* + b^* a^*)$ and $L_2 = (a + b)^*$ find $L_1 \cap L_2$

Solution: $L_1 = \{\epsilon, ab, ba, abb, aabb, baa, bbaa, \dots\}$

$$L_2 = \{\epsilon, a, b, ab, ba, aab, baa, \dots\}$$

$$L_1 \cap L_2 = L_1$$

Example 3: Find the language of the following : $(a^* b^*)^* a$.

Solution: Language L containing a string starts with a followed by a's followed by b's and ends with a $L = \{a, aba, aabba, aaabba, \dots\}$

Example 4: What is the suffix and prefix of string $x = abcd$.

Solution: Suffix = $\epsilon, d, cd, bcd, abcd$

$$\text{Prefix} = \epsilon, a, ab, abc, abcd$$

Example 5: Find the language of the following : $(a + b)^*$

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

Example 6: In $(a^* + b^*)^* = (a + b)^*$ true ? Justify.

Solution: LHS = $\{\epsilon, a, b, aa, bb, ab, ba, bba, aab, \dots\}$

RHS = $\{\epsilon, a, b, aa, bb, ab, ba, bba, \dots\}$

LHS = RHS

Example 7: Write smallest possible string accepted by given RE $(0 + 1^*) 01^*$.

Solution: 00

Example 8: Write smallest possible strings accepted by the following regular expression

$10 + (0 + 11) 0^* 1$

Solution: {10, 01}

Example 9: Construct FA for the following regular expression :

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

Solution:

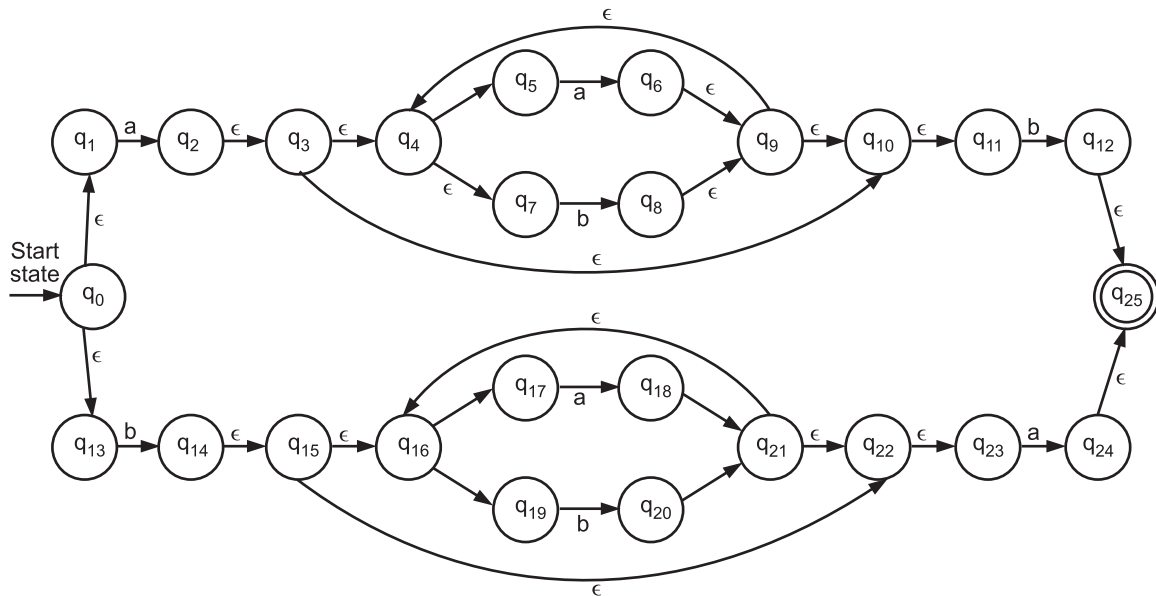


Fig. 2.21

Example 10: Construct NFA without ϵ for language $L = (0 + 1)^* 01$

Solution:

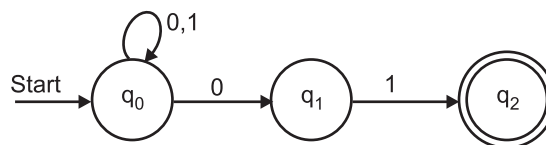


Fig. 2.22

Example 11: Write smallest possible string accepted by the following regular expression :

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

Solution: $L = (ab + ba^*)^* b$

Smallest string is b.

Example 12: Construct FA for the following regular expression :

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

Solution:

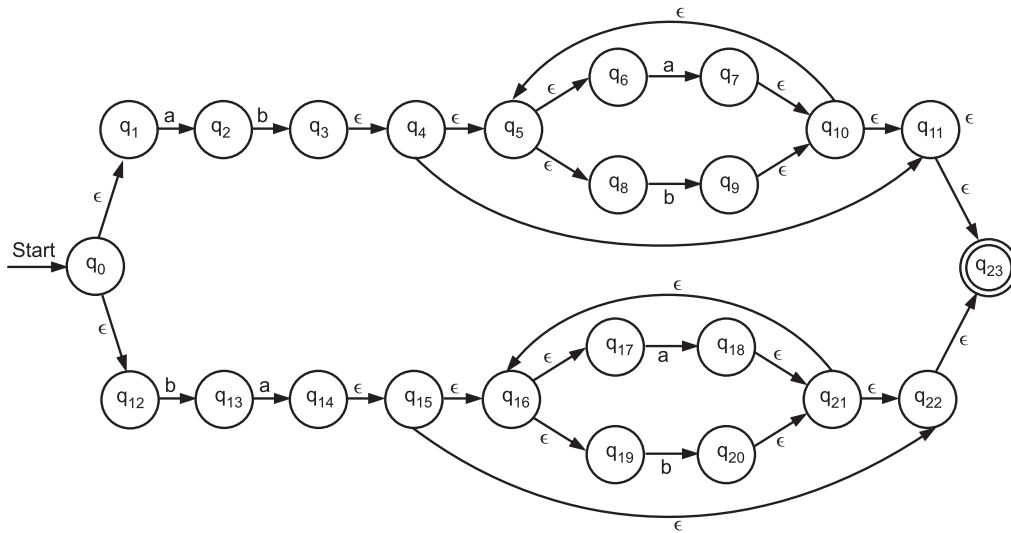


Fig. 2.23

Example 13: If ϵ is a regular expression, then it denotes set $\{\epsilon\}$. Justify.

Solution: True, since ϵ is a regular expression.

Example 14: Construct NFA without ϵ for language

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

Solution:

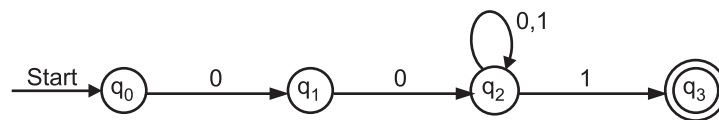


Fig. 2.24

Example 15: Write smallest possible string accepted by the regular expression :

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

Solution: aab is the smallest string.

Example 22: Construct NFA with ϵ -transition for : $ab^* + ba^*$

Solution:

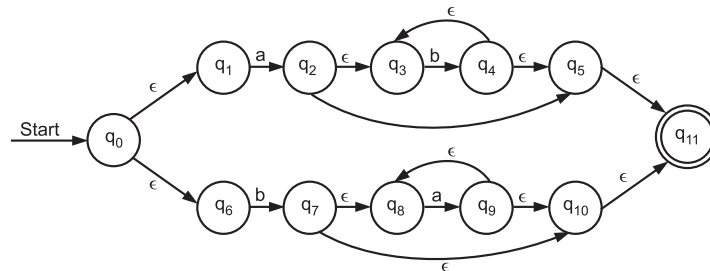


Fig. 2.27

Example 23: Write smallest possible string generated by regular expression :

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

Solution: Smallest string = aa or ab

Example 24: Draw FA equivalent to regular expression : $a(a+b)^* + b(b+a)^*$

Solution:

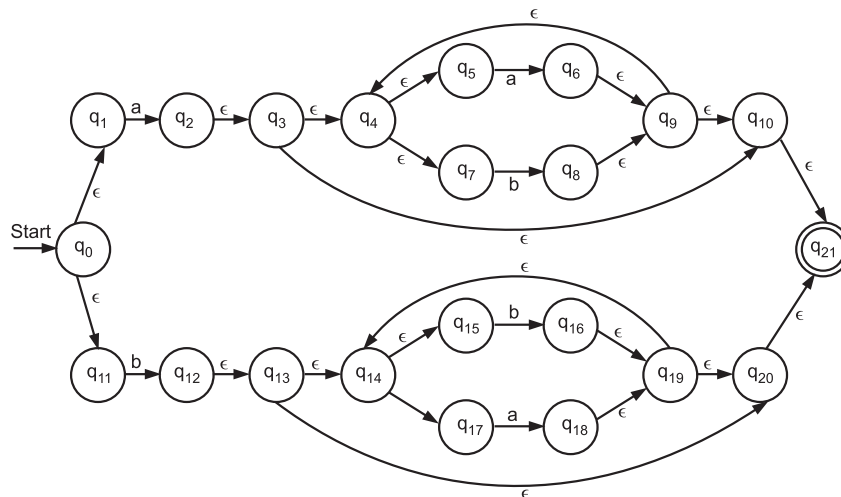


Fig. 2.28

Example 25: Give two kinds of operations that can be carried out on regular language.

Solution: (i) Union, (2) Intersection. (Refer section 3.6.2 for examples)

Example 26: Write the smallest string generated by regular expression.

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

Solution: bac or bbc.

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

- Which is a set of strings of symbols over an alphabet?
 - Expression
 - Grammar
 - Language
 - None of the mentioned
- A language is generated from the rules of a,
 - expression
 - grammar
 - language
 - None of the mentioned
- Which can be defined as a language or string accepted by an FA.
 - Regular Expression (RE)
 - Regular Grammar (RG)
 - Regular Language (RL)
 - None of the mentioned
- The machine format of regular expression is,
 - Push down automata
 - Turing machine
 - Finite Automata (FA)
 - None of the mentioned
- The language of all words with at least 2 a's can be described by the regular expression,
 - $(a + b)^*ab^*a(a + b)^*$
 - $b^*ab^*a(a + b)^*$
 - $(ab)^*a$
 - None of the mentioned
- The pumping lemma for regular expression is used to prove that,
 - Certain sets are not regular
 - Regular grammar don't produce RE
 - Certain sets are regular
 - Certain regular grammar produce RE
- Which of the languages is accepted by the following FA?
 - $a(a + bb^*a^*)^*b^*d$
 - $b(a + bba^*)^*a^*b$
 - $a(a + bba^*)^*b^*$
 - None of the mentioned
- Which of the strings do not belong to the regular expression $(ba + baa)^*aaba$,
 - babaabaaaba
 - baaaba
 - baaaaba
 - babababa
- A finite automata recognizes,
 - Context sensitive language
 - Context-free language
 - Regular language
 - None of the mentioned
- Which of the following regular expressions describe the language over $\{0, 1\}$ consisting of strings that contain exactly two 1's?
 - $0^*10^*10^*$
 - $(0 + 1)^*1(0 + 1)^*1(0 + 1)^*$
 - 0^*110^*
 - $(0 + 1)^*11(0 + 1)^*$

11. Given the language $L = \{ab, aa, baa\}$, which of the following strings are in L^* ?
 - (a) baaaaabaa
 - (b) baaaaabaaaab
 - (c) aaaabaaaa
 - (d) abaabaaabaa
12. Which one of the following regular expressions is not equivalent to the regular expression $(a + b + c)^*$?
 - (a) $(a^*b^*c^*)^*$
 - (b) $(a^*b^* + c^*)$
 - (c) $(a^* + b^* + c^*)^*$
 - (d) $((ab)^* + c^*)^*$
13. The set of all strings over $\{a, b\}$ of even length is represented by the regular expression,
 - (a) $(ab + aa + bb + ba)^*$
 - (b) $(a + b)^*(a^* + b)^*$
 - (c) $(aa + bb)^*$
 - (d) $(ab + ba)^*$
14. The set of strings over $\{a, b\}$ having exactly 3a's is represented by the regular expression,
 - (a) b^*aaa
 - (b) $b^*ab^*ab^*a$
 - (c) ab^*ab^*a
 - (d) $b^*ab^*ab^*ab^*$
15. $\{a^{2n} \mid n \geq 1\}$ is represented by regular expression
 - (a) $(aa)^*$
 - (b) a^*
 - (c) aa^*a
 - (d) a^*a^*
16. (0^*1^*) is equivalent to,
 - (a) $(0 + 1)^*$
 - (b) $(01)^*$
 - (c) $(10)^*$
 - (d) None of the mentioned.

Answers

1. (c)	2. (b)	3. (a)	4. (c)	5. (a)	6. (c)	7. (b)	8. (d)	9. (c)	10. (a)
11. (b)	12. (d)	13. (d)	14. (d)	15. (a)	16. (d)				

Q.II Fill in the Blanks:

1. A regular _____ is a formal language that can be expressed using a regular expression.
2. _____ is accepted by the machine called Finite Automata (FA).
3. _____ is used to prove that given language is not regular.
4. The language specified by a _____ referred as regular language.
5. _____ are those languages that are described by regular expressions and can be accepted by FA.
6. _____ of regular expressions used for simplifying RE.
7. Pumping lemma, which is a powerful tool for proving certain languages is _____.

8. The property which describes when we combine any two elements of the set and the result is also included in the set is called as_____.

Answers

1. language	2. RE	3. Pumping Lemma	4. regular expression
5. Regular languages	6. Identities	7. non-regular	8. ϵ -closure

Q.III State True or False:

- The languages accepted by Finite Automata (FA) are called as regular expressions.
- The regular languages are those languages that can be constructed from the three set operations viz., Union (\cup), Concatenation (\cdot) and Kleene closure ($*$)
- Pumping Lemma should never be used to show a language is regular.
- A regular expression is written as RE or regex or regexp.
- To prove a given language is regular we use Pumping Lemma.
- The language that is accepted by FA is known as Regular language.
- A regular expression can be defined as, a language or string accepted (recognisable) by FA.
- Every FA can have regular expression.
- FA can accept only non-regular sets.
- $(R^*)^* = R^*$ is true or false.
- Regular sets are closed under union.
- Every finite subset of Σ^* is a regular language.
- Every regular language over Σ is finite.
- The regular expression $(0 + 1)^* 2^* = 0^* (1 + 2)^*$.
- $aa^* + bb^* = (a + b)^*$.
- The Language $\{0,1\}^* - \{0101\}$ is regular.

Answers

1. (F)	2. (T)	3. (T)	4. (T)	5. (F)	6. (T)	7. (T)	8. (T)	9. (F)	10. (T)
11. (T)	12. (F)	13. (F)	14. (T)	15. (F)	16. (T)				

Q.IV Answer the following Questions:

(A) Short Answer Questions:

- Define regular language.
- Define regular expression.
- List any four identities of regular expression.
- What is purpose of Pumping Lemma?
- Develop the finite automata for $(a^*ab + ba)^*a$ regular expressions.
- Give closure properties of regular languages.

(B) Long Answer Questions:

1. What is regular language? Explain with example.
2. What is regular expression? Explain with example.
3. Describe in English the sets denoted by the following regular expressions :
 - (i) $(11 + 0)^* (00 + 1)^*$.
 - (ii) $(1 + 01 + 001)^* (\epsilon + 0 + 00)$
 - (iii) $[(00 + 11) + (01 + 10) (00 + 11)^* (01 + 10)]^*$
4. Construct finite automata equivalent to the following regular expressions :
 - (i) $10 + (0 + 11) 0^* 1$
 - (ii) $((a + bb)^* a a)^*$
 - (iii) $(ab + ba)^* a (ab^*)^*$
 - (iv) $01 [((10)^* + 111)^* + 0]^* 1$
 - (v) $0^* (1 + 0)^* 10 + 11$
 - (vi) $(ab + ba)^* a (ab^*)^*$
 - (vii) $a^* b (ab^* + ba) c^*$
5. Check whether the following languages are regular?
 - (i) $\{a^n b^m \mid n \geq m\}$
 - (ii) $\{a_1 a_3 a_5 \dots a_{2n-1} \mid a_1 a_2 a_3 \dots a_{2n} \text{ is in } L\}$
 - (iii) $L = \{0^n \mid n \text{ is a prime}\}$
 - (iv) $\{a^i b^{2i+1} \mid i \geq 1\}$
 - (v) $\{0^p 1^q 0^{p+q} \mid p \geq 1, q \geq 1\}$
 - (vi) $\{xx^R w \mid x, w \text{ in } (0 + 1)^+\}$
 - (vii) $\{0^m 1 0^{m+n} \mid m \geq 1 \text{ and } n \geq 1\}$
 - (viii) $\{0^a 1^b \mid \gcd(a, b) = 1\}$
 - (ix) $L = \{0^n 1^m \mid m \geq n, n > 0\}$
 - (x) $L = L_1 \cap L_2$ where
 $L_1 =$ all strings over $\{a, b\}$ starting with a and ending with b.
 $L_2 =$ all strings over $\{a, b, c\}$ having 'abb' as a substring in it.
6. Find $L = L_1 \cap L_2$ where
 $L_1 =$ all strings starting with 'a' containing even number of b's over $\{a, b\}$
 $L_2 =$ all strings containing odd number of b's over $\{a, b\}$.
7. Find $L = L_1 \cap L_2$ where
 $L_1 =$ All strings starting with 'a'.
 $L_2 =$ All strings not having "ab" as substring over $\{a, b\}$.

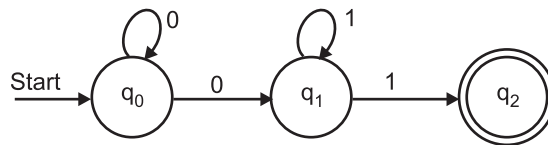
8. Find $L = L_1 L_2$ where
 $L_1 = a^*$ and $L_2 = b^*$.
9. Find $L = L_1 \cup L_2$ where
 $L_1 = \{a^n b^n \mid n \geq 1\}$
 $L_2 = \{a^n b^n c^i \mid n \geq 1, i \geq 1\}$

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Write the regular expression for the following FA:

[1 M]



Ans. Refer to Section 2.1.

2. State pumping lemma of regular set.

[1 M]

Ans. Refer to Section 2.4.

3. Construct FA for the following regular expression:
 $(010 + 00)^* (10)^*$.

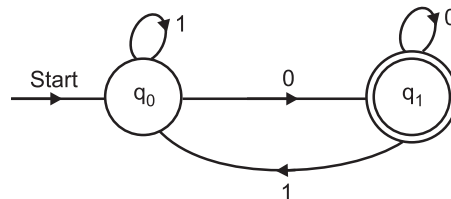
[5 M]

Ans. Refer to Section 2.3.

October 2016

1. Give the regular expression for the following DFA:

[1 M]



Ans. $r = (1 + 0)^*.0$.

2. Give any two identities of regular expression.

[1 M]

Ans. Refer to Section 2.1.3.

April 2017

1. Define regular expression.

[1 M]

Ans. Refer to Section 2.1.

2. Which tool is used to prove that the language is not regular?

[1 M]

Ans. Refer to Section 2.2.

3. Define Kleene Closure.

[1 M]

Ans. Refer to Page 2.2, Point (2).

4. Construct a FA for the given RE = $(a^*b + b^*a).ab + ba.b^*$.

[5 M]

Ans. Refer to Section 2.3.

October 2017

1. Write regular expression for the set of all strings of a's and b's ending with ab over $\Sigma = \{a, b\}$.

[1 M]

Ans. Refer to Section 2.2.

2. Construct NFA for regular expression $1.0^* + 0^*$.

[3 M]

Ans. Refer to Section 2.3.

3. How to apply pumping lemma to prove certain language are non-regular?

[3 M]

Ans. Refer to Section 2.4.

April 2018

1. Write smallest possible string accepted by the following regular expression:

[1 M]

$(01 + 10^*)^* 1$

Ans. Refer to Section 2.1.

2. Construct FA for regular expression:

[5 M]

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

Ans. Refer to Section 2.3.

October 2018

1. Write the smallest possible string accepted by regular expression: $(0 + 1)^* 01^*$.

[1 M]

Ans. Refer to Section 2.1.

2. Construct FA for the following regular expression: $(010 + 00)^* (10)^*$.

[4 M]

Ans. Refer to Section 2.3.

April 2019

1. Write smallest possible string accepted by the following regular expression.

[1 M]

$(11)^* (00)^* + (10+01)^*.1$

Ans. Refer to Section 2.1.

2. Construct FA for regular expression:

[2 M]

$(0 + 1)^* 01 + (1 + 0)^* 11$

Ans. Refer to Section 2.3.

3. Define the term: Kleene closure.

[1 M]

Ans. Refer to Page 2.2, Point (2).

4. Show that CFL's are closed under concatenation.

[2 M]

Ans. Refer to Section 2.2.



Context-Free Grammars and Languages

Objectives ...

- To study Basic Concept of Grammar
- To learn Context Free Grammar
- To understand Normal Forms
- To learn Regular Grammar

3.0 INTRODUCTION

- A language consists of a finite or infinite set of sentences. Infinite language is specified by a grammar.
- A grammar consists of a finite non-empty set of rules or productions, which specify the syntax of language.
- Number of grammar may generate the same language but consists of different structures on the sentences of that language.
- Even though human language's have rules of grammar. For example, English language has grammar and grammar has rules.
- Another method for language specification is to have a machine, called an acceptor, determine whether a given sentence belongs to the language.
- In 1959, Chomsky, cataloged the hierarchy of grammars according to the structure of their productions, which we will discuss further.
- Context-Free Grammar (CFG) is a formal grammar which is used to generate all possible patterns of strings in a given formal language.
- The context-free languages are the languages which can be represented by CFG.

3.1 GRAMMAR

- Grammar can be defined as, a set of formal rules for generating syntactically correct sentence from a particular language for which it is written.

- For any type of language [Formal (like 'C') or Natural (like English)] is required to have a grammar, which can be defined in syntactically correct statement formats or in turn we can say that it is the syntactic definition of the language, or in other words, we can also say that, grammar defines syntax of a language.
- For example: If we want to generate an English statement "Dog runs", we have to use the following rules:

$$\langle \text{sentence} \rangle \rightarrow \langle \text{noun} \rangle \langle \text{verb} \rangle$$

$$\langle \text{noun} \rangle \rightarrow \text{Dog}$$

$$\langle \text{verb} \rangle \rightarrow \text{runs}$$

- These rules describe how the sentence can be generated as 'noun' followed by 'verb' and so on. There are many such rules which can be defined for English language and collectively can be called as grammar for the language.
- Grammar normally consists of two types of basic elements, namely Terminal symbols and Non-terminal symbols or variables.

1. **Terminal symbols** are those which are the constituents of the generated sentence, which we have generated using a grammar. For example, in the above example, 'Dog' and 'runs' are terminal symbols.

2. **Non-terminal symbols** are those which take part in the formation or generation of the statement, but are not the part of generated statement like terminal symbols. For example, 'sentence', 'noun' and 'verb' are non-terminals in the above example, which are not in the generated statement as,

$$\begin{array}{ccc} \langle \text{sentence} \rangle & \xrightarrow{\text{gives}} & \langle \text{noun} \rangle \langle \text{verb} \rangle \\ & \xrightarrow{\text{gives}} & \text{Dog runs.} \end{array}$$

- The rules of the grammar are called as productions or production rules or syntactical rules.

Example:

Sentence = Omkar ate an apple.

We use the following rules for sentence.

$$\langle \text{sentence} \rangle \rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle$$

$$\langle \text{predicate} \rangle \rightarrow \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$$

$$\langle \text{subject} \rangle \rightarrow \langle \text{noun} \rangle$$

$$\langle \text{noun} \rangle \rightarrow \text{Omkar} \mid \text{apple}$$

$$\langle \text{verb} \rangle \rightarrow \text{ate}$$

$$\langle \text{article} \rangle \rightarrow \text{an}$$

Using these rules we can derive the sentence

$\langle \text{sentence} \rangle \Rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle$
 $\Rightarrow \text{Omkar} \langle \text{predicate} \rangle$
 $\Rightarrow \text{Omkar} \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{Omkar ate} \langle \text{article} \rangle \langle \text{noun} \rangle$
 $\Rightarrow \text{Omkar ate an} \langle \text{noun} \rangle$
 Omkar ate an apple.

- So using the production rules we derive a sentence. We also say that a sentence belongs to the language of that grammar.

3.1.1 Definition of the Grammar

- The two aspects of the language which is to be considered are:
 1. Generative capacity.
 2. Grammatical constituents.
- The first aspect indicates that the grammar should generate all and only those sentences which belong to the language. The second aspect deals with primitive structures, like terminals and non-terminals.
- A grammar based on the constituent structure as described above is called as phrase structure grammar.
- **Formal Definition:** A phrase structure grammar or simply a grammar is quadruple denoted as,

$$G = (V, T, P, S)$$
 where

$$V : \text{finite set of non-terminals (variables)}$$

$$T : \text{finite set of terminals}$$

$$S : S \text{ is a non-terminal symbol called as start symbol}$$
 and

$$P : \text{finite set of productions of the form}$$

$$\alpha \rightarrow \beta \text{ where } \alpha, \beta \in (V \cup T)^*$$
 and α has atleast one symbol from V .
- The α and β consist of any number of terminals and non-terminals and they are called as sentential forms.

3.1.2 Examples of the Grammar

- Following is the example of the grammar:

$$S \rightarrow ABC \mid d$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

$$C \rightarrow cC \mid c$$

Here, Set of Non terminals $V = \{S, A, B, C\}$

Set of terminals $T = \{a, b, c, d\}$

Set of production rules,

$$P = \{S \rightarrow ABC, S \rightarrow d, A \rightarrow aA, A \rightarrow a, B \rightarrow bB, B \rightarrow b, C \rightarrow cC, C \rightarrow c\}$$

And starting symbol of the grammar is S.

3.2

DERIVATIONS AND LANGUAGE GENERATED BY A GRAMMAR

[April 16]

- The sequence of application of production rules that produces the finished string of terminals from the start symbol is called a derivation.
- If $A \rightarrow \beta$ is a production of p and $\alpha, \gamma \in (V \cup T)^*$, then $\alpha A \gamma \Rightarrow \alpha \beta \gamma$. We say that $\alpha A \gamma$ directly derives $\alpha \beta \gamma$ in grammar G . i.e. production $A \rightarrow \beta$ is applied on $\alpha A \gamma$ to get $\alpha \beta \gamma$.
- There are two different derivations possible, namely
 1. Leftmost derivation.
 2. Rightmost derivation.
- The language generated by G denoted by $L(G)$ is $\{w \mid w \text{ is in } T^* \text{ and } S \xRightarrow{*} w\}$. Thus string is in $L(G)$ iff
 1. String consists of only terminals.
 2. The string can be derived from S .
- A language generated by a CFG is called as context-free language, abbreviated as CFL.
We say G_1 and G_2 are equivalent if $L(G_1) = L(G_2)$.

3.2.1 Derivation Trees or Parse Tree

[April 16, 19]

- It is the graphical representation of derivation. The derivation in the CFG be represented using a tree is called as derivation tree.
- **Definition:** Let $G = (V, T, P, S)$ be a CFG. A tree is derivation tree for G (parse tree) if
 1. Every vertex has a label which is a symbol $V \cup T \cup \{\epsilon\}$.
 2. The root node is always start symbol, denoted by S .
 3. A vertex is interior and has label A , then A must be in V .
 4. If the string w of the language generated by grammar has a length n , then there are n terminal nodes arranged from left to right.
 5. Vertex n has label ϵ , then n is a leaf and is the only son of its father.

Example 1: Consider grammar $G = (\{S, A, B\}, \{a, b\}, P, S)$ where

P is $\{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$. We can derive the string "ab" as $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$.

Solution: Derivation tree is shown in the Fig. 3.1.

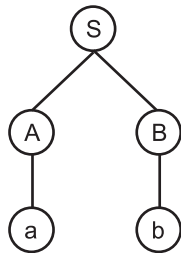


Fig. 3.1

$|ab| = 2$
There are two terminal nodes arranged from left to right.

Example 2: Draw the derivation tree for a substring "001100" using the following grammar G:

$G = (\{S, A\}, \{0, 1\}, \{S \rightarrow 0AS \mid 0, A \rightarrow S1A \mid SS \mid 10\}, S)$

Solution: Since S is the start symbol, string is derived from S.

$S \Rightarrow 0AS \Rightarrow 0S1AS \Rightarrow 001AS \Rightarrow 00110S \Rightarrow 001100$

The derivation tree is shown in the Fig. 3.2.

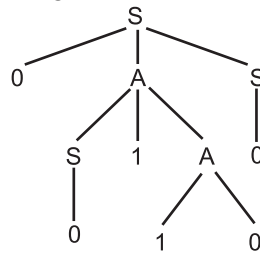


Fig. 3.2

3.2.2 Left and Right Derivations

- If at each step in a derivation, a production is applied to the **leftmost** variable (non-terminal), then the derivation is called as LeftMost Derivation (LMD).
- If at each step in a derivation, a production is applied to the **rightmost** variable (non-terminal) then the derivation is called as RightMost Derivation (RMD).

Example 1: Consider the following grammar $G = (\{S, A, B\}, \{a, b\}, P, S)$

where, P is

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Find leftmost and rightmost derivations for the string "bbaaba".

Solution: 1. Leftmost derivation:

$S \Rightarrow b\underline{A}$
 $\Rightarrow bb\underline{A}A$
 $\Rightarrow bba\underline{S}A$
 $\Rightarrow bbaa\underline{B}A$
 $\Rightarrow bbaab\underline{A}$
 $\Rightarrow bbaaba$

2. Rightmost derivation:

$$\begin{aligned}
S &\Rightarrow b\underline{A} \\
&\Rightarrow bb\underline{AA} \\
&\Rightarrow bb\underline{A}a \\
&\Rightarrow bba\underline{S}a \\
&\Rightarrow bbaa\underline{B}a \\
&\Rightarrow bbaaba
\end{aligned}$$

Example 2: Consider the following grammar: $G = (\{S, A\}, \{a, b\}, P, S)$

Where, P is

$$S \rightarrow AbaaA \mid aA$$

$$A \rightarrow Aa \mid Ab \mid \epsilon$$

Find the leftmost and rightmost derivation for the string “abaabb”.

Solution: 1. Leftmost derivation:

$$\begin{aligned}
S &\Rightarrow \underline{A}baaA \\
&\Rightarrow \underline{A}abaaA \\
&\Rightarrow abaa\underline{A} \\
&\Rightarrow abaa\underline{A}b \\
&\Rightarrow abaa\underline{A}bb \\
&\Rightarrow abaabb
\end{aligned}$$
2. Rightmost derivation:

$$\begin{aligned}
A &\Rightarrow Abaa\underline{A} \\
&\Rightarrow Abaa\underline{A}b \\
&\Rightarrow Abaa\underline{A}bb \\
&\Rightarrow \underline{A}baabb \\
&\Rightarrow \underline{A}abaabb \\
&\Rightarrow abaabb
\end{aligned}$$

3.2.3 Reduction

- Reduction means derivation in reverse. The process starts from a sentence. It finds the string which matches with RHS of the production rule.
 - When the match is found also called handle, then it replace with LHS of same production rule. This is called reduction. The process repeats until we get starting non-terminal symbol. This process is called handle running.
 - Reduction is used in bottom-up parsing.
-

Example: Consider the grammar of expression,

$$E \rightarrow E + E \mid E * E \mid id$$

Find out whether the string $id + id * id$ is generated by this grammar.

<u>id</u> + id * id	(starting string or sentence)
E + <u>id</u> * id	reduce by $E \rightarrow id$
E + E * <u>id</u>	reduce by $E \rightarrow id$ (id is handle)
<u>E + E</u> * E	reduce by $E \rightarrow id$
<u>E * E</u>	reduce by $E \rightarrow E + E$ (handle is $E + E$)
E	(handle is $E * E$)

Since E is the start symbol, the string $id + id * id$ is generated from the grammar or it is yield by grammar of expression.

3.3 CHOMSKY HIERARCHY

[April 16, 17, Oct. 16, 17, 18]

- Linguist Noam Chomsky defined a hierarchy of languages, in terms of complexity. This four-level hierarchy, called the Chomsky hierarchy, corresponds to four classes of machines.
- Each higher level in the hierarchy incorporates the lower levels i.e., anything that can be computed by a machine at the lowest level can be computed by a machine at the next highest level.
- The Chomsky hierarchy classifies grammars according to the form of their productions into the following levels:
 1. Type 0 (unrestricted grammar),
 2. Type 1 (context-sensitive grammar),
 3. Type 2 (context-free grammar), and
 4. Type 3 (regular grammar).

- Let us see above grammars in detail.

1. Type 0 (Unrestricted Grammar):

- There are no restrictions on the productions of grammar of this type. This type of grammar permits productions of the form $\alpha \rightarrow \beta$ with $\alpha \neq \epsilon$, where ' α ' and ' β ' are sentential forms i.e. any combinations of any number of terminals and non-terminals i.e. $\alpha, \beta \in (V \cup T)^*$ but $\alpha \neq \epsilon$. Such grammar is called as unrestricted grammar.
- Unrestricted grammar generates the recursively enumerable languages or every type 0 language forms a recursively enumerable set. i.e. we can construct Turing Machine (TM) to recognize the sentences generated by this type of grammar.
- For example: Grammar, $G = (V, T, P, S)$
 - where $V = \{S, B, C\}, T = \{a, b, c\}$
 - and $P = S \rightarrow SB$
 $SB \rightarrow BC$
 $B \rightarrow a$

2. Type 1 (Context-Sensitive Grammar):**[April 19]**

- The restrictions on this type of grammar are:
 - For each production of the form $\alpha \rightarrow \beta$, length of ' β ' is at least as much as the α except $S \rightarrow \epsilon$.
 - Also, the start symbol 'S' does not appear on the right-hand side of any production.
- The productions are of the form: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ ($\beta \neq \epsilon$)
 where, the replacement of a non-terminal 'A' is allowed by ' β ' only in the context α , preceding 'A' and ' α_2 ' succeeds 'A'.
- For example: Grammar, $G = (\{A, B, C\}, \{a, b\}, P, A)$
 where, P is

$$\begin{aligned} A &\rightarrow AB \\ AB &\rightarrow AC \\ C &\rightarrow ab \end{aligned}$$

Above grammar is context-sensitive grammar.

3. Type 2 (Context Free Grammar):**[Oct. 16]**

- In this grammar, the only allowed type of production is $A \rightarrow \alpha$, where 'A' is a non-terminal and ' α ' is a sentential form i.e. $\alpha \in (V \cup T)^*$ (here, α can be ϵ).
- The left-hand side of the production should contain only one non-terminal.
- For example: Grammar, $G = (\{S\}, \{a, b\}, P, S)$
 where, P is

$$S \rightarrow aSa \mid bSb \mid a \mid b$$

4. Type 3 (Regular Grammar):

- In this type of production, there are following restrictions on the type of production:
 - Left-hand side of each product should contain only one non-terminal.
 - Right-hand side can contain at the most one non-terminal symbol which is allowed to appear as the rightmost symbol or leftmost symbol.
- The language generated using this grammar i.e. regular language is too primitive and can be generated using an finite state machine.
- Depending on the position of a non-terminal, whether leftmost or rightmost, regular grammar is further classified as:
 - Left-linear grammar.
 For example: $G = (\{A, B\}, \{a, b\}, P, A)$

$$\begin{aligned} P: A &\rightarrow Ba \mid a \\ B &\rightarrow b \end{aligned}$$
 - Right-linear grammar.
 For example: $P: A \rightarrow aB \mid a \mid B \rightarrow b$

- The Table 3.1 shows the Chomsky hierarchy of grammars and machine which are acceptors of the grammar.

Table 3.1: The Chomsky hierarchy of grammars and validating machines for languages

Type	Name of languages generated	Production restriction $X \rightarrow Y$	Acceptor	Example of application
0	Phrase-structure = recursively enumerated	X = any string with non-terminals Y = any string	TM	Computers
1	Context sensitive	X = any string with non-terminals Y = any string as long as or longer than X	TM with bounded (not infinite) TAPE, called linear bounded automata LBA.	Computers
2	Context-free (CFL)	X = one non-terminal Y = any string	PDA	Programming languages, statements, compilers.
3	Regular	X = one non-terminal Y = tN or $Y = t$, where t is terminal and N is non-terminal	FA	Text editors

- From the above table we would know that regular languages are accepted by machine or mathematical model called Finite Automata (FA), Non-regular Languages (CFL's) are accepted by Pushdown Automata (PDA) and context sensitive, enumerated languages are accepted by Turing Machine (TM).

3.4 CONTEXT-FREE GRAMMAR (CFG)

[April 16, Oct. 16]

- A CFG is notation that can be used to specify language, and the language specified by a CFG is called as context-free language.

3.4.1 Definition of CFG

- The context free grammar $G = (V, T, P, S)$ where V is finite set of non-terminals.
 T = finite set of terminals
 P = finite set of production rules of the form
Non-terminal \rightarrow terminal + non-terminal

- This form is called Backus Naur Form (BNF) since left hand side of the production rule contains only one non-terminal.

S = start symbol of a grammar

- For example: Grammar of expression,

$$1. \quad E \rightarrow E + E \mid E * E \mid (E) \mid id$$

Here,

$$V = \{E\}$$

$$T = \{+, *, (,), id\}$$

There are 4 production rules and E is start symbol.

$$2. \quad S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Here,

$$V = \{S, A, B\}$$

$$T = \{a, b\}$$

There are three production rules and S is start symbol.

Conventions Regarding Grammars (used in this Chapter):

1. The capital letters A, B, C, D, E and S denote variables, S is the start symbol.
2. Lower case letters a, b, c, d, e digits are terminals.
3. The capital letters X, Y and Z denote symbols that may be either terminal or variable.
4. The lower case letters u, v, w, x, y and z denote string of terminals.
5. The lower case Greek letters α, β and γ denote string of variables and terminals i.e. $(V \cup T)^*$.
6. " \rightarrow " symbol stands for production, For Example: $S \rightarrow a$.
7. " \Rightarrow " symbol stands for process of derivation.
" \Rightarrow^* " stands for deriving in any number of steps.
8. If $A \rightarrow \alpha_1, A \rightarrow \alpha_2 \dots A \rightarrow \alpha_k$ are productions for variable A of some grammar then we can express it using notation $A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \dots \mid \alpha_k$.

3.4.2 Examples

- Consider the grammar:

$$S \rightarrow AB.$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Here, $S \rightarrow AB, A \rightarrow a, B \rightarrow b$ are production rules.

$S \Rightarrow AB \Rightarrow aB \Rightarrow ab$ (The string ab is derived from the S , called derivation).

And, $S \Rightarrow^* ab$ is the derivation from S using number of steps.

3.5 AMBIGUOUS GRAMMAR

[April 16, 17, Oct. 16, 18]

- A grammar is said to be ambiguous if, for a string, there exists more than one parse trees.
- An ambiguous grammar may derive ambiguous strings, which may be undesirable words of a language. Thus, if a grammar is ambiguous, then it has to be modified by making necessary changes in the production rules.
- A CFG is called ambiguous if for at least one word in the language that it generates, there are two possible derivations of the word that correspond to different parse trees or syntax trees.

3.5.1 Concept and Examples

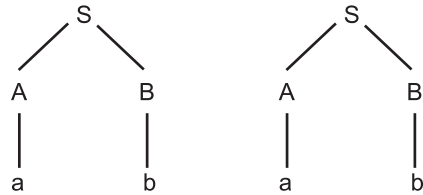
- A CFG is called ambiguous if for at least one word in the language that it generates, there are two possible derivations of the word that correspond to different parse trees or syntax trees.

Example 1: Let consider CFG,

$$\begin{array}{ll} S \rightarrow AB & \text{prod. 1} \\ A \rightarrow a & \text{prod. 2} \\ B \rightarrow b & \text{prod. 3} \end{array}$$

There are two different derivations for the word "ab":

1. $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$ (leftmost derivation)
2. $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$ (rightmost derivation)



(a) Parse Tree for Leftmost Derivation (b) Parse Tree for Rightmost Derivation

Fig. 3.3: Parse Tree

Since the word "ab" represents only one parse tree, therefore, grammar is unambiguous, (only one leftmost derivation is possible).

Hence, we can say that if some string has more than one leftmost or rightmost derivation then it is an ambiguous CFG.

Example 2: Consider the following CFG:

$$\begin{array}{ll} S \rightarrow aB \mid aA \\ A \rightarrow aAB \mid a \mid b \\ B \rightarrow Abb \mid b \end{array}$$

The leftmost derivation for the string "aaabbbbbb"

$$S \Rightarrow aA \Rightarrow aaAB \Rightarrow aaaABB \Rightarrow aaabBB \Rightarrow aaabAbbB \Rightarrow aaabbbbB \Rightarrow aaabbbbbb$$

Clearly the grammar is ambiguous. So find the precedence and associativity of each operator.

Operator	Precedence	Associativity
$+, -$	2	left
$*, /$	4	left
\uparrow (exponent)	6	right

- First level of production rule contain lowest precedence operator. Lower priority symbols are closer to the start symbols.
- Add new non-terminal symbols T and F in the above CFG.
- If operator is left associative, the production rule of that grammar is left recursive.
- If operator is right associative, then the production rule of that grammar is right recursive.

Consider $+, -$ which are having lower priority and left associativity, so production rule should be left recursive.

$$E \rightarrow E + T \mid E - T$$

$$E \rightarrow T$$

Then for operator $*, /$, productions are added.

$$T \rightarrow T * F \mid T / F$$

$$T \rightarrow F$$

id has no associativity.

$$F \rightarrow \text{id}$$

So unambiguous grammar becomes,

$$G = (V, T, P, S)$$

$$\text{where } V = \{E, T, F\} \quad T = \{+, -, *, /, \text{id}\}$$

$$P \text{ is } E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid T$$

$$F \rightarrow \text{id}$$

Now consider string $\text{id} + \text{id} + \text{id}$, we get the following derivation.

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow \text{id} + T \\ &\Rightarrow \text{id} + T * F \\ &\Rightarrow \text{id} + F * F \\ &\Rightarrow \text{id} + \text{id} * F \\ &\Rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$

We get only one parse tree.

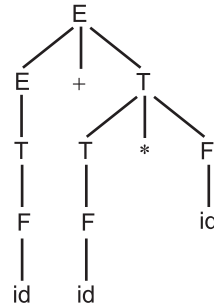


Fig. 3.6

So the above grammar is unambiguous.

EXAMPLES

Example 1: Consider CFG, $G = (\{S\}, \{a\}, \{S \rightarrow aS, S \rightarrow \epsilon\}, S)$. Find the language $L(G)$.

Solution: Firstly consider production $S \rightarrow \epsilon$.

So $S \Rightarrow \epsilon$. Thus ϵ is in $L(G)$.

Now for all $n \geq 1$,

1. $S \Rightarrow aS \Rightarrow a$.
2. $S \Rightarrow aS \Rightarrow aaS \Rightarrow aa$
3. $S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaa$
4. $S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaaS \Rightarrow aaaa$ and so on.

The string a^n comes from n application of production 1 followed by one application of production 2.

The language generated by this CFG is a^* .

Thus, $L(G) = \{\epsilon, a, aa, \dots\}$.

Example 2: Find CFL associated with CFG given below:

$$G = (\{S\}, \{0\}, \{S \rightarrow SS\}, S).$$

Solution: In this case, $L(G) = \emptyset$. This is because only production in G is $S \rightarrow SS$ and in production there is no terminal symbol.

Example 3: Find CFL associated with CFG

$$G = (\{S\}, \{a, b\}, \{S \rightarrow aSb \mid ab\}, S)$$

Solution:

1. $S \rightarrow ab$ production 2
2. $S \Rightarrow aSb \Rightarrow aabb$
3. $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$ and so on.

Thus language $L(G) = \{ab, aabb, aaabbb, \dots\}$

i.e. $L(G) = \{a^n b^n \mid n \geq 1\}$

Example 4: Find CFL associated with CFG

$G = (\{S\}, \{a, b\}, P, S)$, where P is

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Solution:

1. $S \Rightarrow aB \Rightarrow ab$
2. $S \Rightarrow bA \Rightarrow ba$
3. $S \Rightarrow aB \Rightarrow abS \Rightarrow abaB \Rightarrow abab$
4. $S \Rightarrow aB \Rightarrow abS \Rightarrow abbA \Rightarrow abba$
5. $S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabb$
6. $S \Rightarrow bA \Rightarrow baS \Rightarrow babA \Rightarrow baba$
7. $S \Rightarrow bA \Rightarrow bbAA \Rightarrow bbaa$.

Thus CFL, $L(G)$ consists of all strings having equal number of a's and b's.

$\therefore L(G) = \{x \mid x \text{ containing equal number of a's and b's}\}$

Example 5: Construct a context-free grammar which accepts set of palindromes over $\{a, b\}^*$.

Solution: (i) ϵ is a palindrome.

(ii) a and b are palindromes.

(iii) If x is a palindrome then axa and bxb are also palindromes.

To construct grammar G , we use one production for each of above.

So grammar will be, $G = (\{S\}, \{a, b\}, P, S)$

where P is $S \rightarrow \epsilon$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow aSa$

$S \rightarrow aSb$

or $S \rightarrow aSa \mid aSb \mid a \mid b \mid \epsilon$

Example 6: Construct a context-free grammar generating strings with no consecutive b's but may or may not with consecutive a's.

Solution: The grammar G will be as follows:

$G = (\{S, A\}, \{a, b\}, P, S)$

where P is $\{S \rightarrow aS \mid bA \mid a \mid b \mid \epsilon$

$A \rightarrow aS \mid a \mid \epsilon\}$

Example 7: Construct CFG generating $L = \{wcw^R \mid w \in (a+b)^*\}$.

Solution: Here, $w \in \{a, b\}^*$ and $w^R \in \{a, b\}^*$.

$w^R \rightarrow$ is the string reverse of w .

The grammar G will be as follows:

$$G = (\{S\}, \{a, b, c\}, P, S)$$

$$\text{where } P \text{ is } \{S \rightarrow aSa \mid bSb \mid c\}$$

Example 8: Construct CFG for language L , where

$$L = \{a^n b^m c^n \mid n > 1, m > 0\}$$

Solution: Grammar G will be as follows:

$$G = (\{S, A\}, \{a, b, c\}, P, S)$$

$$\text{where } P \text{ is } \{S \rightarrow aSc \mid aAc \mid ac$$

$$A \rightarrow bA \mid b\}$$

Example 9: Construct CFG for $L = L_1 \cup L_2$ where

$$L_1 = \{a^n b^m \mid n \geq m, m > 0\}$$

$$L_2 = \text{All strings not having } 01 \text{ as a substring over } \{0, 1\}.$$

Solution: Language L accepts all strings of L_1 or L_2 since $L = L_1 \cup L_2$.

Let grammar G_1 for $L_1 = (V_1, T_1, P_1, S_1)$ where

$$V_1 = \{S_1, A\}, T_1 = \{a, b\}, P_1 = \{S_1 \rightarrow aS_1b \mid ab \mid aAb \mid A \rightarrow aA \mid a\}$$

Let grammar G_2 for $L_2 = (V_2, T_2, P_2, S_2)$ where

$$V_2 = \{S_2, A_1\}, T_2 = \{0, 1\} \text{ and } P_2 \text{ is}$$

$$\{S_2 \rightarrow 0A_1 \mid 1S_2 \mid 0 \mid 1$$

$$A_1 \rightarrow 0A_1 \mid 0\}$$

\therefore Resultant grammar G for $L = L_1 \cup L_2$ is (V, T, P, S)

$$\text{where } V = V_1 \cup V_2 \cup \{S\}, T = T_1 \cup T_2 = \{a, b, 0, 1\}$$

$$\text{and } P = P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}$$

$$\text{Therefore } P \text{ is, } S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow aS_1b \mid ab \mid aAb$$

$$A \rightarrow aA \mid a$$

$$S_2 \rightarrow 0A_1 \mid 1S_2 \mid 0 \mid 1$$

$$A_1 \rightarrow 0A_1 \mid 0$$

Example 10: Construct a CFG for each of the language defined by the following regular expressions: (i) ab^* , (ii) a^*b^* , (iii) $(baa + abb)^*$.

Solution: 1. The required CFG for ab^* is

$$\begin{aligned} S &\rightarrow aB \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

2. The required CFG for a^*b^* is

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid \epsilon \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

3. The required CFG for $(baa + abb)^*$ is

$$\begin{aligned} S &\rightarrow AS \mid BS \mid \epsilon \\ A &\rightarrow baa \\ B &\rightarrow abb \end{aligned}$$

where 'S' is start symbol and $T = \{a, b\}$.

Example 11: Construct a grammar for $L = \{a^n b^n c^m \mid n \geq 1, m \geq 0\}$.

Solution: The required grammar will be

$$\begin{aligned} G &= (\{S, A\}, \{a, b, c\}, P, S) \\ \text{where } P &= \{S \rightarrow A \mid Sc \\ A &\rightarrow ab \mid aAb\} \end{aligned}$$

Example 12: Construct a CFG for language $L = L_1 \cap L_2$

$$\text{where } L_1 = \{a^n b a^n \mid n \geq 1\}$$

$$L_2 = \text{All strings having odd length over } \{a, b\}.$$

Solution: Since we observe that length of L_1 is odd, therefore, $L_1 \cap L_2 = L_1$.

Required CFG for L is $(\{S\}, \{a, b\}, \{S \rightarrow aSa \mid aba\}, S)$

Note: All examples we discussed in this chapter are called CFG. The property of CFG is that all productions are of the form: one non-terminal \rightarrow string of T and V . This form of production is called as Backus Nour Form (BNF).

Example 13: Let $G = (\{S, A\}, \{a, b\}, P, S)$,

where P is

$$\begin{aligned} S &\rightarrow aAa \\ A &\rightarrow aAa \mid b \end{aligned} \quad \text{construct the language.}$$

Solution:

$$\begin{aligned} S &\Rightarrow aAa \Rightarrow aba \\ S &\Rightarrow aAa \Rightarrow aaAaa \Rightarrow aabaa \\ S &\Rightarrow aAa \Rightarrow aaAaa \Rightarrow aaaAaaa \\ &\Rightarrow aaabaaa \end{aligned}$$

:
:
:

$$L(G) = \{a^n b a^n \mid n \geq 1\}$$

Example 14: Construct a CFG for a language in which all strings with no consecutive b's but may or may not with consecutive a's.

Solution: Here, $L = \{\epsilon, a, b, aa, ba, ab, \dots\}$

The grammar G will be

$$G = (\{S, A\}, \{a, b\}, P, S)$$

where P is $S \rightarrow aS \mid bA \mid a \mid b \mid \epsilon$

$$A \rightarrow aS \mid a \mid \epsilon$$

Example 15: Construct the CFG for language containing the string with atleast “aaa” in it over {a, b}.

Solution: $G = (\{S, A\}, \{a, b\}, P, S)$

where P is $S \rightarrow AaaaA$

$$A \rightarrow aA \mid bA \mid \epsilon$$

Example 16: Find the CFL generated by the following grammar.

$$S \rightarrow AB$$

$$A \rightarrow aA \mid bB \mid a$$

$$B \rightarrow Ba \mid Bb \mid a$$

Solution:

1. $S \Rightarrow AB \Rightarrow aa$
2. $S \Rightarrow AB$
 $\Rightarrow aAB \Rightarrow aaa$
3. $S \Rightarrow AB$
 $\Rightarrow bBB$
 $\Rightarrow bBaB$
 $\Rightarrow baaa$

So, $L = \{aa, aaa, baaa, baab, \dots\}$

$L = \{w \mid w \text{ containing atleast one occurrence of two consecutive a's}\}$

Example 17: Write CFG for a language containing string having atleast one occurrence of “00” over {0, 1}. **[April 16]**

Solution: The regular expression for language is

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

So CFG is

$$S \rightarrow ABA$$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

$$B \rightarrow 00$$

OR

$$S \rightarrow A00A$$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

Example 18: Write CFG for generating IF loop of 'C' language.

Solution:

$$\begin{aligned} S &\rightarrow \text{if COND ST else ST} \\ \text{COND} &\rightarrow \text{id RELOP id} \mid \text{id RELOP num} \\ \text{ST} &\rightarrow \text{id} \mid \text{id} = \text{id} \mid \text{id} = \text{num} \\ \text{RELOP} &\rightarrow < \mid < = \mid > \mid > = \mid < > \mid = \\ \text{id} &\rightarrow \text{LX} \\ X &\rightarrow \text{L X} \mid \text{D X} \mid \epsilon \\ \text{L} &\rightarrow \text{A} \mid \text{B} \mid \text{C} \mid \dots \mid \text{Z} \mid \text{a} \mid \text{b} \dots \mid \text{Z} \\ \text{D} &\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9 \\ \text{num} &\rightarrow \text{D Num} \mid \text{D} \end{aligned}$$

Here, statement is “if” loop i.e. if condition statement else statement.
where condition is

identifier relational operator identifier | num

Regular expression for identifier is

$(l)(l/d)^*$

where $l = \text{A to Z or a - z}$
 $d = 0 \text{ to } 9$

CFG for identifier is \rightarrow

$$\begin{aligned} \text{ID} &\rightarrow \text{L X} \\ X &\rightarrow \text{L X} \mid \text{D X} \mid \epsilon \\ \text{L} &\rightarrow \text{A} \mid \text{B} \mid \dots \mid \text{Z} \mid \text{a} \mid \dots \mid \text{Z} \\ \text{D} &\rightarrow 0 \mid \dots \mid 9 \end{aligned}$$

CFG for number is,

$$\text{Num} \rightarrow \text{D Num} \mid \text{D}$$

i.e. at least one digit = (d^+) is regular expression.

Example 19: Construct a grammar for regular expression $(a + b)^*$ over $\{a, b\}$.

Solution: $L = \{\epsilon, a, b, ab, ba, abb, aab, aa, \dots\}$

CFG is $S \rightarrow aA \mid bA \mid \epsilon$

$A \rightarrow bS \mid aS$

Example 20: Construct a grammar which accept a language $(a + b + c)^*$.

Solution: $L = \{\epsilon, a, b, c, ab, bc, ac, aa, bb, cc, abc, \dots\}$

CFG is $G = \{V, T, P, S\}$

where $V = \{S, A\}$ $T = \{a, b, c\}$

P is $S \rightarrow aA \mid bA \mid cA \mid \epsilon$

$S \rightarrow aS \mid bS \mid cS$

Example 21: Construct CFG for language defined by regular expression

$$(abb + baa)^*$$

Solution: $L = \{\epsilon, abb, baa, abbbbaa, \dots\}$

Grammar is $S \rightarrow AS \mid BS \mid \epsilon$

$A \rightarrow abb$

$B \rightarrow baa$

where S is start symbol.

$V = \{A, B\}$

$T = \{a, b\}$

Example 22: Construct CFG for $L = \{0^i, 1^n, 2^n \mid n \geq 1, i \geq 0\}$

[April 16]

Solution: $L = \{12, 012, 0012, 0112, 0122, \dots\}$

$G = (V, T, P, S)$

where P is $S \rightarrow S \mid A$

$A \rightarrow 1A2 \mid 12$

where $V = \{S, A\}$ $T = \{0, 1, 2\}$ $S = \text{Start symbol}$

Example 23: Construct CFG for Do fragment of C.

Solution: $S \rightarrow \text{DO ST while COND}$

$\text{ST} \rightarrow \text{id} = \text{id} \mid \text{num}$

$\text{COND} \rightarrow \text{id rel op id} \mid \text{num}$

$\text{rel op} \rightarrow < \mid < = \mid > \mid > = \mid < > \mid =$

$\text{id} \rightarrow L X$

$X \rightarrow L X \mid D X \mid \epsilon$

$L \rightarrow A \mid B \mid \dots \mid Z \mid a \mid \dots \mid z$

$D \rightarrow 0 \mid 1 \mid \dots \mid 9$

$\text{num} \rightarrow D \text{num} \mid D$

Example 24: Draw the derivation tree for substring 001100 using following grammar

$G = (\{S, A\}, \{0, 1\}, \{S \rightarrow 0AS \mid 0, A \rightarrow S \mid A \mid SS \mid 10\}, S)$

Solution:

$$\begin{aligned}
 S &\Rightarrow 0\underline{A}S \\
 &\Rightarrow 0\underline{S}1AS \\
 &\Rightarrow 001\underline{A}S \\
 &\Rightarrow 00110\underline{S} \\
 &\Rightarrow 001100
 \end{aligned}$$

Derivation tree or Parse tree is,

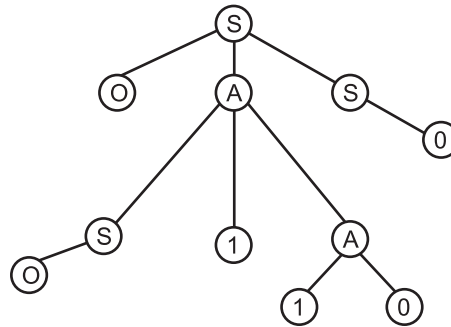


Fig. 3.7

\therefore 001100 is the yield of above language.

3.6 SIMPLIFICATION OF CFG

[Oct. 17]

- In CFG, sometimes all the productions rules and symbols are not needed for the derivation to solve.
- Some productions rules are never used during derivation of any string. Elimination of these types of productions or symbols is called Simplification of Context Free Grammar.
- There are certain standard ways of writing CFG. They impose certain restrictions on the productions in the CFG, so that it is easier to analyze the grammar. One such form is Chomsky Normal Form (CNF).
- Another aspect of grammar is that not always the grammars are optimized i.e. they consist of some extra symbols. Such symbols can be removed from the grammar without changing the language represented by the grammar.
- In some methods, the grammar which provides a reduced form of grammar is called as reduced grammar.
- There are three steps for simplification of CFG:
 1. Elimination of useless symbol.
 2. Elimination of unit production.
 3. Elimination of ϵ -production.

3.6.1 Elimination of Useless Symbols

[Oct. 17]

- The productions in context free grammar that contains useless symbols are called Useless productions. The grammar that we obtain after deleting useless production rules are called reduced Context Free Grammar.
- Let $G = (V, T, P, S)$ be a grammar. A symbol X is useful if there is a derivation $S \xRightarrow{*} \alpha X \beta \Rightarrow w$ for some α, β and w , where $w \in T^*$, otherwise X is useless.

Symbol X is useful if

1. Some string must be derivable from X.
2. 'X' must appear in the derivation of atleast one string derivable from 'S' (start symbol).
3. It should not occur in any sentential form that contains a variable from which no terminal string can be derived.

Example 1: Consider the grammar,

$$S \rightarrow AB \mid a$$

$$A \rightarrow a$$

Solution: Here, variable 'B' is not deriving any string of terminals. Thus, it is an useless symbol. For removing 'B' to get reduced grammar, we should delete all productions for which 'B' is appearing on the right hand side. Hence, we drop only $S \rightarrow AB$ to get simplified grammar without useless symbol as

$$S \rightarrow a$$

$$A \rightarrow a$$

Here to derive a string of terminal through 'A', symbol 'A' cannot occur in any derivation of some string derivable from 'S'. Therefore A is also a useless symbol.

Hence, the simplified grammar without useless symbol is

$$G = (\{S\}, \{a\}, \{S \rightarrow a\}, S)$$

Example 2: Consider a grammar,

$$S \rightarrow AB \mid BC$$

$$A \rightarrow aAa \mid aAb$$

$$B \rightarrow bB \mid b$$

$$D \rightarrow dD \mid d$$

Solution: Consider production $S \rightarrow AB$. As 'A' cannot derive any terminal string, it is a useless symbol. So drop production $S \rightarrow AB$. Similarly, for $S \rightarrow BC$, symbol 'C' does not derive any terminal string, drop $S \rightarrow BC$.

Hence start symbol itself does not derive any string. Grammar is useless.

Example 3: Construct a grammar without useless symbols for the grammar

$$S \rightarrow AB \mid CA$$

$$B \rightarrow BC \mid AB$$

$$A \rightarrow a$$

$$C \rightarrow aB \mid b$$

$$D \rightarrow SS \mid d$$

Solution: Consider $S \rightarrow AB$, here B is useless because it cannot derive any terminal string. Hence remove all productions which contain 'B' in RHS of production.

We rewrite the grammar as,

$$\begin{aligned} S &\rightarrow CA \\ A &\rightarrow a \\ C &\rightarrow b \\ D &\rightarrow SS \mid d \end{aligned}$$

Since, D is not in the derivation of string which derives from the starting symbol S, hence, 'D' is useless.

Thus, resulting grammar without useless symbol is

$$G = (\{S, A, C\}, \{a, b\}, P, S)$$

where P is $\{S \rightarrow CA, C \rightarrow b, A \rightarrow a\}$

3.6.2 Elimination of Unit Productions

[Oct. 17]

- A production of the form $A \rightarrow B$ where A and B both are non-terminals are called unit productions. All other productions including ϵ -productions are non-unit productions.
- **Elimination Rule:** If $A \rightarrow B$ is a unit production or if there is a chain of unit productions leading from A to B such as $A \Rightarrow X_1 \Rightarrow X_2 \Rightarrow \dots \Rightarrow B$ where all X_i 's are non-terminals, then introduce new production(s) according to the following rule.

If the non-unit productions for 'B' are

$$B \rightarrow \alpha_1 \mid \alpha_2 \mid \dots$$

where $\alpha_1, \alpha_2 \dots$ are all sentential forms (not containing only one non-terminal)

then, create the productions for 'A' as

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots$$

Example 1: Consider the grammar,

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow B \mid b \\ B &\rightarrow S \mid a \end{aligned}$$

Solution: As we can see there is a chain of unit productions $S \rightarrow A \rightarrow B \rightarrow S$.

As $A \rightarrow B$ and $B \rightarrow S \mid a$, we can add new productions to A as

$$A \rightarrow S \mid a$$

Hence the grammar becomes,

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow S \mid a \mid b \end{aligned}$$

Still there is a unit production $S \rightarrow A$, removing this we get $S \rightarrow S \mid a \mid b \mid bb$.

There is one more unit production $S \rightarrow S$. We can directly remove as both side symbols are same.

Therefore, the equivalent grammar without unit production will be,

$$S \rightarrow a | b | bb$$

Example 2: Consider the grammar G as,

$$A \rightarrow B$$

$$B \rightarrow a | b$$

Find the equivalent grammar without unit production.

Solution: By rule, $B \rightarrow \alpha_1 | \alpha_2$ where $\alpha_1 = a$ and $\alpha_2 = b$

\therefore Reduced grammar without unit production is $A \rightarrow a | b$.

Example 3: Consider the grammar G as,

$$S \rightarrow Saab | A$$

$$A \rightarrow Sbba | B$$

$$B \rightarrow aS | bS | a | b$$

Solution: First substitute all B-productions in A

$$S \rightarrow Saab | A$$

$$A \rightarrow Sbba | aS | bS | a | b$$

Now substitute all A-productions in S.

$$S \rightarrow Saab | Sbba | aS | bS | a | b$$

This is the grammar without unit production.

3.6.3 Elimination of ϵ -Production

- Production of the form $A \rightarrow \epsilon$ where A is non-terminal is called as ϵ -production.
- Note that if ϵ is in $L(G)$ for grammar G , we cannot eliminate all ϵ -productions from G but ' ϵ ' is not in $L(G)$, then we can.
- **Theorem 1:** Let $G = (V, T, P, S)$ be a CFG then we can find CFG G' having no null production such that $L(G) = L(G') - \{\epsilon\}$.

Construction:

Step 1: Delete all ϵ -productions from grammar.

Step 2: If CFG G_1 there is non-terminal N and production $N \rightarrow \epsilon$ or $N \xRightarrow{*} \epsilon$ then N is called nullable non-terminal.

Identify nullable non-terminal.

Step 3: If there is a production of the form ' $A \rightarrow \alpha$ ' where α is any sentential form containing atleast one nullable non-terminal, then add new productions having right hand side formed by deleting all possible subsets of nullable non-terminals from α .

Step 4: If using step 3, we get production $A \rightarrow \epsilon$ then do not add that to final grammar.

Example 1: Eliminate ϵ -productions from G , where G is $S \rightarrow aSa \mid bSb \mid \epsilon$.

Solution: 1. As step 1, deleting ϵ -production we get,

$$S \rightarrow aSa \mid bSb$$

2. As $S \rightarrow \epsilon$ is production, hence S is nullable.

3. There are two productions containing nullable non-terminal on the RHS, these are $S \rightarrow aSa$ and $S \rightarrow bSb$.

Deleting S from RHS from both productions, we get,

$$S \rightarrow aa \text{ and } S \rightarrow bb, \text{ which we add to the grammar.}$$

Thus, the final grammar without ϵ -productions is

$$S \rightarrow aSa \mid bSb \mid aa \mid bb.$$

Example 2: Eliminate ϵ -productions from grammar G as,

$$S \rightarrow ABA$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

Solution: 1. Deleting ϵ -productions, $A \rightarrow \epsilon$ and $B \rightarrow \epsilon$.

2. A and B both are nullable.

$S \Rightarrow ABA \Rightarrow BA \Rightarrow A \Rightarrow \epsilon$ is also nullable.

3. Find production consisting of nullable symbol on RHS and delete nullable non-terminals from RHS. From $S \rightarrow ABA$ after deleting all possible subsets of nullable non-terminals, we get,

$$S \rightarrow AB$$

$$S \rightarrow BA$$

$$S \rightarrow AA$$

$$S \rightarrow A \mid B \mid \epsilon$$

We can add all other productions except $S \rightarrow \epsilon$ to final set.

From $A \rightarrow aA$, we get $A \rightarrow a$

From $B \rightarrow bB$, we get $B \rightarrow b$

Therefore, the final grammar G' is,

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Here again we have unit production so remove unit production, we get

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

or Substitute A - productions and B - productions in S for all A 's and B 's.

Example 3: Eliminate ϵ -productions from G as $S \rightarrow AB, A \rightarrow SA \mid BB \mid bB, B \rightarrow b \mid aA \mid \epsilon$.

Solution: Set of nullable variables = $\{B, A, S\}$.

The final grammar is,

$$\begin{aligned} S &\rightarrow AB \mid A \mid B \\ A &\rightarrow bB \mid b \\ A &\rightarrow B \\ A &\rightarrow SA \mid S \mid A \\ B &\rightarrow aA \mid a \\ B &\rightarrow b \end{aligned}$$

Example 4: Eliminate ϵ -productions for grammar,

$$\begin{aligned} S &\rightarrow aS \mid bA \mid a \mid b \mid \epsilon \\ A &\rightarrow aS \mid a \mid \epsilon \end{aligned}$$

Solution: 1. Eliminate ϵ -productions we get

$$\begin{aligned} S &\rightarrow aS \mid bA \mid a \mid b \\ A &\rightarrow aS \mid a \end{aligned}$$

2. S and A are nullable symbols.
3. Find productions consisting of nullable symbols on RHS and delete nullable non-terminals.

$$\begin{aligned} S &= aS \mid a \mid bA \mid b \\ A &\rightarrow aS \mid a \end{aligned}$$

This is grammar without ϵ -productions.

3.7 NORMAL FORMS

- A grammar is said to be in a normal form when every production of grammar has specific form.
- In context free grammar, LHS is only one non-terminal, but RHS is any string of non-terminal and terminals. When the productions in G satisfy certain constraints, then G is said to be in “normal form”.
- In this section we will discuss two normal forms:
 1. Chomsky Normal Form (CNF),
 2. Greibach Normal Form (GNF).

3.7.1 Chomsky Normal Form (CNF)

[April 16, 17, 18, 19, Oct. 17, 18]

- In CNF, there are restrictions on the length of right-hand side, and type of symbols is used in right hand side of production rules.

- Any CFL without ϵ is generated by a grammar in which all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B and C are variables and 'a' is a terminal. This type of grammar is said to be in CNF.
- Note that any CFL that does not contain ϵ as a word has a CFG in CNF that generates exactly as it's.
- However, if the CFL contains ϵ , then when we convert the CFG into CNF, the ϵ -word drops out of the language while all other words stay the same.

Example 1: Convert the following CFG into CNF:

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid aa \mid bb$$

Solution: In CNF, we have only two types of productions $A \rightarrow BC$ or $A \rightarrow a$.

We add two productions $A \rightarrow a$ and $B \rightarrow b$ and get,

$$S \rightarrow ASA \mid BSB \mid a \mid b \mid AA \mid BB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Consider $S \rightarrow ASA$ which is not in CNF

$$S \rightarrow D_1A$$

$$D_1 \rightarrow AS$$

Similarly, $S \rightarrow BSB$ which is not in CNF

$$S \rightarrow D_2B$$

$$D_2 \rightarrow BS$$

Thus the equivalent grammar in CNF is

$$S \rightarrow D_1A \mid D_2B \mid a \mid b \mid AA \mid BB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$D_1 \rightarrow AS$$

$$D_2 \rightarrow BS$$

Example 2: Convert the following grammar into CNF:

$$S \rightarrow aAab \mid Aba$$

$$A \rightarrow aS \mid bB$$

$$B \rightarrow ASb \mid a$$

Solution: Thus the required grammar in CNF is

$$S \rightarrow C_a D_1 \mid A D_3$$

$$D_1 \rightarrow A D_2$$

$$D_2 \rightarrow C_a C_b$$

$$\begin{aligned}
D_3 &\rightarrow C_b C_a \\
A &\rightarrow C_a S \mid C_b S \\
B &\rightarrow A D_4 \mid a \\
D_4 &\rightarrow S C_b \\
C_a &\rightarrow a \\
C_b &\rightarrow b
\end{aligned}$$

Example 3: Convert the following grammar into CNF,

$$\begin{aligned}
S &\rightarrow ABA \\
A &\rightarrow aA \mid \epsilon \\
B &\rightarrow bB \mid \epsilon
\end{aligned}$$

Solution: First remove ϵ -productions to get new grammar $L(G) - \{\epsilon\}$. After removing ϵ -productions as per rule, we get new grammar as

$$\begin{aligned}
S &\rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\
A &\rightarrow aA \mid a \\
B &\rightarrow bB \mid b
\end{aligned}$$

Then by eliminating unit-productions $S \rightarrow A$ and $S \rightarrow B$, we get grammar as,

$$\begin{aligned}
S &\rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b \\
A &\rightarrow aA \mid a \\
B &\rightarrow bB \mid b
\end{aligned}$$

Now, convert the grammar into CNF.

So final grammar in CNF will be as follows:

$$\begin{aligned}
S &\rightarrow D_1 A \mid AB \mid BA \mid AA \mid C_a A \mid a \mid C_b B \mid b \\
A &\rightarrow C_a A \mid a \\
B &\rightarrow C_b B \mid b \\
D_1 &\rightarrow AB \\
C_a &\rightarrow a \\
C_b &\rightarrow b
\end{aligned}$$

Example 4: Convert the following grammar into CNF,

$$\begin{aligned}
S &\rightarrow bA \mid aB \\
A &\rightarrow bAA \mid aS \mid a \\
B &\rightarrow aBB \mid bS \mid b
\end{aligned}$$

Solution: Substitute X for a and Y for b.

$$\begin{aligned} S &\rightarrow YA \mid XB \\ A &\rightarrow YAA \mid XS \mid a \\ B &\rightarrow XBB \mid YS \mid b \\ X &\rightarrow a \\ Y &\rightarrow b \end{aligned}$$

1. All s-productions are in CNF.
2. Consider A-production

$$A \rightarrow YAA$$

Replace AA by P

$$A \rightarrow YP$$

$$P \rightarrow AA$$

3. Consider B $\rightarrow XBB$

Replace BB by R

$$B \rightarrow XR$$

$$R \rightarrow BB$$

\therefore Equivalent CNF grammar is

$$\begin{aligned} S &\rightarrow YA \mid XB \\ A &\rightarrow YP \mid XS \mid a \\ B &\rightarrow XR \mid YS \mid b \\ X &\rightarrow a \\ Y &\rightarrow b \\ P &\rightarrow AA \\ R &\rightarrow BB \end{aligned}$$

Example 5: Convert the following CFG into CNF.

$$\begin{aligned} S &\rightarrow ABA \\ A &\rightarrow aA \mid \epsilon \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

Solution: The grammar contains ϵ -production, we have to eliminate ϵ -production. Since CNF is always represent language without ϵ .

After eliminating ϵ -productions, we get

$$\begin{aligned} S &\rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

This grammar contains unit productions, so remove unit productions, we get

$$\begin{aligned} S &\rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

Now grammar is simplified grammar without ϵ -productions and unit productions, so convert into equivalent CNF.

$$\begin{aligned} S &\rightarrow \underline{ABA} \\ \text{is } S &\rightarrow AX \\ X &\rightarrow BA \end{aligned}$$

Equivalent CNF grammar is

$$\begin{aligned} S &\rightarrow AX \mid AB \mid BA \mid AA \mid D_1 A \mid a \mid D_2 B \mid b \\ A &\rightarrow D_1 A \mid a \\ B &\rightarrow D_2 B \mid b \\ X &\rightarrow BA \\ D_1 &\rightarrow a \\ D_2 &\rightarrow b \end{aligned}$$

3.7.2 Greibach Normal Form (GNF)

[Oct. 16, 17, 18, April 17, 19]

- In GNF, there is restriction on the position, in which, terminals and variables can appear on right-hand side of production rules.
- In GNF, every production must start with a single terminal followed by any number of variables.
- In every CFL, L without ϵ -production can be generated by a grammar for which every production is of the form $A \rightarrow a\alpha$, where 'A' is a variable, 'a' is a terminal and α is a string of only variables. This type of grammar is said to be GNF.
- **Lemma 1:** The production with variable A on left is called as A-production.
Let $G = (V, T, P, S)$ be a CFG. Let $A \rightarrow \alpha_1\beta\alpha_2$ be a production in P and $B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_r$ be the set of all B-productions. Let $G_1 = (V, T, P_1, S)$ be obtained by deleting production $A \rightarrow \alpha_1\beta\alpha_2$ from P and adding productions $A \rightarrow \alpha_1\beta_1\alpha_2 \mid \alpha_1\beta_2\alpha_2 \mid \dots \mid \alpha_1\beta_r\alpha_2$. Then $L(G) = L(G_1)$.
- **Lemma 2:** Let $G = (V, T, P, S)$ be a CFG. Let $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_r$ be the set of A-productions for which A is the leftmost symbol of the RHS. Let $A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_s$ be the remaining A-productions. Let $G_1 = (V \cup \{B\}, T, P_1, S)$ be the CFG formed by adding the variable B to V and replacing all the A-productions by the productions:

$$\begin{aligned} 1. \quad &A \rightarrow \beta_i \\ &A \rightarrow \beta_i B \quad 1 \leq i \leq s \end{aligned}$$

$$2. \quad B \rightarrow \alpha i$$

$$1 \leq i \leq r$$

$$B \rightarrow \alpha i B$$

Then, $L(G) = L(G_1)$

Note : Grammar should be in CNF before converting into GNF.

Example 1: Construct a grammar in GNF equivalent to grammar

$$S \rightarrow AA \mid a$$

$$A \rightarrow SS \mid b$$

Solution: Observe that the CFG is in CNF. If we rename S as A_1 and A as A_2 respectively, then productions will be

$$A_1 \rightarrow A_2 A_2 \mid a$$

$$A_2 \rightarrow A_1 A_1 \mid b$$

We leave $A_2 \rightarrow b$ as it is in the required form.

Now, consider $A_2 \rightarrow A_1 A_1$. To convert this we will use lemma 4.1, and get,

$$A_2 \rightarrow A_2 A_2 A_1 \quad \text{By replacing the first } A_1 \text{ on}$$

$$A_2 \rightarrow a A_1 \quad \text{RHS of } A_2 \rightarrow A_1 A_1 \text{ by definition of } A_1$$

(**Note :** Here we have been considering production $A_2 \rightarrow A_1 A_1$ because production is in the form $A_i \rightarrow A_j \alpha$, $j < i$).

Now the production $A_2 \rightarrow a A_1$ is in the required form.

But we need lemma for $A_2 \rightarrow A_2 A_2 A_1$ as it is in the form $A \rightarrow A \alpha$.

Applying lemma to productions of A_2 , A_2 productions are,

$$A_2 \rightarrow A_2 A_2 A_1 \mid a A_1 \mid b$$

Here, $\beta_1 = a A_1$, $\beta_2 = b$, $\alpha = A_2 A_1$

\therefore Adding new non-terminal B_2 , we get,

$$A_2 \rightarrow a A_1 \mid b$$

$$A_2 \rightarrow a A_1 B_2 \mid b B_2$$

$$B_2 \rightarrow A_2 A_1$$

$$B_2 \rightarrow A_2 A_1 B_2$$

Now, all A_2 productions are in the required form.

Now we will have to consider A_1 production,

$$A_1 \rightarrow A_2 A_2 \mid a$$

Applying lemma 4.2 i.e. replacing all A_2 productions on RHS, we get,

$$A_1 \rightarrow a A_1 A_2 \mid b A_2 \mid a A_1 B_2 A_2 \mid b B_2 A_2 \mid b$$

Now, modify B_2 production by replacing A_2 production on RHS.

Consider $B_2 \rightarrow A_2 A_1 \mid A_2 A_1 B_2$.

Both RHS A_2 are replaced by A_2 productions.

$$\begin{aligned} \therefore B_2 &\rightarrow aA_1A_1 \mid bA_1 \mid aA_1B_2A_1 \mid bB_2A_1 \\ B_2 &\rightarrow aA_1A_1B_2 \mid bA_1B_2 \mid aA_1B_2A_1B_2 \mid bA_1B_2 \end{aligned}$$

So the grammar in GNF will be

$$G' = (\{A_1, A_2, B_2\}, \{a, b\}, P', A_1)$$

where P is

$$\begin{aligned} \{A_1 &\rightarrow a \mid aA_1A_2 \mid bA_2 \mid aA_1B_2A_2 \mid bB_2A_2 \\ A_2 &\rightarrow aA_1 \mid b \mid aA_1B_2 \mid bB_2 \\ B_2 &\rightarrow aA_1A_1 \mid bA_1 \mid aA_1B_2A_1 \mid bB_2A_1 \\ B_2 &\rightarrow aA_1A_1B_2 \mid bA_1B_2 \mid aA_1B_2A_1B_2 \mid bB_2A_1B_2\} \end{aligned}$$

Example 2: Convert the following CFG into GNF:

$$\begin{aligned} S &\rightarrow AB \mid B \\ A &\rightarrow BS \\ B &\rightarrow A1 \mid 1 \end{aligned}$$

Solution: Above grammar contains unit production. So first eliminate it, and we get,

$$\begin{aligned} S &\rightarrow AB \mid A1 \mid 1 \\ A &\rightarrow BS \\ B &\rightarrow A1 \mid 1 \end{aligned}$$

This grammar is not in CNF. Converting into CNF, we get,

$$\begin{aligned} S &\rightarrow AB \mid AC_1 \mid 1 \\ A &\rightarrow BS \\ B &\rightarrow AC_1 \mid 1 \\ C_1 &\rightarrow 1 \end{aligned}$$

Replacing variable S by A_1 , A by A_2 , B by A_3 and C_1 by A_4 , we get,

$$\begin{aligned} A_1 &\rightarrow A_2A_3 \mid A_2A_4 \mid 1 \\ A_2 &\rightarrow A_3A_1 \\ A_3 &\rightarrow A_2A_4 \mid 1 \\ A_4 &\rightarrow 1 \end{aligned}$$

Only $A_3 \rightarrow A_2A_4$ is not in the required form $A_i \rightarrow A_j \alpha, j > i$.

\therefore Apply lemma i.e. substitute all A_2 productions in A_3 productions.

We get, $A_3 \rightarrow A_3 A_1 A_4 | 1$

Now, applying lemma 4.2, we get,

$$A_3 \rightarrow 1 | 1 B_1$$

$$B_1 \rightarrow A_1 A_4 | A_1 A_4 B_1$$

Now convert all productions in GNF.

A_3 and A_4 productions are already in GNF.

Applying lemma 4.1 on A_2 productions we get,

$$A_2 \rightarrow 1 A_1 | 1 B A_1$$

Applying lemma 4.2 on A_1 productions, we get,

$$A_1 \rightarrow 1 A_1 A_3 | 1 B_1 A_1 A_3 | 1 A_1 A_4 | 1 B_1 A_1 A_4 | 1$$

We get B_1 productions as

$$\begin{aligned} B &\rightarrow 1 A_1 A_3 A_4 | 1 B_1 A_1 A_3 A_4 | 1 A_1 A_4 A_4 | 1 B_1 A_1 A_4 A_4 | 1 A_4 | \\ &\quad 1 A_1 A_4 A_4 B_1 | 1 B_1 A_1 A_4 A_4 B_1 | 1 A_1 A_3 A_4 B_1 | 1 B_1 A_1 A_3 A_4 B_1 | \\ &\quad 1 A_4 B_1 \end{aligned}$$

Thus the required grammar in GNF is

$$A_1 \rightarrow 1 A_1 A_3 | 1 B_1 A_1 A_3 | 1 A_1 A_4 | 1 B_1 A_1 A_4 | 1$$

$$A_2 \rightarrow 1 A_1 | 1 B A_1$$

$$A_3 \rightarrow 1 | 1 B$$

$$A_4 \rightarrow 1$$

$$\begin{aligned} B_1 &\rightarrow 1 A_4 A_3 A_4 | 1 B_1 A_1 A_3 A_4 | 1 A_1 A_4 A_4 | 1 B_1 A_1 A_3 A_4 B_1 | \\ &\quad 1 A_4 | 1 A_1 A_4 A_4 B_1 | 1 B_1 A_1 A_4 A_4 B_1 | 1 A_1 A_3 A_4 B_1 | \\ &\quad 1 B_1 A_1 A_3 A_4 B_1 | 1 A_4 B_1 \end{aligned}$$

Example 3: Convert the following grammar G to GNF:

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 | b$$

$$A_3 \rightarrow A_1 A_2 | a$$

Solution: Given grammar is in CNF.

Only the production $A_3 \rightarrow A_1 A_2$ is not in the required form.

Since $j < i$ (i.e. $1 < 3$).

Applying lemma 4.1 on A_3 production, we get,

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 | b$$

$$A_3 \rightarrow A_2 A_3 A_2 | a$$

Again substitute 'A₂' on RHS of A₃ with A₂ productions, and we get,

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$$

Now, applying lemma 4.2, we get,

$$A_3 \rightarrow b A_3 A_2 \mid a$$

$$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3$$

$$B_3 \rightarrow A_1 A_3 A_2 \mid A_1 A_3 A_2 B_3$$

All A₃ productions are in GNF form.

Therefore, substitute A₃ productions in A₂ productions, and we get,

$$A_2 \rightarrow b A_3 A_2 A_1 \mid b A_3 A_2 B_3 A_1 \mid a B_3 A_1 \mid a A_1 \mid b$$

Now, substitute A₂ productions in A₁ productions, and we get,

$$A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3 \mid a B_3 A_1 A_3 \mid b A_3 A_2 A_1 A_3 \mid a A_1 A_3 \mid b A_3$$

Now, substitute all A₁ productions in B₃ productions.

An equivalent grammar in GNF can be written as,

$$A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3 \mid a B_3 A_1 A_3 \mid b A_3 A_2 A_1 A_3 \mid a A_1 A_3 \mid b A_3$$

$$A_2 \rightarrow b A_3 A_2 B_3 A_1 \mid a B_3 A_1 \mid b A_3 A_2 A_1 \mid a A_1 \mid b$$

$$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$$

$$B_3 \rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2 \mid b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3 \mid a B_3 A_1 A_3 A_3 A_2 \mid$$

$$a B_3 A_1 A_3 A_3 A_2 B_3 \mid b A_3 A_2 A_1 A_3 A_2 \mid b A_3 A_2 A_1 A_3 A_3 A_2 B_3 \mid$$

$$a A_1 A_3 A_3 A_2 \mid a A_1 A_3 A_3 A_2 B_3 \mid b A_3 A_3 A_2 \mid b A_3 A_3 A_2 B_3$$

Example 4: Convert the following grammar into GNF.

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Solution: In previous example we have already simplified this grammar as

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid aA \mid a \mid bB \mid b$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Substitute A and B-productions, we get

$$S \rightarrow aABA \mid aBA \mid aAB \mid aB \mid bBA \mid bA \mid aAA \mid aA \mid a \mid bB \mid b$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

3.8 REGULAR GRAMMAR

[April 16, Oct. 18]

- Regular grammar is a formal grammar, in which every production is restricted to have one of the following forms:
 - $A \rightarrow aB$
 - $A \rightarrow a$, where A and B are the non-terminals, a is a terminal symbol.

With ϵ -productions permitted as special case $L(G)$ contain ϵ .

- It is called a regular grammar, because if the format of every production in CFG is restricted to $A \rightarrow aB$ or $A \rightarrow a$, then the grammar can specify only regular sets and hence there exists a finite automata accepting $L(G)$, if G is regular.
- Definition:** A regular grammar is a formal grammar (N, Σ, P, S) where the rules of P are of the following forms:
 - $A \rightarrow a$, where A is a non-terminal in N and a is terminal in Σ .
 - $A \rightarrow \epsilon$, where A is N and ϵ is the empty string and either.
 - $A \rightarrow Ba$, where A and B are in N and a is in Σ . In this case, the grammar is a left regular grammar of the form.
 - $A \rightarrow aB$, where A and B are in N and a is in Σ . In this case, the grammar is a right regular grammar.

Example:

$$S \rightarrow aB \mid \epsilon$$

$$B \rightarrow Ba \mid \epsilon$$

3.9 LEFT LINEAR AND RIGHT LINEAR GRAMMARS [April 16, 18]

- Regular grammar is classified as Left-linear grammar and Right-linear grammar.
 - Left-linear grammar (Definition):** If all productions of a CFG are of the form $A \rightarrow Bw$, or $A \rightarrow w$ or $A \rightarrow \epsilon$ where A, B are variables and w is a string of terminals then we say that grammar is a left-linear grammar.
 - Right linear grammar (Definition):** If all productions of a CFG are of the form $A \rightarrow wB$ or $A \rightarrow w$ or $A \rightarrow \epsilon$ where A, B and w are same as above then grammar is called a right-linear grammar.
- A right and left linear grammar is called a regular grammar. A grammar is called linear grammar in which at most one non-terminal can occur on the right side of any production rule.

Example 1: The language $0(10)^*$ is generated by the right-linear grammar $S \rightarrow 0A$, $A \rightarrow 10A \mid \epsilon$ and left-linear grammar $S \rightarrow S10 \mid 0$.

Example 2:

$$A \rightarrow a, A \rightarrow aB, A \rightarrow \epsilon$$

where, A and B are non-terminals and a is terminal.

Example 3:

$$S \rightarrow 00A \mid 11S$$

$$A \rightarrow 0A \mid 1A \mid 0 \mid 1$$

where, S and A are non-terminals and 0 and 1 are terminals.

Left-linear grammar examples:

$$A \rightarrow a, A \rightarrow Ba, A \rightarrow \epsilon$$

where,

A and B are non-terminals, a is terminal and ϵ is empty string.

$$S \rightarrow A00 \mid S11$$

$$A \rightarrow A0 \mid A1 \mid 0 \mid 1$$

where, S and A are non-terminals and 0 and 1 are terminals.

3.10 EQUIVALENCE OF REGULAR GRAMMAR AND FA

- The equivalence exists between regular grammar and finite automata in accepting languages.

3.10.1 Construction of Regular Grammar from the DFA

- If L is a regular set then L is generated by regular grammar.
- Proof:** Let $L = L(M)$ for DFA $M = (Q, \Sigma, \delta, q_0, F)$. Let $q_0 \notin F$ then $L = L(G)$ for right linear grammar. $G = (Q, \Sigma, p, q_0)$ where p is set of productions, is constructed as follows :

- If $\delta(q_i, a) = q_j$ and $q_j \notin F$ then add production $q_i \rightarrow a q_j$ in p.
- If $\delta(q_i, a) = q_j$ and $q_j \in F$ then add productions $q_i \rightarrow a q_j \mid a$ in p.

So $\delta(q_i, w) = q_j$ iff $q_i \xRightarrow{*} wq_j$.

- Here we had assumed that $q_0 \notin F$.

If $q_0 \in F$, then ϵ is accepted by L (M) then we add production $S \rightarrow q_0 \mid \epsilon$ in p.

Example 1: Construct regular grammar for DFA shown in the Fig. 3.8.

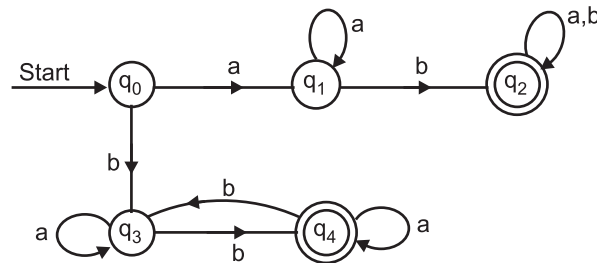


Fig. 3.8

Solution: $\delta(q_0, a) = q_1$ and $q_1 \notin F$ gives $q_0 \rightarrow a q_1$
 $\delta(q_0, b) = q_3$ and $q_3 \notin F$ gives $q_0 \rightarrow b q_3$
 $\delta(q_1, a) = q_1$ and $q_1 \notin F$ gives $q_1 \rightarrow a q_1$
 $\delta(q_1, b) = q_2$ and $q_2 \in F$ gives $q_1 \rightarrow b q_2 | b$
 $\delta(q_2, a) = q_2$ and $q_2 \in F$ gives $q_2 \rightarrow a q_2 | a$
 $\delta(q_2, b) = q_2$ and $q_2 \in F$ gives $q_2 \rightarrow b q_2 | b$
 $\delta(q_3, a) = q_3$ and $q_3 \notin F$ gives $q_3 \rightarrow a q_3$
 $\delta(q_3, b) = q_4$ and $q_4 \in F$ gives $q_3 \rightarrow b q_4 | b$
 $\delta(q_4, a) = q_4$ and $q_4 \in F$ gives $q_4 \rightarrow a q_4 | a$
 $\delta(q_4, b) = q_3$ and $q_3 \notin F$ gives $q_4 \rightarrow b q_3$

\therefore Equivalent regular grammar is,

$$G = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, p, q_0)$$

where p is

$$\begin{aligned} q_0 &\rightarrow a q_1 | b q_3 \\ q_1 &\rightarrow a q_1 | b q_2 | b \\ q_2 &\rightarrow a q_2 | b q_2 | a | b \\ q_3 &\rightarrow a q_3 | b q_4 | b \\ q_4 &\rightarrow a q_4 | b q_3 | a \end{aligned}$$

Example 2: Construct regular grammar for DFA shown in the Fig. 3.9.

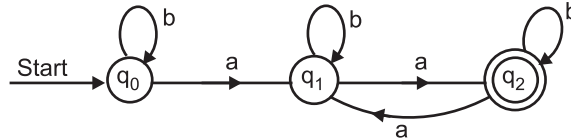


Fig. 3.9

Solution: Equivalent regular grammar is

$$G = (\{q_0, q_1, q_2\}, \{a, b\}, p, q_0)$$

where p is

$$\begin{aligned} q_0 &\rightarrow b q_0 | a q_1 \\ q_1 &\rightarrow b q_1 | a q_2 | a \\ q_2 &\rightarrow a q_1 | b q_2 | b \end{aligned}$$

Example 3: Construct regular grammar for the DFA shown in the Fig. 3.10.

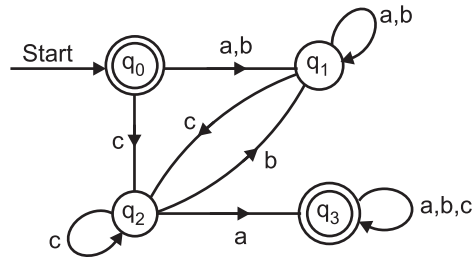


Fig. 3.10

Solution: Equivalent regular grammar is

$$G = (\{S, q_0, q_1, q_2, q_3\}, \{a, b, c\}, P, S)$$

where P is $\{S \rightarrow q_0 \mid \epsilon$

$$q_0 \rightarrow a q_1 \mid b q_1 \mid c q_2$$

$$q_1 \rightarrow a q_1 \mid b q_1 \mid c q_2$$

$$q_2 \rightarrow a q_3 \mid b q_1 \mid c q_2 \mid a$$

$$q_3 \rightarrow a q_3 \mid b q_3 \mid c q_3 \mid a \mid b \mid c\}$$

3.10.2 Construction of FA from the given Right Linear Grammar

- Let $G = (V, T, P, S)$ be a regular grammar. We will construct a FA (NFA) with ϵ -transitions $M = (Q, T, \delta, [S], \{[\epsilon]\})$, where
 - Q consists of the symbol $[\alpha]$ such that α is a variable or a suffix of some RHS of a production in 'P' and
 - ' δ ' is defined as,
 - If A is variable then $\delta([A], \epsilon) = \{[\alpha] \mid A \rightarrow \alpha \text{ is a production}\}$
 - If $a \in T$ and α in $(T^* \cup T^* V)$ then $\delta([a\alpha], a) = \{[\alpha]\}$

Example 1: Construct FA for following regular grammar:

$$S \rightarrow aA \mid bB$$

$$A \rightarrow aS \mid a$$

$$B \rightarrow bS \mid b$$

Solution: First we start with $[S]$. As there are two productions from S we get,

$$\delta([S], \epsilon) = \{[aA], [bB]\}$$

$$\delta([aA], a) = \{[A]\} \text{ and}$$

$$\delta([bB], b) = \{[B]\}$$

$$\delta([A], \epsilon) = \{[aS], [a]\}$$

$$\delta([B], \epsilon) = \{[bS], [b]\}$$

$$\delta([aS], a) = \{[S]\}$$

$$\delta([a], a) = \{[\epsilon]\}$$

$$\delta([bS], b) = \{[S]\}$$

$$\delta([b], b) = \{[\epsilon]\}$$

Thus FA is shown in the Fig. 3.11.

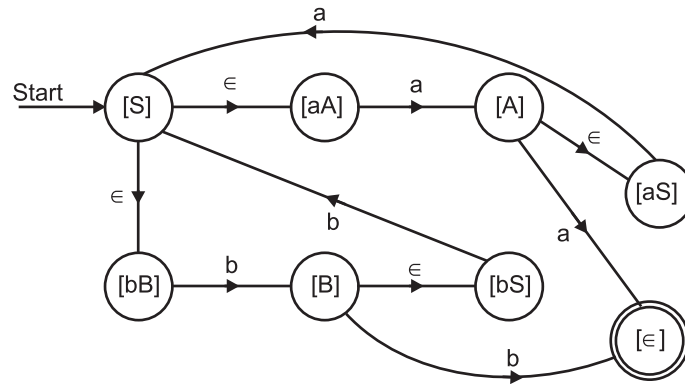


Fig. 3.11

Example 2: Construct FA for following regular 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: FA is shown in the Fig. 3.12.

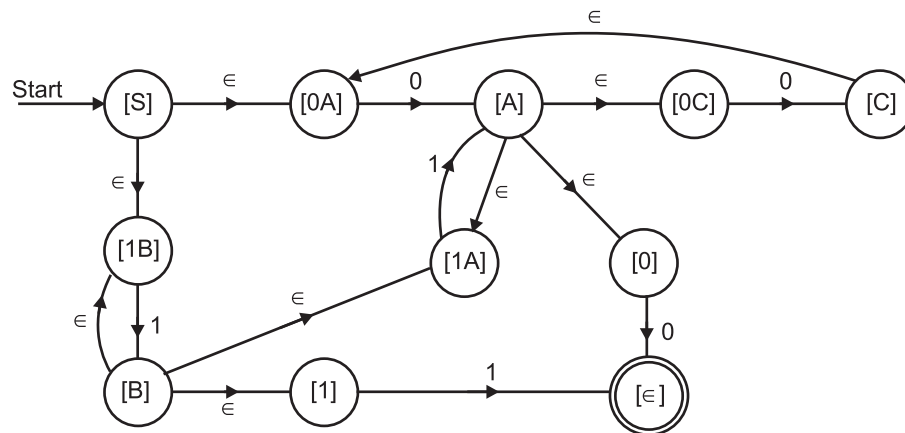


Fig. 3.12

Example 3: Construct NFA for following regular grammar:

$$S \rightarrow bB$$

$$b \rightarrow bC \mid aB \mid b$$

$$C \rightarrow a$$

Solution: Equivalent NFA with ϵ -transition is shown in the Fig. 3.13.

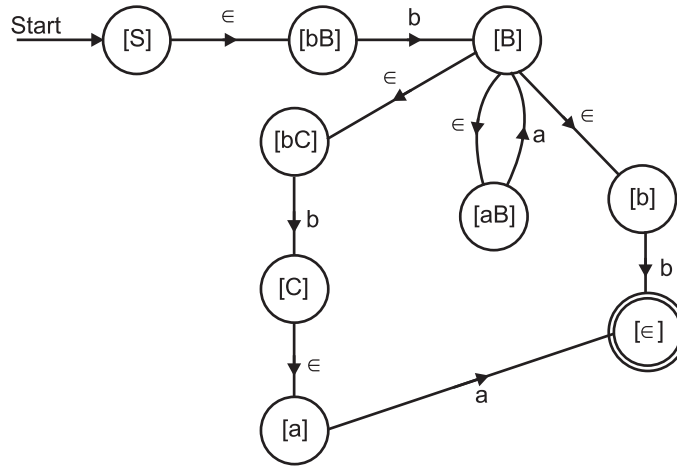


Fig. 3.13

EXAMPLES

Example 1: Construct regular grammar for a language over $\{a, b\}$ consisting of all strings starting with a and having ba as a substring in it.

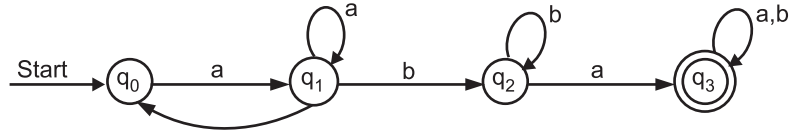


Fig. 3.14

Solution:

- | | | | |
|-------|-------------------------|----------------|-------------------------------|
| (i) | $\delta(q_0, a) = q_1$ | $q_1 \notin F$ | $q_0 \rightarrow aq_1$ |
| | $\delta(q_0, b) = \phi$ | | |
| (ii) | $\delta(q_1, a) = q_1$ | $q_1 \notin F$ | $q_1 \rightarrow aq_1$ |
| | $\delta(q_1, b) = q_2$ | $q_2 \notin F$ | $q_1 \rightarrow bq_2$ |
| (iii) | $\delta(q_2, a) = q_3$ | $q_3 \in F$ | $q_2 \rightarrow aq_3 \mid a$ |
| | $\delta(q_2, b) = q_2$ | $q_2 \notin F$ | $q_2 \rightarrow bq_2$ |
| (iv) | $\delta(q_3, a) = q_3$ | $q_3 \in F$ | $q_3 \rightarrow aq_3 \mid a$ |
| | $\delta(q_3, b) = q_3$ | $q_3 \in F$ | $q_3 \rightarrow bq_3 \mid b$ |
- $\therefore G = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, p, q_0)$

P is:

$$\begin{aligned}
 q_0 &\rightarrow aq_1 \\
 q_1 &\rightarrow aq_1 \mid bq_2 \\
 q_2 &\rightarrow aq_3 \mid bq_2 \mid a \\
 q_3 &\rightarrow aq_3 \mid bq_3 \mid a \mid b
 \end{aligned}$$

Example 2: Construct regular grammar for a language over $\{0, 1\}$. That starts with 00 and ends with 1 having a substring 10 in it.

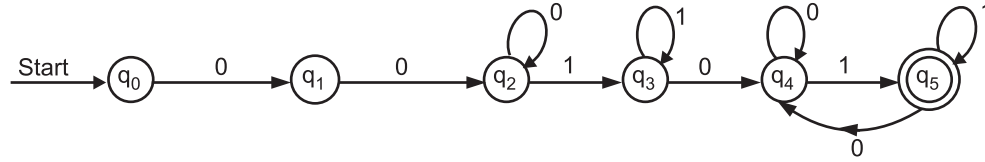


Fig. 3.15

Solution:

- | | | | |
|--------------|--|----------------|-------------------------------|
| (i) | $\delta(q_0, 0) = q_1$ | $q_1 \notin F$ | $q_0 \rightarrow 0q_1$ |
| | $\delta(q_0, 1) = \phi$ | | |
| (ii) | $\delta(q_1, 0) = q_2$ | $q_2 \notin F$ | $q_1 \rightarrow 0q_2$ |
| | $\delta(q_1, 1) = \phi$ | | |
| (iii) | $\delta(q_2, 0) = q_2$ | $q_2 \notin F$ | $q_2 \rightarrow 0q_2$ |
| | $\delta(q_2, 1) = q_3$ | $q_3 \notin F$ | $q_2 \rightarrow 1q_3$ |
| (iv) | $\delta(q_3, 0) = q_4$ | $q_4 \notin F$ | $q_3 \rightarrow 0q_4$ |
| | $\delta(q_3, 1) = q_3$ | $q_3 \notin F$ | $q_3 \rightarrow 1q_3$ |
| (v) | $\delta(q_4, 0) = q_4$ | $q_4 \notin F$ | $q_4 \rightarrow 0q_4$ |
| | $\delta(q_4, 1) = q_5$ | $q_5 \in F$ | $q_4 \rightarrow 1q_5 \mid 1$ |
| (vi) | $\delta(q_5, 0) = q_4$ | $q_4 \notin F$ | $q_5 \rightarrow 0q_4$ |
| | $\delta(q_5, 1) = q_5$ | $q_5 \in F$ | $q_5 \rightarrow q_5 \mid 1$ |
| \therefore | $G = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, P, q_0)$ | | |
| \therefore | P is: | | |

- $$\begin{aligned}
 q_0 &\rightarrow 0q_1 \\
 q_1 &\rightarrow 0q_2 \\
 q_2 &\rightarrow 0q_2 \mid 1q_3 \\
 q_3 &\rightarrow 0q_4 \mid 1q_3 \\
 q_4 &\rightarrow 0q_4 \mid 1q_5 \mid 1 \\
 q_5 &\rightarrow 0q_4 \mid 1q_5 \mid 1
 \end{aligned}$$

Example 3: Construct regular grammar for $L = L_1 \cap L_2$.

L_1 = All strings over $\{a, b, c\}$ having equal no. of a's and c's

$L_2 = \{a^n b c^n \mid n \geq 0 \text{ and } n \leq 5\}$

- (vii) $\delta(q_6, a) = \phi$
 $\delta(q_6, b) = \phi$
 $\delta(q_6, c) = q_3 \quad q_3 \in F \quad q_6 \rightarrow cq_3$
- (viii) $\delta(q_7, a) = q_{10} \quad q_{10} \notin F \quad q_7 \rightarrow aq_{10}$
 $\delta(q_7, b) = q_8 \quad q_8 \notin F \quad q_7 \rightarrow bq_8$
 $\delta(q_7, c) = \phi$
- (ix) $\delta(q_8, a) = \phi$
 $\delta(q_8, b) = \phi$
 $\delta(q_8, c) = q_9 \quad q_9 \notin F \quad q_8 \rightarrow cq_9$
- (x) $\delta(q_9, a) = \phi$
 $\delta(q_9, b) = \phi$
 $\delta(q_9, c) = q_6 \quad q_6 \notin F \quad q_9 \rightarrow cq_6$

Similarly, we get,

$$\begin{aligned} q_{10} &\rightarrow aq_{13} \mid bq_{11} \\ q_{11} &\rightarrow cq_{12} \\ q_{12} &\rightarrow cq_9 \\ q_{13} &\rightarrow bq_{14} \\ q_{14} &\rightarrow cq_{15} \\ q_{15} &\rightarrow cq_{12} \end{aligned}$$

$\therefore G = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}\}, \{a, b, c\}, P, q_0)$
 P is:

$$\begin{aligned} q_0 &\rightarrow aq_1 \\ q_1 &\rightarrow aq_4 \mid bq_2 \\ q_2 &\rightarrow cq_3 \mid c \\ q_4 &\rightarrow aq_7 \mid bq_5 \\ q_5 &\rightarrow cq_6 \\ q_6 &\rightarrow cq_3 \\ q_7 &\rightarrow aq_{10} \mid bq_8 \\ q_8 &\rightarrow cq_9 \\ q_9 &\rightarrow cq_6 \\ q_{10} &\rightarrow aq_{13} \mid bq_{11} \end{aligned}$$

$$q_{11} \rightarrow cq_{12}$$

$$q_{12} \rightarrow cq_9$$

$$q_{13} \rightarrow bq_{14}$$

$$q_{14} \rightarrow cq_{15}$$

$$q_{15} \rightarrow cq_{12}$$

Example 4: Construct a regular grammar for a language over $\{a, b, c\}$ starting with a and having odd no. of b's.

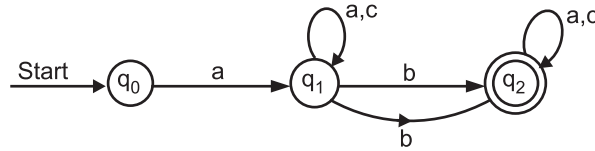


Fig. 3.17

- | | | | |
|-------|-------------------------|----------------|-------------------------------|
| (i) | $\delta(q_0, a) = q_1$ | $q_1 \notin F$ | $q_0 \rightarrow aq_1$ |
| | $\delta(q_0, b) = \phi$ | | |
| | $\delta(q_0, c) = \phi$ | | |
| (ii) | $\delta(q_1, a) = q_1$ | $q_1 \notin F$ | $q_1 \rightarrow aq_1$ |
| | $\delta(q_1, b) = q_2$ | $q_2 \in F$ | $q_1 \rightarrow bq_2 \mid b$ |
| | $\delta(q_1, c) = q_1$ | $q_1 \notin F$ | $q_1 \rightarrow cq_1$ |
| (iii) | $\delta(q_2, a) = q_2$ | $q_2 \in F$ | $q_2 \rightarrow aq_2 \mid a$ |
| | $\delta(q_2, b) = q_1$ | $q_1 \notin F$ | $q_2 \rightarrow bq_1$ |
| | $\delta(q_2, c) = q_2$ | $q_2 \in F$ | $q_2 \rightarrow cq_2 \mid c$ |

The grammar is: $G = (\{q_0, q_1, q_2\}, \{a, b, c\}, P, q_0)$

P is:

$$\begin{aligned} q_0 &\rightarrow aq_1 \\ q_1 &\rightarrow aq_1 \mid bq_2 \mid cq_1 \mid b \\ q_2 &\rightarrow aq_2 \mid bq_2 \mid cq_2 \mid a \mid c \end{aligned}$$

Example 5: Construct regular grammar for the language $\{a^{2n} \mid n \geq 1\}$.

Solution: Explanation.

When $n = 1$

$$a^{2n} \rightarrow a^{2 \times 1} = a^2 = aa$$

When $n = 2$

$$a^{2 \times 2} = a^4 = aaaa$$

When $n = 3$
 $a^{2 \times 3} = a^6 = a^6 = aaaaaa$

\therefore We draw DFA for which accepts only even number of a's.

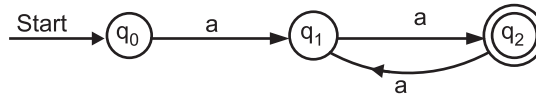


Fig. 3.18

\therefore Regular grammar is

$$\begin{aligned} q_0 &\rightarrow aq_1 \\ q_1 &\rightarrow aq_2 \mid a \\ q_2 &\rightarrow aq_1 \end{aligned}$$

EXAMPLES

Example 1: Define nullable symbol.

Solution: If N is any non-terminal in CFG and $N \rightarrow^* \epsilon$ or $N \in \epsilon$, then N is nullable.

Example 2: What is the yield of following derivation tree?

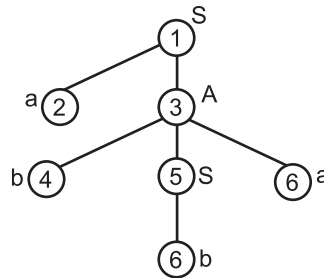


Fig. 3.19

Solution: The yield of derivation tree is abba.

Example 3: Convert the following grammar into GNF.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BS \mid b \\ B &\rightarrow SA \mid a \end{aligned}$$

Solution: Substituting S-productions in B, we get,

$$B \rightarrow ABA \mid a$$

Substitute A-productions in B.

$$B \rightarrow BSBA \mid bBA \mid a$$

Grammar becomes

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BS \mid b \\ B &\rightarrow BSBA \mid bBA \mid a \end{aligned}$$

Apply lemma (2) for B-productions

$$B \rightarrow BSBA \mid bBA \mid a \\ \alpha \beta_1 \beta_2$$

we get $B \rightarrow bBAB' \mid aB' \mid bBA \mid a$

$$B' \rightarrow SBAB' \mid SBA$$

Now all B-productions are in GNF. Substitute all B-productions in A-production.

$$A \rightarrow BS \mid b$$

$$\therefore A \rightarrow bBAB's \mid aB's \mid bBAS \mid aS \mid b$$

Now substitute all A-productions in S.

$$S \rightarrow AB$$

$$S \rightarrow bBAB' SB \mid aB' SB \mid bBASB \mid aSB \mid bB$$

Now substitute all S-productions in B'.

$$B' \rightarrow bBAB' SBBAB' \mid aB' SBBAB' \mid bBASBBAB' \mid aSBBAB' \mid bBBAB'$$

$$B' \rightarrow bBAB'SBBA \mid aB'SBBA \mid bBASBBA \mid aSBBA \mid bBBA$$

\therefore Equivalent grammar in GNF is

$$S \rightarrow bBAB'SB \mid aB'SB \mid bBASB \mid aSB \mid bB$$

$$A \rightarrow bBAB'S \mid aB'S \mid bBAS \mid aS \mid b$$

$$B \rightarrow bBAB' \mid aB' \mid bBA \mid a$$

$$B' \rightarrow bBAB'SBBAB' \mid aB'SBAB' \mid bBASBAB' \mid aSBAB'$$

$$\mid bBAB' \mid bBAB'SBBA \mid aB'SBBA \mid bBASBBA \mid aSBBA \mid bBBA$$

Example 4: Convert following into CNF.

$$S \rightarrow aAbB \mid BbS$$

$$B \rightarrow aAbA \mid aAB \mid b$$

$$A \rightarrow aB \mid aBb \mid a$$

Solution:

$$1. \quad S \rightarrow C_a \underline{A} C_b \underline{B}$$

$$S \rightarrow XY$$

$$X \rightarrow C_a A$$

$$Y \rightarrow C_b B$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

$$2. \quad S \rightarrow BC_b S$$

$$S \rightarrow PS$$

$$P \rightarrow BC_b$$

3. $B \rightarrow \underline{aA} \underline{bA}$
 $B \rightarrow \underline{C_a A} \underline{C_b A}$
 $B \rightarrow X Q$
 $X \rightarrow C_a A$ (already added)
 $Q \rightarrow C_b A$

4. $B \rightarrow aAB$
 $B \rightarrow \underline{C_a A} B$
 $B \rightarrow XB$

5. $A \rightarrow C_a B$

6. $A \rightarrow \underline{C_a B} C_b$
 $A \rightarrow RC_b$
 $R \rightarrow C_a B$

\therefore Equivalent grammar in CNF is

$$\begin{aligned} S &\rightarrow XY \mid PS \\ X &\rightarrow C_a A \\ Y &\rightarrow C_b B \\ P &\rightarrow BC_b \\ B &\rightarrow XQ \mid XB \mid b \\ Q &\rightarrow C_b A \\ A &\rightarrow C_a B \mid RC_b \mid a \\ R &\rightarrow C_a B \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

Example 5: Define useless symbols.

Solution: Let $G = (V, T, P, S)$ be a grammar. A symbol X is useful if there is a derivation.

$$S \Rightarrow \alpha X \beta \Rightarrow w, \text{ where } \alpha, \beta \in (VUT)^*$$

Otherwise X is useless.

Example 6: What is the yield of following derivation tree?

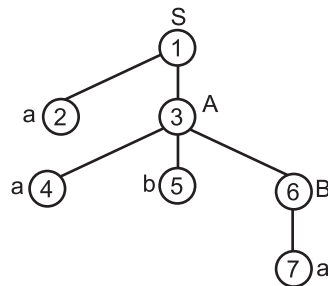


Fig. 3.20

Solution: The yield is aaba.

Example 7: Construct CFG for a language over $\{a, b\}$ which accepts equal number of a's and b's.

Solution:

$$\begin{aligned} G &= (V, T, P, S) \\ \text{where } V &= \{S\} \quad T = \{a, b\} \\ P \text{ is } S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

Example 8: Construct the following grammar into GNF.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow SB \mid a \\ B &\rightarrow AB \mid b \end{aligned}$$

Solution: Replace all S-productions in A-productions.

We get

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow ABB \mid a && \text{By lemma 1} \\ B &\rightarrow AB \mid b \end{aligned}$$

Now apply lemma 2 for A-productions.

$[A \rightarrow A\alpha \mid \beta, \text{ then } A \rightarrow B_i \mid B_i A' \text{ and } A' \rightarrow \alpha_i \mid \alpha_i A']$

$$A \rightarrow \underbrace{ABB}_{\alpha} \mid \underbrace{a}_{\beta}$$

$$\begin{aligned} A &\rightarrow aA' \mid a \\ A' &\rightarrow BBA' \mid BB \end{aligned}$$

Now all A-productions are in GNF, replace all A-productions in S-production, we get

$$\begin{aligned} S &\rightarrow aA'B \mid aB \\ A &\rightarrow aA' \mid a \\ A' &\rightarrow BBA' \mid BB \\ B &\rightarrow AB \mid b \end{aligned}$$

Now replace all A-productions in B-productions

$$\begin{aligned} S &\rightarrow aA'B \mid aB \\ A &\rightarrow aA' \mid a \\ A' &\rightarrow BBA' \mid BB \\ B &\rightarrow aA'B \mid aB \mid b \end{aligned}$$

Now replace all B-productions in A'-productions we get

$$\begin{aligned} S &\rightarrow aA'B \mid aB \\ A &\rightarrow aA' \mid a \\ A' &\rightarrow aA'BBA' \mid aA'BB \mid aBBA' \mid aBB \mid bBA' \mid bB \\ B &\rightarrow aA'B \mid aB \mid b \end{aligned}$$

The grammar is in GNF with total productions = 13.

Example 9: Convert the following into CNF.

$$S \rightarrow aSa \mid bSb$$

$$S \rightarrow a \mid b \mid aa \mid bb$$

Solution: CNF means the productions are in the form

$$A \rightarrow BC \text{ or } A \rightarrow a \text{ where } A, B, C \in V$$

$$S \rightarrow ASA \mid BSB$$

$$S \rightarrow a \mid b \mid AA \mid BB \quad \text{Replacing } a \text{ by } A \text{ and } b \text{ by } B$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Then convert $S \rightarrow ASA$ and $S \rightarrow BSB$ in CNF by introducing new variable C and D .

$$S \rightarrow AC \mid BD \mid a \mid b \mid AA \mid BB$$

$$C \rightarrow SA$$

$$D \rightarrow SB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

This is an equivalent CNF.

Example 10: Define unit production with an example.

Solution: A production of the form $A \rightarrow B$, where $A, B \in V$.

Example 11: Construct CFG which accepts set of palindromes over $\{0, 1\}$.

Solution: $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$

Example 12: Convert following grammar into CNF.

$$S \rightarrow AaB \mid a$$

$$A \rightarrow SBb \mid bA$$

$$B \rightarrow Ba \mid b$$

Solution: $S \rightarrow AC_a B \mid a$

$$A \rightarrow SBC_b \mid C_b A$$

$$B \rightarrow BC_a \mid b$$

Convert $S \rightarrow AC_a B$, we get $S \rightarrow AD_1$

$$D_1 \rightarrow C_a B$$

Convert $A \rightarrow SBC_b$, we get $A \rightarrow SD_2$

$$D_2 \rightarrow BC_b$$

∴ Grammar in CNF is,

$$\begin{aligned} S &\rightarrow AD_1 \mid a \\ A &\rightarrow SD_2 \mid C_b A \\ B &\rightarrow BC_a \mid b \\ D_1 &\rightarrow C_a B \\ D_2 &\rightarrow BC_b \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

Example 13: Rewrite the grammar after removing ϵ -productions.

$$\begin{aligned} S &\rightarrow aS \mid AB \\ A &\rightarrow a \mid \epsilon \\ B &\rightarrow b \mid \epsilon \\ D &\rightarrow b \end{aligned}$$

Solution: After removing ϵ -productions, we get,

$$\begin{aligned} S &\rightarrow aS \mid AB \\ A &\rightarrow a \\ B &\rightarrow b \\ D &\rightarrow b \end{aligned}$$

A and B are nullable symbols.

$$\begin{aligned} S &\rightarrow aS \mid AB \mid A \mid B \mid a \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Symbol D is useless.

Example 14: Every CFL is regular language. State true or false.

Solution: False. CFL are regular or non-regular.

Example 15: Define ambiguous grammar.

Solution: CFG is ambiguous if for atleast one word in the language that generates two possible derivations that corresponds to different parse trees or syntax trees.

Example 16: What is yield of tree given below:

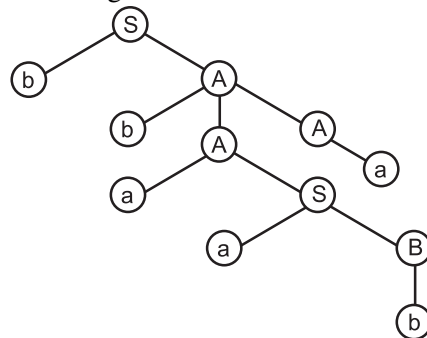


Fig. 3.21

Solution: bbaaba is the yield of tree.

Example 17: Convert the following grammar into GNF.

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow a \mid b \\ B &\rightarrow Bb \mid aa \\ C &\rightarrow aC \mid cC \mid ba \end{aligned}$$

Solution: First convert the grammar into CNF.

$$\begin{aligned} S &\rightarrow AD \\ D &\rightarrow BC \\ A &\rightarrow a \mid b \\ B &\rightarrow BC_b \mid aC_a \\ C &\rightarrow C_a C \mid C_c C \mid C_b C_a \\ C_b &\rightarrow b \text{ and } C_a \rightarrow a \end{aligned}$$

Replace A-productions in S, we get

$$\begin{aligned} S &\rightarrow aD \mid bD \\ A &\rightarrow a \mid b \\ B &\rightarrow B\underline{C}_b \mid a\underline{C}_a && \text{Apply lemma 2} \\ &\quad \alpha \beta \\ B &\rightarrow aC_a \mid aC_a B' \\ B' &\rightarrow C_b \mid C_b X \end{aligned}$$

Replace C_b by b and C_a by a , we get

$$\begin{aligned} S &\rightarrow aD \mid bD \\ A &\rightarrow a \mid b \\ B &\rightarrow aC_a \mid aC_a B' \\ B' &\rightarrow b \mid bX \\ C &\rightarrow aC \mid cC \mid bC_a \end{aligned}$$

Now consider D productions, replace B-productions in it we get,

$$D \rightarrow aC_a C \mid aC_a B' C$$

\therefore The equivalent GNF grammar is

$$\begin{aligned} S &\rightarrow aD \mid bD \\ A &\rightarrow a \mid b \\ B &\rightarrow aC_a \mid aC_a B' \\ B' &\rightarrow b \mid bX \\ C &\rightarrow aC \mid cC \mid bC_a \\ D &\rightarrow aC_a C \mid aC_a B' C \end{aligned}$$

Example 18: Remove unit productions from the following grammar.

$$S \rightarrow A \mid bb$$

$$A \rightarrow B \mid b$$

$$B \rightarrow S \mid a$$

Solution:

$$S \rightarrow A \mid bb$$

$$A \rightarrow B \mid b$$

$$B \rightarrow A \mid bb \mid a \quad \text{Removing unit production from B}$$

After, remove unit production in A by replacing B-production.

$$A \rightarrow A \mid bb \mid a \mid b$$

$$A \rightarrow bb \mid a \mid b \quad (\text{Since } A \rightarrow A, A \text{ is useless})$$

Substitute A-production in S, we get the grammar

$$S \rightarrow bb \mid a \mid b$$

Example 19: State any one lemma used in the procedure for constructing GNF.

Solution:

$$A \rightarrow \alpha\beta\gamma$$

$$B \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \dots \dots \mid \beta_m$$

then

$$A \rightarrow \alpha\beta_1\gamma \mid \alpha\beta_2\gamma \mid \dots \dots \mid \alpha\beta_m\gamma$$

Example 20: Define right linear grammar.

Solution: If all productions of CFG are of the form $A \rightarrow \omega B$ or $A \rightarrow \omega$ or $A \rightarrow \epsilon$, where A, B are variables and ω is string of terminal, then grammar is called right linear grammar.

Example 21: Construct CFG for $L = \{0^x 1^y 0^z \mid y > x + z\}$

Solution:

$$G = (V, T, P, S) \quad V = \{S, A, B, C, D\} \quad T = \{0, 1\}$$

where P is

$$\{ S \rightarrow 0A00 \mid 00B0 \mid 0C0$$

$$A \rightarrow 1A11 \mid 11D1$$

$$D \rightarrow 1 \mid D1$$

$$B \rightarrow 1B11 \mid 11D1$$

$$C \rightarrow 1C1 \mid 1D1$$

}

S is start symbol.

Example 22: Convert the following grammar into GNF.

$$S \rightarrow aAS \mid a$$

$$A \rightarrow SbA \mid SS \mid bA$$

Solution: Substitute S-production in A-productions, we get

$$S \rightarrow aAS \mid a$$

$$A \rightarrow aAsbA \mid aASS \mid aSbA \mid aS \mid bA$$

Substitute b by B, we get

$$\begin{aligned} S &\rightarrow aAS \mid a \\ A &\rightarrow aASBA \mid aASS \mid aSBA \mid aS \mid bA \\ B &\rightarrow b \end{aligned}$$

The grammar in GNF.

Example 23: Remove unit production from the following grammar.

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow B \mid b \\ B &\rightarrow S \mid a \end{aligned}$$

Solution:

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow B \mid b \\ B &\rightarrow A \mid bb \mid a \end{aligned}$$

Remove unit production from B-production.

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow A \mid bb \mid a \mid b \quad (A \text{ is useless in } A \rightarrow A) \end{aligned}$$

Substitute A-production in S.

$$\begin{aligned} S &\rightarrow A \mid bb \\ A &\rightarrow bb \mid a \mid b \end{aligned}$$

After substituting we get

$$S \rightarrow bb \mid a \mid b$$

Example 24: What are the types of grammar in the Chomsky hierarchy?

Solution: Type 0(unrestricted)

Type 1 (context-sensitive)

Type 2 (context-free)

Type 3 (regular)

Example 25: Define inherently ambiguous context free languages.

Solution: If small some word of a language has more than one leftmost derivation or more than one rightmost derivation then it is ambiguous. A CFL for which every CFG is ambiguous is said to be an inherently ambiguous CFL.

Example 26: Construct CFG for each of the following languages:

(i) ab^*a , (ii) $0^*1^*2^*$, (iii) $\{a^n b^n c^m \mid n \geq 1, m \geq 0\}$.

Solution: (i) ab^*a : Grammar is

$$\begin{aligned} S &\rightarrow aBa \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

(ii) $0^*1^*2^*$: Grammar is

$$S \rightarrow ABC$$

$$A \rightarrow 0A \mid \epsilon$$

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

$$C \rightarrow 2C \mid \epsilon$$

(iii) $\{a^n b^n c^m \mid n \geq 1, m \geq 0\}$

$$\text{CFG is } S \rightarrow AS \mid Sc$$

$$A \rightarrow ab \mid aAb$$

Example 27: State lemma 2 for converting a CFG to GNF.

Solution: If $A \rightarrow A\alpha_1 \mid A\alpha_2 \dots \mid A\alpha_i \mid \beta_1 \beta_2 \dots \mid \beta_i$

then, $A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_i \mid \beta_1\beta_2 \mid \beta_2\beta_3 \dots \mid \beta_i\beta$

$$B \rightarrow \alpha_1 \mid \alpha_2 \dots \mid \alpha_i \mid \alpha_1\beta_1\alpha_2\beta_2 \dots \mid \alpha_i\beta$$

Example 28: Construct CFG for the following sets.

(i) $\{a^n b^n c^i \mid n \geq 1, i \geq 0\}$

(ii) $\{a^j b^n c^n \mid n \geq 1, j > 0\}$

(iii) language in which every string has triple 1's.

Solution: (i) $S \rightarrow AS \mid Sc$

$$A \rightarrow ab \mid aAb$$

(ii) $S \rightarrow Sa \mid SA$

$$A \rightarrow bc \mid bAc$$

(iii) $S \rightarrow A111B$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

Example 29: Define context-sensitive grammar.

Solution: For each production $\alpha \rightarrow \beta$ length of β is atleast as much as the α except $S \rightarrow \epsilon$.

Example, $A \rightarrow AB$

$$AB \rightarrow AC$$

$$C \rightarrow ab$$

Example 30: Construct the CFG accepting the sets

(i) $L = \{0^n 1^m \mid n, m \geq 0\}$

(ii) $L = L_1 \cup L_2$, where $L_1 = \{a^n b \mid n \geq 0\}$ and $L_2 =$ All strings not having 01 as a substring over $\{0, 1\}$.

Solution: (i) $S \rightarrow 0S1 \mid 01 \mid 0A1$

$A \rightarrow 0A \mid 0$

(ii) L_1 is $S_1 \rightarrow Ab$

$A \rightarrow aA \mid \epsilon$

L_2 is $S_2 \rightarrow 0X \mid 1S_2 \mid 0 \mid 1$

$X \rightarrow 0X \mid 0$

For $L_1 \cup L_2$ the CFG is

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow Ab$

$A \rightarrow aA \mid \epsilon$

$S_2 \rightarrow 0X \mid 1S_2 \mid 0 \mid 1$

$X \rightarrow 0X \mid 0$

Example 31: Write the steps for eliminating ϵ -production in CFG.

Solution: (i) Eliminate ϵ -productions.

(ii) Find nullable non-terminals N where $N \Rightarrow \epsilon$ or $N \rightarrow \epsilon$

(iii) Replace nullable non-terminals as ϵ in RHS for all productions where it is present. Find all subsets by replacing ϵ .

Example 32: Class of CFG and PDA is same. Justify true or false.

Solution: True. PDA is the acceptor of CFG.

Example 33: What is yield from the following Parse tree?

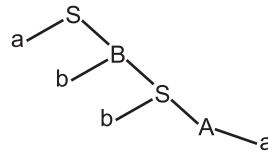


Fig. 3.22

Solution: abba.

Example 34: Construct CFG for language $L = \{a^n b^n c^m d^m \mid m, n \geq 1\}$.

Solution: $S \rightarrow AB$
 $A \rightarrow aAb \mid ab$
 $B \rightarrow cBd \mid cd$

(i) $S \Rightarrow AB$
 $\Rightarrow abcd$

(ii) $S \Rightarrow AB$
 $\Rightarrow aAbcBd$
 $\Rightarrow aabbccdd$

Solution:

$$\begin{aligned} S &\rightarrow AC \\ A &\rightarrow aAb \mid ab \mid \epsilon \\ C &\rightarrow cC \mid c \end{aligned}$$
$$\begin{aligned} S &\rightarrow a \mid AA \mid BA \\ A &\rightarrow a \mid AB \mid b \\ B &\rightarrow a \end{aligned}$$
$$\begin{aligned} \therefore \quad & A \rightarrow aA' \mid bA' \\ & A' \rightarrow BA' \mid B \end{aligned}$$
$$\begin{aligned} S &\rightarrow a \mid aA'A \mid bA'A \mid aA \\ A &\rightarrow aA' \mid bA' \\ A' &\rightarrow aA' \mid a \\ B &\rightarrow a \end{aligned}$$

Example 37: Convert the grammar into CNF.

$$\begin{aligned} S &\rightarrow A \mid B \mid C \\ A &\rightarrow aAa \mid B \\ B &\rightarrow bB \mid bb \\ C &\rightarrow aCaa \mid D \\ D &\rightarrow baD \mid abD \mid aa \end{aligned}$$

Solution:

$$\begin{aligned} S &\rightarrow aAa \mid B \mid bB \mid bb \mid aCaa \mid D \\ S &\rightarrow A'AA' \mid B'B \mid B'B \mid BB \mid A'CA'A' \mid D \\ A' &\rightarrow a \\ B' &\rightarrow b \\ S &\rightarrow A'D \\ S &\rightarrow B'B \\ S &\rightarrow BD \\ S &\rightarrow B'B \mid BB \end{aligned}$$

$$\begin{aligned}
 S &\rightarrow D_2 D_3 \\
 D &\rightarrow AA' \\
 D_2 &\rightarrow A'C \\
 D_3 &\rightarrow A'A' \\
 D &\rightarrow B'A'D \mid A'B'D \mid A'A' \\
 D &\rightarrow B'D_4 \mid A'D_5 \mid A'A' \\
 D_4 &\rightarrow A'D \\
 D_5 &\rightarrow B'D
 \end{aligned}$$

Example 38: Construct CFG for $L = L_1 \cup L_2$ where $L_1 = \{a^n b \mid n \geq 1\}$ and $L_2 = \{0^n \mid n^n n \geq 1\}$

Solution: $L_1 \cup L_2 =$ all strings of L_1 and L_2

CFG of L_1 is $S_1 \rightarrow Ab$
 $A \rightarrow aA \mid b$

CFG of L_2 is $S_2 \rightarrow AB$
 $A \rightarrow 0A \mid 0$
 $B \rightarrow 1B \mid 1$

CFG of $L_1 \cup L_2$ is $S \rightarrow S_1 \mid S_2$

Example 39: Rewrite the following CFG after eliminating unit productions.

$$\begin{aligned}
 S &\rightarrow aAb \mid A \\
 A &\rightarrow B \mid b \\
 B &\rightarrow \epsilon \\
 D &\rightarrow F \\
 F &\rightarrow 01 \mid B
 \end{aligned}$$

Solution: $S \rightarrow aAb \mid b \mid \epsilon$
 $A \rightarrow b \mid \epsilon$
 $D \rightarrow 01 \mid \epsilon$

Example 40: What is the field of following derivation tree.

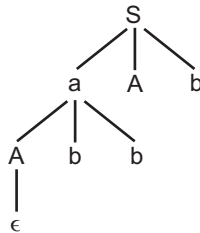


Fig. 3.23

Solution: abbb.

Example 41: Find nullable symbols of following CFG.

$$S \rightarrow AB \mid aBb$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow AD \mid aAb$$

$$D \rightarrow bD \mid \epsilon$$

Solution: S, A, B, D all are nullable.

Example 42: Find CFG for the language, $L = \{a^n b^m c^n \mid m, n \geq 1\}$.

Solution: $S \rightarrow aSc \mid aAc \mid abc$

$$A \rightarrow aAc \mid \epsilon$$

Example 43: If $L_1 = \{a^m b^m \mid m \geq 1\}$ and $L_2 = \{b^m \mid m \geq 0\}$

Find $L_1 \cup L_2$ and $L_1 \cap L_2$.

Solution: $\{\epsilon, ab, b, bb, aabb, bbb, aaabbb, \dots\}$

Example 44: Define useless symbol.

Solution: Useless symbol: If $S \Rightarrow \alpha X \beta \Rightarrow \alpha \omega \beta$, X is useful if it is derivable from S and it derives the sentence (input string) otherwise X is useless.

Example 45: Find CFG for the language represent by the following regular expression:

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

Solution: $S \rightarrow ABA$

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

$$B \rightarrow 01$$

Example 46: Construct PDA for a language: $L = \{W \subset W^R \mid W \in (0 + 1)^*\}$.

Solution: $\delta(q_1, 0, R) = (q, BR)$

$$\delta(q_1, 0, B) = (q_1, BB)$$

$$\delta(q_1, 0, G) = (q_1, BG)$$

$$\delta(q, 1, G) = (q_1, GG)$$

$$\delta(q_1, 1, B) = (q_1, GB)$$

$$\delta(q_1, C, R) = (q_2, R)$$

$$\delta(q_1, C, B) = (q_2, B)$$

$$\delta(q_1, C, G) = (q_2, G)$$

$$\delta(q_2, 0, B) = (q_2, \epsilon)$$

$$\delta(q_2, 1, G) = (q_2, \epsilon)$$

$$\delta(q_2, \epsilon, R) = (q_2, \epsilon)$$

Example 47: Construct regular grammar for the following DFA.

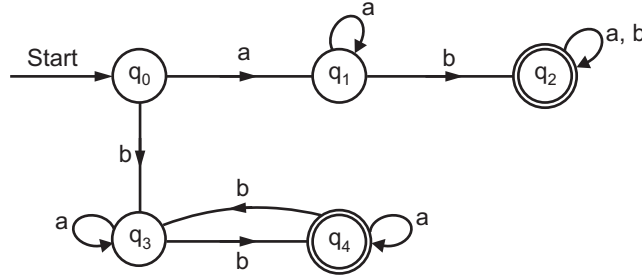


Fig. 3.24

Solution:

$$q_0 \rightarrow aq_1 \mid bq_3$$

$$q_1 \rightarrow aq_1 \mid bq_2 \mid b$$

$$q_2 \rightarrow aq_2 \mid bq_2 \mid a \mid b$$

$$q_3 \rightarrow aq_3 \mid bq_4 \mid b$$

$$q_4 \rightarrow aq_4 \mid bq_3 \mid a$$

Example 48: Define left linear and right linear grammar.

Solution: Left linear $\rightarrow A \rightarrow Bw \rightarrow w$ or $A \rightarrow w$ or $A \in \epsilon$.

Right linear $\rightarrow A \rightarrow wB$ or $A \rightarrow w$ or $A \rightarrow \epsilon$

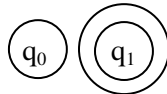
Example 49: Determine whether the following language is regular? Justify: $L = L_1 \cap L_2$ where:

$$L_1 = \{a^n b^m \mid n \geq m \text{ and } n > 0\}$$

$$L_2 = \{b^m a^n \mid m > 0, n > 1\}$$

Solution:

$$L = L_1 \cap L_2 = \emptyset \text{ It is regular.}$$



Example 50: Construct CFG for $L = L_1 L_2$

where

$$L_1 = \{a^n b \mid n \geq 1\} \text{ and}$$

$$L_2 = \{a^n b^{n+2} \mid n \geq 0\}$$

Solution:

$$S_1 \rightarrow aAb \mid ab \text{ and } S_2 \rightarrow aAbbb \mid bb$$

$$A \rightarrow aA \mid \epsilon \text{ and } A \rightarrow aAb \mid \epsilon$$

$$L = L_1 L_2$$

\therefore

$$S \rightarrow S_1 S_2$$

PRACTICE QUESTIONS**Q.I Multiple Choice Questions:**

1. Which grammars define the context free languages?
(a) Context Free Grammars (CFG)
(b) Regular Grammars (RG)
(c) Context Sensitive Grammars (CSG)
(d) Unrestricted Grammars (URG)
2. The language generated by the CFG is called as, context-free language (CFL).
(a) Context-Automata Language (CAL)
(b) Context-Free Language (CFL)
(c) Context-Regular Language (CRL)
(d) None of the mentioned
3. The productions (rules of a grammar) are applied iteratively to generate a string or language this process is called as,
(a) language
(b) Alphabet
(c) derivation
(d) expression
4. Which in grammar contains less number of non-terminals and productions, so the time complexity for the language generating process becomes less from the reduced grammar?
(a) derivation
(b) expression
(c) automata
(d) reduction
5. Which grammars generate the regular languages? Such a grammar restricts its rules to a single non-terminal on the left hand side. The right hand side consists of either a single terminal or a string of terminals with a single non-terminal on the left or the right end?
(a) Context Free Grammars (CFG)
(b) Regular Grammars (RG)
(c) Context Sensitive Grammars (CSG)
(d) Unrestricted Grammars (URG)
6. Which is the tree representation of deriving a CFL from a given context grammar?
(a) parse tree
(b) derivation tree
(c) Both (a) and (b)
(d) None of the mentioned
7. If we replace only the rightmost non-terminal by some production rule at each step of the generating process of the language, then the derivation is called as a,
(a) leftmost derivation
(b) rightmost derivation
(c) linear derivation
(d) None of the mentioned

8. A grammar is said to be _____ if any Non-terminal generates a single Non-terminal on the RHS.
 (a) linear (b) non-linear
 (c) right-linear (d) left-linear
9. The _____ symbols are those which are the constituents of the sentence generated using a grammar.
 (a) non-terminal (b) terminal
 (c) linear-terminal (d) None of the mentioned
10. A language is context free if it can be generated by,
 (a) CFG (b) FA
 (c) RE (d) RL
11. A grammar is said to be in Greibach Normal Form (GNF) if every production of the grammar is of the form,
 (a) Non-terminal \rightarrow single terminal (string of non-terminals)
 (b) Non-terminal \rightarrow single terminal (in one line, it can be said that all the productions will be in the form)
 (c) Non-terminal \rightarrow (single terminal)(non-terminal)
 (d) All of the mentioned
12. Useless symbols in CFG are,
 (a) Non-terminal symbols (b) Null alphabets and null string
 (c) Both (a) and (b) (d) Non-generating and non-reachable symbols
13. Parsing a string from a given grammar means,
 (a) Finding a derivation tree (b) Finding a leftmost derivation
 (c) Finding a derivation (d) Finding a rightmost derivation
14. Simplification of CFG includes,
 (a) Removal of useless symbols (b) Removal of unit productions
 (c) Removal of null productions (d) All of the mentioned
15. A grammar is called ambiguous if,
 (a) It generates more than one parse tree for a given string
 (b) It generates both leftmost and rightmost derivation for a given string
 (c) It generates more than one string
 (d) All of the mentioned
16. The intersection of CFL and regular set is,
 (a) regular (b) context free
 (c) non-context free (d) None of the mentioned

Answers

1. (a)	2. (b)	3. (c)	4. (d)	5. (b)	6. (c)	7. (b)	8. (a)	9. (b)	10. (a)
11. (d)	12. (d)	13. (c)	14. (d)	15. (a)	16. (b)				

Q.II Fill in the Blanks:

1. A regular (or finite) _____ describes a regular language and is equivalent to a finite automaton.
2. In a _____ grammar, all production rules have a single non-terminal on the left-hand side, and either a single terminal or a combination of a single non-terminal and a single terminal on the right-hand side.
3. Regular grammar is a _____ of Context-free grammar (CFG).
4. A _____ is called a leftmost derivation if we replace only the leftmost non-terminal by some production rule at each step of the generating process of the language from the grammar.
5. A grammar is said to be _____ if the productions contain a single one non-terminal on LHS of production which occur at the Left most of the string.
6. A _____ defines a formal language i.e. a set of all sentences that can be derived by the grammar.
7. A grammar of a language is called _____ if any of the cases for generating a particular string, more than one parse tree can be generated.
8. A CFG is said to be in Chomsky Normal Form (CNF) if all the _____ of the grammar are in the following form:
 - o Non-terminal \rightarrow String of exactly two non-terminals
 - o Non-terminal \rightarrow Single terminal
9. A grammar is said to be in _____ when every production of the grammar has some specific form.
10. The _____ for CFL is used to prove that certain sets are not context free.
11. A grammar is called _____ grammar if it is context free, and the RHS of all productions have at most one non-terminal.
12. A grammar is said to be '_____ if the productions contain a single non-terminal on R.H.S. of production which occur at the Right most of the string.
13. A parse tree (sometimes called as derivation trees) is the tree representation of deriving a CFL from a given _____ grammar.
14. _____ a string is finding a derivation for that string from a given grammar.
15. A _____ generated by a Context-Free Grammar (CFG) is called as Context-Free Language or CFL.

Answers

1. grammar	2. regular	3. subset	4. derivation
5. left-linear	6. CFG	7. ambiguous	8. productions
9. normal form	10. Pumping Lemma	11. linear	12. right-linear
13. context	14. Parsing	15. language	

Q.III State True or False:

1. Converting any context-free grammar to Chomsky normal form will ensure that it is unambiguous.
2. Non-context free languages are always defined by regular expression.
3. A regular grammar cannot describe a context free language.
4. Regular Languages are always Context-Free Languages (CFL).
5. A linear grammar can be converted to regular grammar.
6. Any FA can be converted into regular grammar.
7. A regular grammar is the least powerful type of grammar in the Noam Chomsky hierarchy (a context free grammar describes a context free language, a context sensitive grammar describes a context sensitive language, and an unrestricted grammar describes a recursively enumerable language).
8. All context free languages are infinite set.
9. In a CFG, when the productions in a grammar satisfy certain restrictions, then that grammar is said to be in its normal form.
10. A linear grammar is called left linear if the RHS non-terminal in each productions are at the left end.
11. A linear grammar is called right linear if the RHS non-terminal in each productions are at the right end.
12. String consists of only non-terminal symbols.
13. A regular grammar is a subset of CFG thus, for every regular language, there exists a CFG.
14. A context-free grammar with more than one parse tree for some expression is called ambiguous.
15. Context Free Languages are always Regular Languages.
16. Pumping Lemma is used to show that language is not context free.
17. Different types of derivation can be generated for deriving a particular string from a given grammar and for each of the derivations, a parse tree is generated.
18. In simplification of CFG unnecessary productions of a grammar should be eliminated.
19. The derivations in a CFG can be represented using trees called as derivation trees.
20. A parse tree is an ordered tree in which the LHS of a production represents a parent node and the RHS of a production represents a children node.

Answers

1. (T)	2. (F)	3. (T)	4. (T)	5. (T)	6. (T)	7. (T)	8. (F)	9. (T)	10. (T)
11. (T)	12. (F)	13. (T)	14. (T)	15. (F)	16. (T)	17. (T)	18. (T)	19. (T)	20. (T)

Q.IV Answer the following Questions:**(A) Short Answer Questions:**

1. Define grammar.
2. Define CFG.
3. What is CPL.
4. Define regular language.
5. Define normal form.
6. Define linear grammar.
7. Define RG.
8. Compare CNF and GNF (any two points).
9. Define derivation.
10. Define ambiguous grammar.
11. Define left and right linear grammar.
12. What is meant by RMD and LMD?

(B) Long Answer Questions

1. What is grammar? Explain with example.
2. With the help of example describe derivation.
3. What is reduction? Define it. Also explain with example.
4. Construct the CFG accepting each of the following sets:
 - (i) $\{a^n b^m a^m b^n \mid m, n \geq 1\}$
 - (ii) $\{a^n b^{2n} \mid n \geq 1\}$
 - (iii) $L = L_1 \cup L_2$

$$L_1 = \{0^m 1^n 2^{m-n} \mid m, n \geq 1\}$$

$$L_2 = \{0^i 2^{2i-2} \mid i \geq 0\}$$
 - (iv) Set of balanced parenthesis over $\{(,), [,], \{, \}\}$.
 - (v) The set of all strings over $\{a, b\}$ consisting of equal number of a's and b's.
 - (vi) Set of all strings over $\{a, b\}$ starting with "a" and ending with "bb".
 - (vii) $L = L_1 \cup L_2$ where

$$L_1: \{a^n b^m \mid n \geq m, n > 0\}$$

$$L_2: \text{all strings not having "ab" as substring over } \{0, 1\}.$$
5. Consider the following grammar:

$$S \rightarrow aB \mid bA$$

$$A \rightarrow aS \mid bAA \mid a$$

$$B \rightarrow bS \mid aBB \mid b$$

For the string "aaabbabbba", find

- (i) Leftmost derivation
- (ii) Rightmost derivation
- (iii) Parse tree.

6. Show that the following CFGs are ambiguous:

$$(i) \quad S \rightarrow a \mid abSb \mid aAb$$

$$A \rightarrow bS \mid aAAb$$

$$(ii) \quad S \rightarrow aB \mid ab$$

$$A \rightarrow aAB \mid a$$

$$B \rightarrow ABb \mid b$$

$$(iii) \quad S \rightarrow S_0S_0S \mid 1$$

$$(iv) \quad S \rightarrow S + S \mid S * S \mid (S) \mid a$$

7. Find CFGs with no useless symbols equivalent to CFG given below:

$$(i) \quad S \rightarrow AB \mid CA$$

$$A \rightarrow a$$

$$B \rightarrow BC \mid AB$$

$$C \rightarrow aB \mid b$$

$$(ii) \quad S \rightarrow AB \mid BC$$

$$A \rightarrow aAa \mid aAb$$

$$B \rightarrow bB \mid b$$

$$D \rightarrow dD \mid d$$

$$(iii) \quad S \rightarrow aAB \mid BC \mid aB$$

$$A \rightarrow bA \mid aC$$

$$B \rightarrow bBB \mid aS \mid b$$

$$C \rightarrow CA \mid BC$$

8. Design equivalent CFG without unit productions for following CFGs:

$$(i) \quad S \rightarrow 0A \mid 1B$$

$$A \rightarrow S$$

$$B \rightarrow 1B \mid 1$$

$$(ii) \quad S \rightarrow 0A1 \mid 1B0$$

$$A \rightarrow 0A \mid B10$$

$$B \rightarrow 1A \mid S \mid 1$$

- (iii) $S \rightarrow AB$
 $A \rightarrow B$
 $B \rightarrow 0B \mid 1 \mid \epsilon$

9. Convert following CFG into CNF:

- (i) $S \rightarrow aSd \mid aAd$
 $A \rightarrow bAc \mid bc$
- (ii) $S \rightarrow 01S1 \mid 0 \mid 0A1$
 $A \rightarrow 1S \mid 0AA1$
- (iii) $S \rightarrow abAB \mid bAda$
 $A \rightarrow baB \mid a$
 $B \rightarrow CAb \mid Bb$
- (iv) $S \rightarrow aAbB \mid BbS$
 $B \rightarrow aAbA \mid aAB \mid b$
 $A \rightarrow aB \mid aBb \mid a$
- (v) $S \rightarrow aAab \mid Aba$
 $A \rightarrow aS \mid bB$
 $B \rightarrow ASB \mid a$

10. Convert following CFG into GNF:

- (i) $S \rightarrow 0AB \mid A \mid B$
 $A \rightarrow S0B \mid 1B \mid 1$
 $B \rightarrow A1 \mid 0$
- (ii) $S \rightarrow AB$
 $A \rightarrow BSB \mid BB \mid b$
 $B \rightarrow aAb \mid a$
- (iii) $S \rightarrow aSd \mid aAd$
 $A \rightarrow bAc \mid bc$
- (iv) $S \rightarrow aA \mid b$
 $A \rightarrow SA \mid a$
- (v) $S \rightarrow AB$
 $A \rightarrow BS \mid b$
 $B \rightarrow SA \mid a$

- (vi) $S \rightarrow AaB \mid a$
 $A \rightarrow SBb \mid bA$
 $B \rightarrow Ba \mid b$
- (vii) $S \rightarrow 0A0 \mid 1B1 \mid BB$
 $A \rightarrow C$
 $B \rightarrow S \mid A$
 $C \rightarrow S \mid \epsilon$
- (viii) $S \rightarrow AAA \mid B$
 $A \rightarrow aA \mid B$
 $B \rightarrow \epsilon$

11. Consider following grammar:

- $S \rightarrow A * B \mid * A$
 $A \rightarrow \# B \mid B \#$
 $B \rightarrow * A \mid \#$

For the string " $\# * \# \# * \#$ ", find

- (i) Leftmost derivation
(ii) Rightmost derivation
(iii) Parse tree.

12. Write a short note on: Simplification of CFG.

13. Describe Chomsky hierarchy with four types of grammars.

14. What is ambiguous grammar? Explain its concept with example.

15. Explain equivalence of FA and regular grammar with example.

16. How to construct of regular grammar equivalent to a given DFA?

17. What is normal form? Explain Greibach Normal Form (GNF) and Chomsky Normal Form (CNF) with example. Also compare them.

18. Construct CFG accepting following sets:

- (i) $L = \{0^n 1^m \mid n, m \geq 0, n \text{ is not equal to } m\}$
(ii) $L = \{a^m b^n c^{m-n} \mid m \geq 1, m > n\}$
(iii) $L = \{a^m b^n c^m \mid n \geq 1, m \geq 0\}$
(iv) The set of all strings with exactly twice as many b's as a's.
(v) $L = L_1 \cup L_2$ where
 $L_1 = \{a^n b \mid n \geq 0\}$
 $L_2 = \{0^m 1^n 2^{n+1} \mid m, n \geq 1\}$

19. Construct leftmost and rightmost derivations for ababa and draw parse tree.

$$S \rightarrow AS \mid a$$

$$A \rightarrow SA \mid b$$

20. Show that the grammar

$$S \rightarrow a \mid abSb \mid aAb$$

$$A \rightarrow bS \mid aAAb \text{ is ambiguous.}$$

21. For the following grammar, find an equivalent grammar with no unit production:

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow C \mid b$$

$$C \rightarrow D$$

$$D \rightarrow E$$

$$E \rightarrow a$$

22. Eliminate ϵ -productions from the following grammar:

$$S \rightarrow AB \mid \epsilon$$

$$A \rightarrow aASb \mid a$$

$$B \rightarrow bS$$

23. Construct regular grammar for the following languages:

- (i) Set of all strings over $\{a, b\}$ such that if it starts with 'a' then it ends with "ab" and if it starts with 'b' then it contains even number of a's in it.

- (ii) $(a + b)^* aba (a + b)^* bb$

- (iii) $\{a^{2n} \mid n \geq 1\}$

- (iv) The set of all strings over $\{0, 1\}$ beginning with "0".

- (v) $\{a^l b^m c^n \mid l, m, n \geq 1\}$

- (vi) $01^*((01)(10))^* + 1(10)^*$.

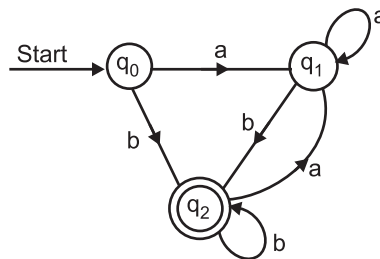
- (vii) $L = L_1 \cap L_2$, where

$L_1 =$ All strings over $\{a, b, c\}$ having equal number of a's and c's.

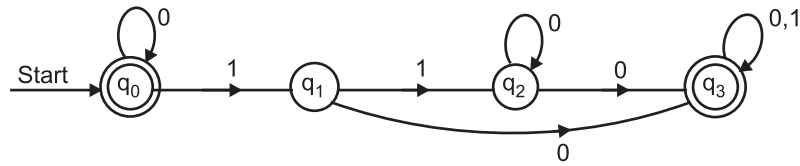
$L_2 = \{a^n b c^n \mid n \geq 0 \text{ and } n \leq 5\}$

24. Construct regular grammar for DFA in Fig. 4.19 to 4.21

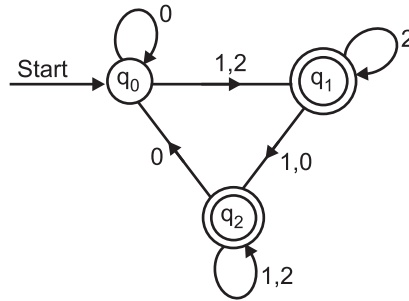
- (i)



(ii)



(iii)

25. $S \rightarrow ABC \mid BaB$ $A \rightarrow aA \mid BaC \mid aaa$ $B \rightarrow bBb \mid a \mid D$ $C \rightarrow CA \mid AC$ $D \rightarrow \epsilon$

- (i) Eliminate ϵ -productions.
- (ii) Eliminate any unit productions.
- (iii) Eliminate any useless symbols.
- (iv) Convert into CNF.

26. Show that following languages are not context free.

- (a) $\{a^m b^n c^k \mid m < n < k\}$
- (b) $\{a^n \mid n \text{ is a prime}\}$
- (c) $\{a^m b^{m^2} \mid m \geq 1\}$
- (d) $\{a^n b^n c^m \mid n \leq m \leq 2n\}$
- (e) $\{a^n b^n a^n b^n \mid n \geq 1\}$
- (f) $\{a^n b^n c^n d^n \mid n \geq 1\}$

27. Show that the CFL's are closed under the following properties:

- (a) union,
- (b) concatenation,
- (c) closure.

28. Prove that CFL's are not closed under intersection.
29. Is the language $L = \{a^n b^{3n} a^n \mid n \geq 1\}$ context free?
If so, find a CFG. If not, prove it.
30. L_1 and L_2 are two regular languages?
 $L_1 - L_2 = \{x \mid x \text{ is in } L_1 \text{ but not in } L_2\}$
 Is $(L_1 - L_2)$ is regular language?
(Hint: $L_1 - L_2 = L_1 \cap \bar{L}_2$)
31. L_1 and L_2 are two non-regular languages. Then their union is also non-regular, state true or false and explain.
32. With the help of example describe concept of left linear and right linear grammar.
33. How to construction of a FA from the given right linear grammar? Explain with example.
34. What is reduction? Describe with example.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Define left linear and right linear grammar. **[1 M]**
Ans. Refer to Section 3.9.
2. State the machines used for context free grammar and context-sensitive grammar. **[1 M]**
Ans. Refer to Section 3.2.
3. Construct CFG for the $L = \{a^x b^y c^{x+y} \mid x, y \geq 1\}$. **[1 M]**
Ans. Refer to Section 3.4.
4. Define the terms:
 (i) Ambiguous grammar (ii) Parse tree. **[2 M]**
Ans. Refer to Sections 3.5 and 3.2.1.

October 2016

1. State the machines used for CFL and CFG. **[1 M]**
Ans. Refer to Sections 3.1 and 3.2.
2. Define ambiguous grammar. **[1 M]**
Ans. Refer to Section 3.5.
3. Construct CFG for language $L = \{a^n b^n c^m d^r \mid m, n, r \geq 1\}$. **[4 M]**
Ans. Refer to Section .
4. Define type-2 grammar. **[1 M]**
Ans. Refer to Section 3.3, Point (3).

5. Convert the following grammar to GNF:

$$S \rightarrow AA \mid a$$

$$A \rightarrow SS \mid b$$

[5 M]

Ans. The given CFG is in CNF

By replacing the production of A in S we get

$$S \rightarrow SSA \mid bA \mid a$$

After removing the left recursion, we get

$$S \rightarrow bA \mid a \mid bAS' \mid as'$$

$$S \rightarrow SA \mid SAS'$$

The productions of S are in GNF.

By replacing the productions of S in A we get

$$A \rightarrow bAS \mid as \mid bAS's \mid as's \mid b$$

Now the productions of A are in GNF

By replacing the productions of s in s' we get

$$S' \rightarrow bAA \mid aA \mid bAs'A \mid as'A$$

$$S' \rightarrow bAAS' \mid aAS' \mid bAS'AS' \mid as'AS'$$

So the grammar in GNF is $G = ((S, AS'), (a, b), P, S)$

Where $P = S \rightarrow bA \mid a \mid bAS' \mid as'$

$$= S' \rightarrow bAA \mid aA \mid bAs'A \mid as'A$$

$$= S' \rightarrow bAAS' \mid aAS' \mid bAS'AS' \mid as'AS'$$

$$= A \rightarrow bAS \mid as \mid bAS'S \mid as'S \mid b$$

April 2017

1. Write a language for CFG: $S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$.

[1 M]

Ans. Refer to Examples on Pages 3.14 to 3.15.

2. Construct CFG for language $L = \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$.

[3 M]

Ans. Refer to Examples on Pages 3.14 to 3.15.

3. Construct CFG for language L which accepts set of all palindromes over $\Sigma = \{a, b\}$.

[2 M]

Ans. Refer to Examples on Pages 3.14 to 3.15.

4. Convert the following grammar to GNF:

$$S \rightarrow AB \mid B$$

$$A \rightarrow BS$$

$$B \rightarrow A1 \mid 1$$

[5 M]

Ans. Refer to Section 3.7.2.

October 2017

1. Define ambiguous grammar.

[1 M]

Ans. Refer to Section 3.5.

2. Consider the following grammar:

$$S \rightarrow ADa$$

$$A \rightarrow a$$

$$D \rightarrow d$$

The grammar is in CNF. Justify.

[1 M]

Ans. Refer to Section 3.7.1.

3. Convert the following grammar in GNF:

$$S \rightarrow aAS \mid a$$

$$A \rightarrow SbA \mid SS \mid bA$$

[5 M]

Ans. Refer to Section 3.7.2.

4. Construct CFG for the language $L = L_1L_2$

where, $L_1 = \{a^n b \mid n \geq 0\}$

$$L_2 = \{b^m c \mid m \geq 0\}.$$

[4 M]

Ans. Refer to Examples on Pages 3.14 to 3.15.

5. Write a short note on Chomsky's hierarchy.

[4 M]

Ans. Refer to Section 3.3.

6. Consider the following grammar:

$$S \rightarrow AB \mid aD \mid a$$

$$A \rightarrow a$$

$$D \rightarrow aD \mid aDD$$

Remove useless symbols and rewrite the grammar.

[2 M]

Ans. Refer to Section 3.6.1.

April 2018

1. Define right linear grammar.

[1 M]

Ans. Refer to Section 3.9.

2. State lemma 1 for converting a CFG to GNF.

[1 M]

Ans. Refer to Examples on 3.31 to 3.33.

3. Construct the following CFG into Chomsky Normal Form (CNF):

$$S \rightarrow aSa \mid bSb \mid aab \mid bba$$

[5 M]

Ans. Refer to Section 3.7.1.

4. Construct CFG for the following: [5 M]

(i) $L = \{a^n b^{n+2} \mid n \geq 0\}$

(ii) A language containing string having at least one occurrence of 11 over $\{0, 1\}$. [5 M]

Ans. Refer to Examples on Pages 3.14 to 3.15.

October 2018

1. Define ambiguous grammar. [1 M]

Ans. Refer to Section 3.5.

2. What are the types of grammar in Chomsky hierarchy? [1 M]

Ans. Refer to Section 3.3.

3. Convert the following grammar in CNF: [5 M]

$$S \rightarrow ABA$$

$$A \rightarrow aA/\epsilon \text{ (epsilon)}$$

$$B \rightarrow bB/\epsilon \text{ (epsilon)}$$

Ans. Refer to Section 3.7.1.

4. Convert the following grammar in GNF: [5 M]

$$S \rightarrow AB \mid A$$

$$A \rightarrow BS$$

$$B \rightarrow AI \mid I$$

Ans. Refer to Section 3.7.2.

5. Explain types of regular grammar. [2 M]

Ans. Refer to Section 3.8.

6. Construct CFG for: [4 M]

(a) $\{a^n b^m \mid n, m \geq 0\}$

(b) $\{a^n b^i c^j \mid n > i, i \geq 0\}$

Ans. Refer to Examples on Pages 3.14 to 3.15.

April 2019

1. Define context sensitive grammar. [1 M]

Ans. Refer to Section 3.3, Point (2).

2. State lemma 2 for converting a CFG to GNF. [1 M]

Ans. Refer to Section 3.7 and Examples on 3.31 to 3.33.

3. Construct CFG for the following: [5 M]

(i) $L_1 = \{0^n 1^n 2^m \mid n > 1, m > 0\}$

(ii) $L_2 = \{0^n 1^m \mid n, m > 0\}$

Ans. Refer to Examples on Pages 3.14 to 3.15.

4. Construct the following CFG into Chomsky Normal Form (CNF):

[5 M]

$$S \rightarrow A a B \mid a$$

$$A \rightarrow S B b \mid bA$$

$$B \rightarrow B a \mid b$$

Ans. Refer to Section 3.7.1.

5. Define the term: Derivation tree.

[1 M]

Ans. Refer to Section 3.2.1.

■ ■ ■

Pushdown Automata

Objectives ...

- To study Basic Concepts in Pushdown Automata
- To learn Construction of Pushdown Automata
- To understand Deterministic and Non-deterministic Pushdown Automata

4.0 INTRODUCTION

- We have seen finite automata (abstract machine) with the following dual property:
 1. For each regular language, there is at least one machine that runs successfully only on the input string from that language.
 2. For each machine in the class the set of words it accepts is a regular set.
- We are now considering a different class of languages but we want to answer the same questions, so we would again like to find a machine formulation.
- We are looking for a mathematical model of some class of machines that corresponds to CFLs i.e., there should be at least one machine that accepts each CFL and the language accepted by each machine is context-free. We want CFL-acceptor just as FA's are regular language recognizers and acceptors.
- To build these new machines, we start with our old FA's and add stack and input tape to make it more powerful. This FA with stack (LIFO) is called as Pushdown Automata (PDA).
- A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.
- The pushdown automaton is a finite automaton with an additional tape, which behaves like a stack. The PushDown Automata (PDA) is the machine format of the context-free language.
- Input tape is infinitely long in one direction to hold any possible input. The tape has a first location for the first letter of the input, then a second location and so on.
- The locations into which we put the input letters are called cells as shown in the Fig. 4.1.

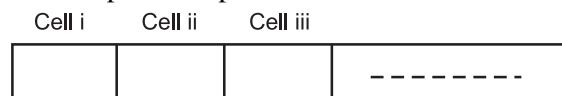


Fig. 4.1

- As we process this tape on the machine we read one letter at a time and eliminate each as it is used.
- When we reach the first blank cell we stop, i.e. rest of tape is also blank. We read from left to right and never go back to a cell that was read before.
- As a part of our new pictorial representation for FAs, let us introduce the symbols shown below.



Fig. 4.2

- An accept state is a final state and reject state is not a final. Read state is shown by diamond shaped boxes as shown below.

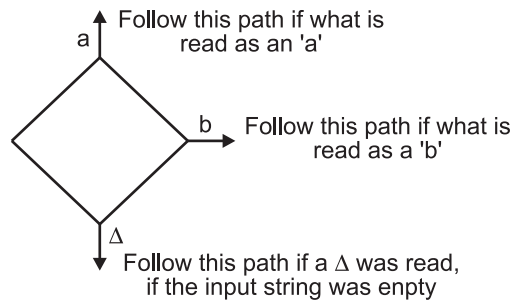


Fig. 4.3

(Note: Symbol Δ stands for blank).

- The finite automata that accepts all words ending in the letter a is as shown in the Fig. 4.4.

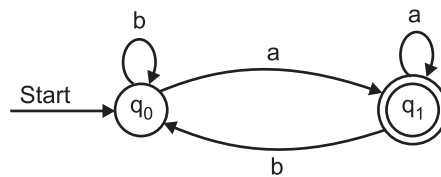


Fig. 4.4

- The FA in the new symbolism is shown in the Fig. 4.5.

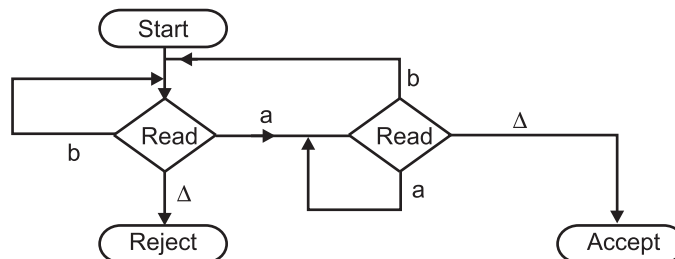


Fig. 4.5

- Our machine is still an FA. Here we chosen this representation because we now want to add additional component called pushdown stack (last-in-first-out) to our machine. The only stack operations allowed to us are push and pop.
- Popping an empty stack, like reading an empty tape, gives us the blank character Δ . We include the states as:

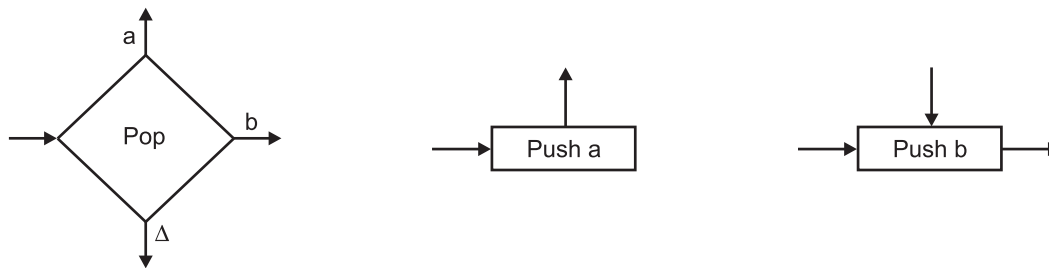


Fig. 4.6

- The edges coming out of a pop state are labeled in the same way as the edges from a read state.

Basic Concept of Pushdown Automata:

- Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages.
- The term "pushdown" refers to the fact that the stack can be regarded as being "pushed down" like a tray dispenser at a cafeteria, since the operations never work on elements other than the top element.
- Fig. 4.7 shows diagram for PDA. The components of it are described below:
 1. The **input tape** contains the input symbols. The tape is divided into a number of squares. Each square/block contains a single input character. The string placed in the input tape is traversed from left to right. The two end sides of the input string contain an infinite number of blank symbols.
 2. The **reading head** scans each square in the input tape and reads the input from the tape. The head moves from left to right. The input scanned by the reading head is sent to the finite control of the PDA.
 3. The **finite control** can be considered as a control unit of a PDA. An automaton always resides in a state. The reading head scans the input from the input tape and sends it to the finite control.
 4. A **stack** is a temporary storage of stack symbols. Every move of the PDA indicates one of the following to the stack
 - One stack symbol may be added to the stack (push)
 - One stack symbol may be deleted from the top of the stack (pop)

The stack is the component of the PDA which differentiates it from the finite automata. In the stack, there is always a symbol z_0 which denotes the bottom of the stack.

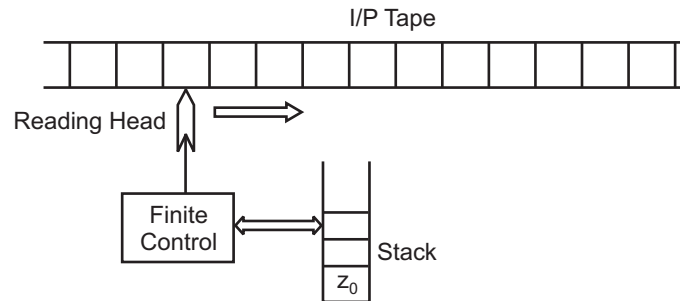


Fig. 4.7

Example: Consider the following PDA shown in the Fig. 4.8.

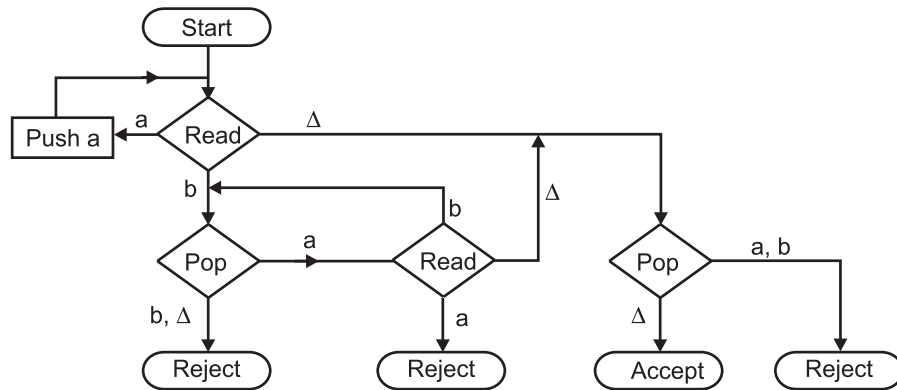
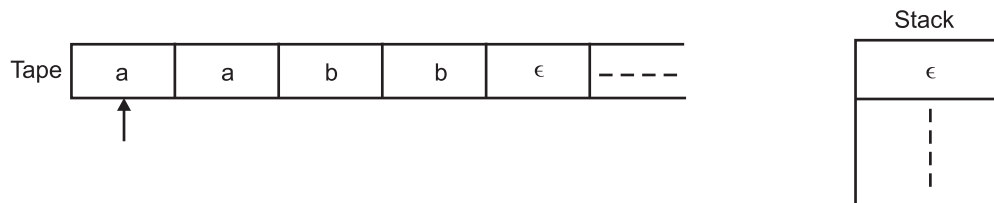


Fig. 4.8

Solution: The question is what language is represented by the above PDA? To find this, we will analyze the PDA. Let us consider input string aabb. We assume that this string has been put on the tape.

We must begin at start and then read the symbol 'a' from the input tape. Push a state tells us to push an a onto the stack as shown in Fig. 4.9 (b). We now read another a and proceed as before. See Fig. 4.9 (c).

After the second push a, we return back to same read state again. Here we read the letter b, we take the b edge out of this state down to left pop. The state pop takes the top element off the stack as shown in Fig. 4.9 (d). Now, next read symbol is b, it returns to pop state again and pop the top element off the stack as shown in Fig. 4.9 (e), leaving the stack empty. Next read is symbol Δ ; which will be the accept state. We observe that the language of words accepted by this machine is exactly $\{a^n b^n, n = 0, 1, 2, \dots\}$.



(a) Initial configuration

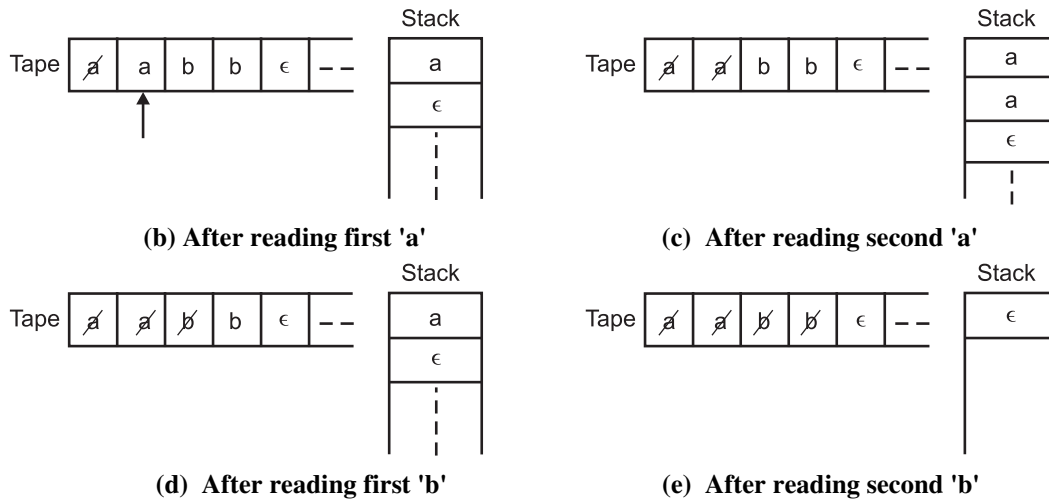


Fig. 4.9

This language is non-regular language, generated by CFG. We constructed machine (PDA) which accepts this language. So PDAs are more powerful than FA.

FA could not keep the track of how many times 'n' occurs $\{a^n b^n\}$. PDA has a primitive memory unit. It can keep track of how many a's are read at the beginning.

Difference between PDA and FA:

[April 16, 18, Oct. 16, 18]

1. **The length of the path formed by a given input:** If a string of seven letters is fed into an FA, it follows a path exactly seven edges long. In a PDA, the path could be longer or shorter.

For example, following PDA accepts only the language of all words beginning with an a.

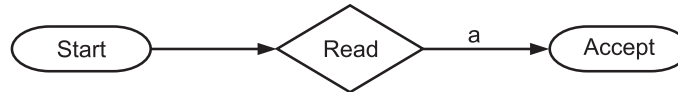


Fig. 4.10

No matter how long the input string, the path is only one or two edges long.

2. PDA accepts regular or non-regular language. FA accepts regular language.
3. PDA is more powerful than FA because it has unlimited memory units (stack); so it can remember arbitrarily long strings.
4. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

4.1 DEFINITION OF PUSHDOWN AUTOMATA [April 17, 19 Oct. 17]

- A pushdown automata M is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, f)$,

where,

Q is finite set of states.

Σ is an alphabet called the input alphabet.

Γ is an alphabet called stack alphabet.

q_0 is initial state and $q_0 \in Q$.

$Z_0: Z_0 \in \Gamma$ is a particular stack symbol called start symbol.

F : set of final states $F \subseteq Q$.

δ : is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

Moves:

1. The interpretation of $\delta(q, a, z) = \{(p_1, \gamma_1), (p_2, \gamma_2) \dots (p_m, \gamma_m)\}$, where q and $p_i \forall 1 \leq i \leq m$ are states, a is in Σ , Z is stack symbol and γ_i is in Γ^* where $1 \leq i \leq m$ i.e. a PDA in state q with input symbol a and Z is top of stack (TOS) $\forall i$ enter state p_i , replace the symbol Z by string γ_i and advance input head one symbol.
2. The interpretation of $\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots (p_m, \gamma_m)\}$ is that the PDA in state q , independent of the input symbol being scanned and with Z the TOS, can enter the state p_i and replace Z by $\gamma_i \forall i$ where $1 \leq i \leq m$. Here the input head is not advanced.

Instantaneous Description (ID):

[April 17]

- It is defined as the configuration of PDA at a given instant.
- It is denoted as (q, w, γ) where q is state, w is input string and γ is a string of stack symbols.
- If $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA we say $(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$ if $\delta(q, a, Z)$ contains (p, β) . We use \vdash for one or more than one moves.

Accepted Language:

[Oct. 17]

- There are two methods for defining languages accepted by PDA.
 1. **Acceptance by empty stack:** It is a set of all strings for which some sequence of moves causes PDA to empty its stack. We define, $N(M) = \{w \mid (q_0, w, Z_0) \vdash (p, \epsilon, \epsilon) \text{ for some } p \in Q\}$.
 2. **Acceptance by final state:** It is a set of all inputs for which some sequence of moves causes PDA to enter a final state. We define:

$$L(M) = \{w \mid (q_0, w, z_0) \vdash (p, \epsilon, \gamma) \text{ where } p \in F \text{ and } \gamma \in \Gamma^*\}$$

Example 1: Construct PDA for $L = \{a^n b^n \mid n \geq 1\}$.

Solution: We have already seen in example 5.1 that how it works. Let us put this using δ mapping and construct PDA M .

1. Initially PDA is in state q_0 and stack is empty.
2. We push 'A' on stack for a's and PDA remains in state q_0 .
3. When we get first 'b', we change the state into q_1 and pop the 'A' from stack.
4. If the input is b and state is q_1 , then pop the symbol A. Do not change the state.
5. If input string is ϵ and stack is empty, PDA accepts the string by empty stack.

\therefore PDA $M = (\{q_0, q_1\}, \{a, b\}, \{A\}, \delta, q_0, \epsilon, \phi)$, where δ is defined as

$$\delta(q_0, a, \epsilon) = (q_0, A)$$

$$\delta(q_0, a, A) = (q_0, AA)$$

$$\delta(q_0, b, A) = (q_1, \epsilon)$$

$$\delta(q_1, b, A) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, \epsilon) = (q_1, \epsilon)$$

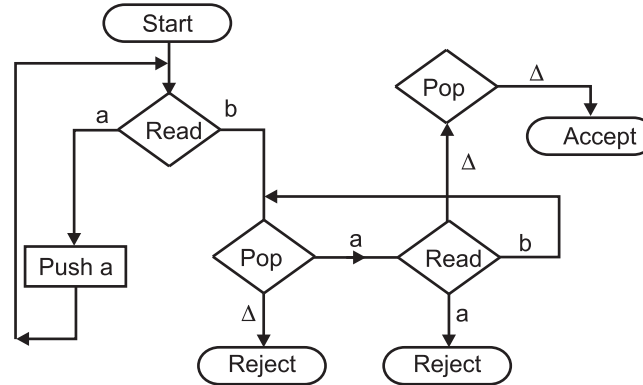


Fig. 4.11: PDA

Simulation for string 'aabb':

Consider the input string aabb. Instantaneous description is as follows:

$$\begin{aligned} (q_0, \underline{aabb}, \underline{\epsilon}) &\vdash (q_0, \underline{aabb}, \underline{A}) \\ &\vdash (q_0, \underline{bb}, \underline{AA}) \\ &\vdash (q_1, b, A) \\ &\vdash (q_1, \epsilon, \epsilon) = q_1 \text{ is accept state} \end{aligned}$$

Since the language is accepted by empty stack, $aabb \in L(M)$. The PDA is shown in Fig. 5.10.

Example 2: Construct PDA for $L = \{w cw^R \mid w \text{ in } (0 + 1)^*\}$.

Solution: To accept L we consider two states q_1 and q_2 and a stack on which you place blue, green and red plates. The device will operate by following rules:

1. The start state is q_1 and R (red) plate on the stack, so $Z_0 = R$.
2. If the input is 0 and the state is q_1 , then push blue plate on the stack. Do not change the state.
3. If the input is 1 and the state is q_1 , then push green plate on the stack. Do not change the state.
4. If the input is c and the state is q_1 , then change the state to q_2 , no plates are added or removed.
5. If the input is 0 and the state is q_2 with blue plate on the top of stack then pop the blue plate. If the input is 1 and the state is q_2 with green plate on the top of stack then pop the green plate. State remains in q_2 .

6. If the device is in state q_2 and a red plate is on the top of stack, then plate is popped and stack is empty.

We write PDA as follows:

$$\text{PDA } M = (\{q_1, q_2\}, \{0, 1, c\}, \{R, B, G\}, \delta, q_1, R, \phi)$$

where δ is $\delta(q_1, 0, R) = \{(q_1, BR)\}$

$$\delta(q_1, 0, B) = \{(q_1, BB)\}$$

$$\delta(q_1, 0, G) = \{(q_1, BG)\}$$

$$\delta(q_1, 1, G) = \{(q_1, GG)\}$$

$$\delta(q_1, 1, B) = \{(q_1, GB)\}$$

$$\delta(q_1, c, R) = \{(q_2, R)\}$$

$$\delta(q_1, c, B) = \{(q_2, B)\}$$

$$\delta(q_1, c, G) = \{(q_2, G)\}$$

$$\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$$

$$\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$$

$$\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$$

Consider the string "01c10".

The sequence of moves is as follows:

$$\begin{aligned} (q_1, 01c10, R) &\vdash (q_1, 1 \underline{c} 10, \underline{B}R) \text{ as } \delta(q_1, 0, R) = \{(q_1, BR)\} \\ &\vdash (q_1, \underline{c} 10, \underline{G}BR) \text{ as } \delta(q_1, c, G) = \{(q_2, G)\} \\ &\vdash (\underline{q}_2, \underline{1}0, \underline{G}BR) \text{ as } \delta(q_2, 1, G) = \{(q_2, \epsilon)\} \\ &\vdash (q_2, \underline{0}, \underline{B}R) \text{ as } \delta(q_2, 0, B) = \{(q_2, \epsilon)\} \\ &\vdash (q_2, \epsilon, R) \text{ as } \delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\} \\ &\vdash (q_2, \epsilon, \epsilon) \end{aligned}$$

The pictorial representation of PDA is shown in the Fig. 4.12.

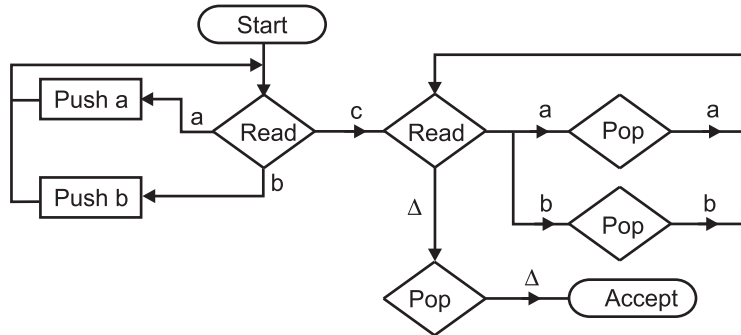


Fig. 4.12

4.2 CONSTRUCTION OF PDA

[April 16, Oct. 16]

- In this section we will study construction of PDA.

4.2.1 Acceptance by Final State

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA. Then $L(P)$, the language accepted by P by final state is $\{w \mid (q_0, w, z_0) \vdash^* (q, \epsilon, \alpha)\}$, where q is some state in F and stack is not empty (say α).

Example: $L = \{ww^R \mid w \in (0+1)^*\}$

Solution: We have to show the accepting computation of P .

If string

$$\begin{aligned} S &= ww^R \text{ then} \\ (q_0, ww^R, z_0) &\vdash^* (q_0, w^R, w^R z_0) \\ &\vdash (q_1, w^R, w^R z_0) \vdash^* (q_1, \epsilon, z_0) \vdash^* (q_2, \epsilon, z_0) \end{aligned}$$

Here one option is to read from its inputs and store it on its stack, in reverse. After reading of w , (in reverse), state is changed from q_0 and q_1 . At state q_1 , it matches w^R on the input with the same string on its stack and finally enter in state q_2 .

So P accepts by final state.

4.2.2 Acceptance by Empty Stack

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ we define $N(P) = \{w \mid (q_0, w, z_0) \vdash^* (q, \epsilon, \epsilon)\}$ Where, q is any state and $N(P)$ is the set of inputs w that P can consume and at the same time empty its stack.

Example:

$$L = \{ww^R \mid w \in (0+1)^*\}$$

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi) \text{ stack is empty}$$

We define

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{\overline{0}, \overline{1}, z_0\}, \delta, q_0, z_0, \phi)$$

where δ is

$$1. \quad \delta(q_0, 0, z_0) = \{(q_0, 0, z_0)\}$$

$$\text{and} \quad \delta(q_0, 1, z_0) = \{(q_0, 1, z_0)\}$$

we read first input symbol and push into the stack.

$$2. \quad \left. \begin{aligned} \delta(q_0, 0, 0) &= \delta(q_0, 00) \\ \delta(q_0, 0, 1) &= \delta(q_0, 01) \\ \delta(q_0, 1, 0) &= \delta(q_0, 10) \\ \delta(q_0, 1, 1) &= \delta(q_0, 11) \end{aligned} \right\} \text{ use for pushing each onto the top of the stack}$$

$$3. \quad \delta(q_0, \epsilon, z_0) = \{(q_1, z_0)\}$$

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

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

This three rules allow P to go from state q_0 to state q_1 simultaneously (on ϵ input), leaving intact whatever symbol is at the top of the stack.

$$4. \quad \delta(q_1, 0, 0) = (q_1, \epsilon)$$

$$\delta(q_1, 1, 1) = (q_1, \epsilon)$$

In state q_1 , match input symbols against the top symbols on the stack. Then pop the symbol.

5. Pop all symbols until z_0 marker we get. Then pop z_0 and stack is empty.

4.3

DETERMINISTIC AND NON-DETERMINISTIC PDA (DPDA AND NPDA)

[April 17, Oct. 17]

- We have seen that in case of FA, DFA or NFA are equivalent. Similarly, we have deterministic and non-deterministic PDA (DPDA and NPDA) but they are not equivalent.
- A language accepted by a NPDA may not be accepted by a DPDA. Thus, for every NPDA there may not exist an equivalent DPDA.
- A PDA is said to be a Deterministic PushDown Automata (DPDA) if all derivations in the design give only a single move.
- A pushdown automata is called Non-deterministic PushDown Automata (NPDA) if one of the derivations generates more than one move.
- If a PDA being in a state with a single input and a single stack symbol gives a single move, then the PDA is called DPDA. If a PDA being in a state with a single input and a single stack symbol gives more than one move for any of its transitional functions, then the PDA is called NPDA.
- The Venn diagram shown in the Fig. 4.13 gives the relative powers of the machines DFA, NFA, DPDA and NPDA.

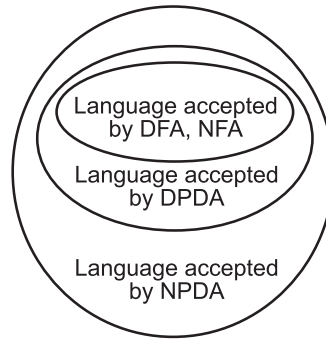


Fig. 4.13: Venn Diagram

- Thus though DFA and NFA have equal powers, NPDA and DPDA have different capabilities. NPDA can accept any CFL but DPDA is a special case of CFL, NPDA accepts.
- Thus, the PDAs that are equivalent to CFGs is the class of NPDAs.
So, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if
 1. for each q in Q and Z in Γ , whenever $\delta(q, \epsilon, Z)$ is non-empty then $\delta(q, a, Z)$ is empty for all $a \in \Sigma$.
 2. for no q in Q , Z in Γ and a in $\Sigma \cup \{\epsilon\}$ does $\delta(q, a, Z)$ contains more than one element.
- If any of the above conditions is not satisfied then PDA is a Non-deterministic PDA (NPDA).

Example 1: Consider PDA for language $L = \{ww^R \mid w \in (a + b)^*\}$.

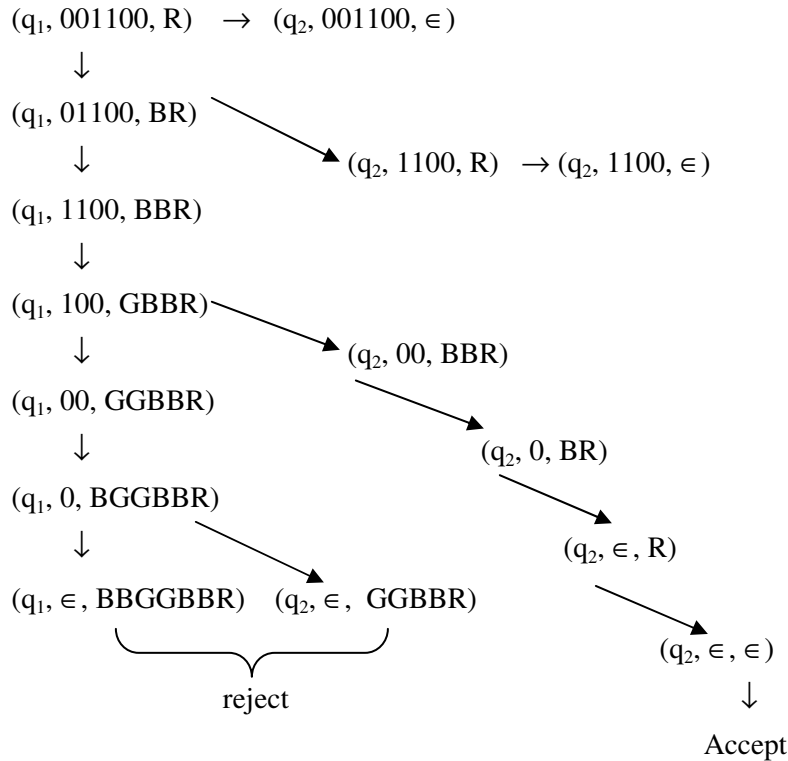
Solution: To construct NPDA, we define

$$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G\}, \delta, q_1, R, \phi)$$

where, δ is

$$\begin{aligned} \delta(q_1, 0, R) &= \{(q_1, BR)\} \\ \delta(q_1, 1, R) &= \{(q_1, GR)\} \\ \delta(q_1, 0, B) &= \{(q_1, BB) (q_2, \epsilon)\} \\ \delta(q_1, 0, G) &= \{(q_1, BG)\} \\ \delta(q_1, 1, B) &= \{(q_1, GB)\} \\ \delta(q_1, 1, G) &= \{(q_1, GG), (q_2, \epsilon)\} \\ \delta(q_2, 0, B) &= \{(q_2, \epsilon)\} \\ \delta(q_2, 1, G) &= \{(q_2, \epsilon)\} \\ \delta(q_1, \epsilon, R) &= \{(q_2, \epsilon)\} \\ \delta(q_2, \epsilon, R) &= \{(q_2, \epsilon)\} \end{aligned}$$

This is NPDA since moves for string 001100 are as follows:



Input string 001100 is accepted by NPDA, since there is one sequence of moves which empties the stack.

EXAMPLES USING STACK METHODS

Example 1: Construct PDA to accept a string of properly balanced parenthesis over the set of alphabet $\{\{, (, [, \},), \}\}$.

Solution: We push A, B, C for {, (, [respectively and consider red (R) at bottom of the stack.

Let, $M = (\{q_0, q_1\}, \{\{, (, [, \},), \}\}, \{A, B, C, R\}, \delta, q_0, R, \phi)$

where δ is defined as

$$\delta(q_0, \{, R) = (q_0, AR)$$

$$\delta(q_0, (, R) = (q_0, BR)$$

$$\delta(q_0, [, R) = (q_0, CR)$$

$$\delta(q_0, \{, A) = (q_0, AA)$$

$$\delta(q_0, \{, B) = (q_0, AB)$$

$$\delta(q_0, \{, C) = (q_0, AC)$$

$$\delta(q_0, (, A) = (q_0, BA)$$

$$\delta(q_0, (, B) = (q_0, BB)$$

$$\delta(q_0, (, C) = (q_0, BC)$$

$$\delta(q_0, [, A) = (q_0, CA)$$

$$\delta(q_0, [, B) = (q_0, CB)$$

$$\delta(q_0, [, C) = (q_0, CC)$$

$$\delta(q_0, \}, A) = (q_0, \epsilon)$$

$$\delta(q_0,), B) = (q_0, \epsilon)$$

$$\delta(q_0,], C) = (q_0, \epsilon)$$

$$\delta(q_0, \epsilon, R) = (q_1, \epsilon)$$

Simulation of string "{([])}":

$$\begin{aligned} (q_0, \{([])\}, R) &\vdash (q_0, ([])\}, AR) \\ &\vdash (q_0, [])\}, BAR) \\ &\vdash (q_0,])\}, (BAR) \\ &\vdash (q_0,)\}, BAR) \\ &\vdash (q_0, \}, AR) \\ &\vdash (q_0, \epsilon, R) \\ &\vdash (q_1, \epsilon, \epsilon) = \text{accept} \end{aligned}$$

Example 2: Construct PDA for language $L = \{0^m 1^n 2^k \mid m, n, k \geq 1, m = n + k\}$.

Solution: We push B for each 0 and when we get first 1 start popping plates and change the state. For first occurrence of 2, we have to change the state because order is important.

PDA $M = (\{q_0, q_1, q_2\}, \{0, 1, 2\}, \{R, B\}, \delta, q_0, R, \phi)$, where δ is defined as follows:

$$\delta(q_0, 0, R) = (q_0, BR)$$

$$\delta(q_0, 0, B) = (q_0, BB)$$

$$\delta(q_0, 1, B) = (q_1, \epsilon)$$

$$\delta(q_1, 1, B) = (q_1, \epsilon)$$

$$\delta(q_1, 2, B) = (q_2, \epsilon)$$

$$\delta(q_2, 2, B) = (q_2, \epsilon)$$

$$\delta(q_2, \epsilon, R) = (q_2, \epsilon)$$

Simulation of string “0012”:

$$\begin{aligned} (q_0, 0012, R) &\vdash (q_0, 012, BR) \\ &\vdash (q_0, 12, BB) \\ &\vdash (q_1, 2, B) \\ &\vdash (q_2, \epsilon) = \text{accept} \end{aligned}$$

Example 3: Construct a PDA to check the well-formedness of parenthesis.

Solution: The PDA will scan the input symbols and everytime it finds an opening bracket '(', it will push A onto the stack. Whenever a closing bracket ')' is found, the PDA will pop A from the stack.

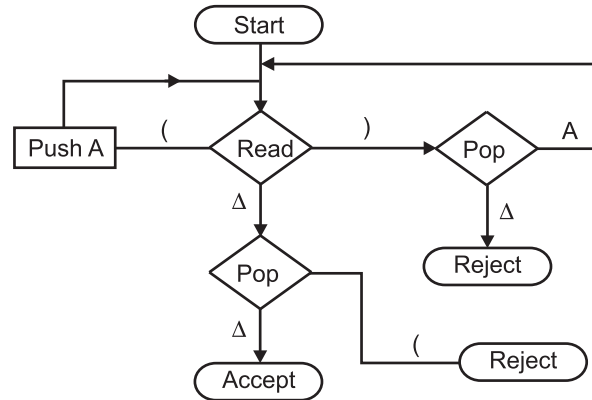


Fig. 4.14

$$\delta(q_0, (, R) = (q_0, AR)$$

$$\delta(q_0, (, A) = (q_0, AA)$$

$$\delta(q_0,), A) = (q_1, \epsilon)$$

$$\delta(q_1,), A) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, R) = (q_1, \epsilon)$$

Example 4: Design a PDA which accepts a language $L = \{a^n b^{2n+1} \mid n \geq 1\}$.

Solution: Each string in the language will be of the form $a^n b^{2n+1}$.

PDA reads and push all a's. When it reads the first b, it recognises that all a's are over. So hereafter for every two b's in the string it should pop one a from the stack. At the end, it should find only one b and accept the strings. PDA is as shown in the Fig. 4.15.

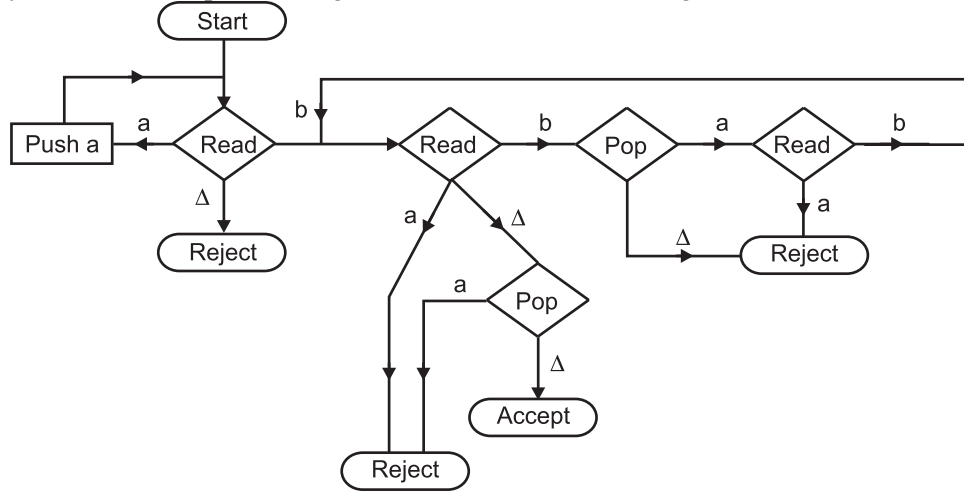


Fig. 4.15

We construct PDA $M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, (R, A), \delta, q_0, R, \phi)$, where δ is defined as follows:

$$\begin{aligned}
 \delta(q_0, a, R) &= (q_0, AR) \\
 \delta(q_0, a, A) &= (q_0, AA) \\
 \delta(q_0, b, A) &= (q_1, A) \\
 \delta(q_1, b, A) &= (q_2, \epsilon) \\
 \delta(q_2, b, A) &= (q_1, A) \\
 \delta(q_2, b, R) &= (q_3, \epsilon) \\
 \delta(q_2, \epsilon, R) &= (q_3, \epsilon)
 \end{aligned}
 \quad \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \text{ loop}$$

Simulation of string “aabbbbb”:

$$\begin{aligned}
 (q_0, aabbbbb, R) &\vdash (q_0, abbbbbb, AR) \\
 &\vdash (q_0, bbbbbb, AAR) \\
 &\vdash (q_1, bbbbbb, AAAR) \\
 &\vdash (q_1, bbbb, AAR) \\
 &\vdash (q_2, bb, AR) \\
 &\vdash (q_2, b, R) \\
 &\vdash (q_3, \epsilon) = \text{accept}
 \end{aligned}$$

Example 5: Construct a PDA that accepts the language generated by CFG,

$$S \rightarrow S + S \mid S * S \mid 2$$

[April 16]

Solution:

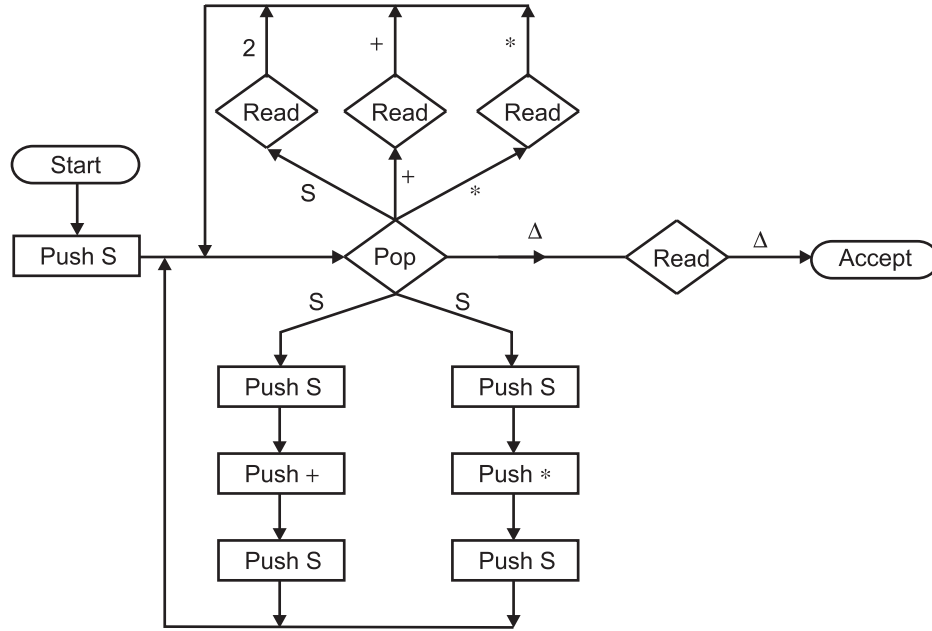


Fig. 4.16

Example 6: Construct PDA for $L = \{0^m 1^n 2^{n+2} 3^m \mid n, m \geq 1\}$

Solution: $L = \{0^m 1^n 2^{n+2} 3^m \mid n, m \geq 1\}$

is written as, $\{0^m 1^n 2^n 2 2 3^m \mid n, m \geq 1\}$

In this language total number of 0's and total number of 3's are same.

Also total number of 1's and total number of 2's are same and two 2's are extra.

Steps:

1. All 0's are push into the stack at state q_0 .
2. All 1's are push into the stack at state q_1 .
3. Read extra two 2's without changing the stack.
4. Read one 2 and pop one 1 from the stack.
5. Pop will continue until we encounter 3.
6. When 3 is read, then for each 3 read, pop 0 from stack.
7. When string is over and stack is empty, then the string is accepted.

The PDA is

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$$

where

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

$$\Sigma = \{0, 1, 2, 3\}$$

$$\Gamma = \{0, 1, R\}$$

and δ mapping is

$$\delta(q_0, 0, R) = (q_0, 0R) \rightarrow \text{Push 1st 0}$$

$$\delta(q_0, 0, 0) = (q_0, 00) \rightarrow \text{Push all 0's}$$

$$\delta(q_0, 1, 0) = (q_1, 10) \rightarrow \text{Push first 1}$$

$$\delta(q_1, 1, 1) = (q_1, 11) \rightarrow \text{Push all 1's}$$

$$\delta(q_1, 2, 1) = (q_2, 1) \rightarrow \text{Change state without push or pop}$$

$$\delta(q_2, 2, 1) = (q_3, 1) \rightarrow \text{Change state without push or pop}$$

$$\delta(q_3, 2, 1) = (q_3, \epsilon)$$

$$\delta(q_3, 3, 0) = (q_4, \epsilon)$$

$$\delta(q_4, 3, 0) = (q_4, \epsilon)$$

$$\delta(q_4, \epsilon, R) = (q_4, \epsilon) = \text{accept}$$

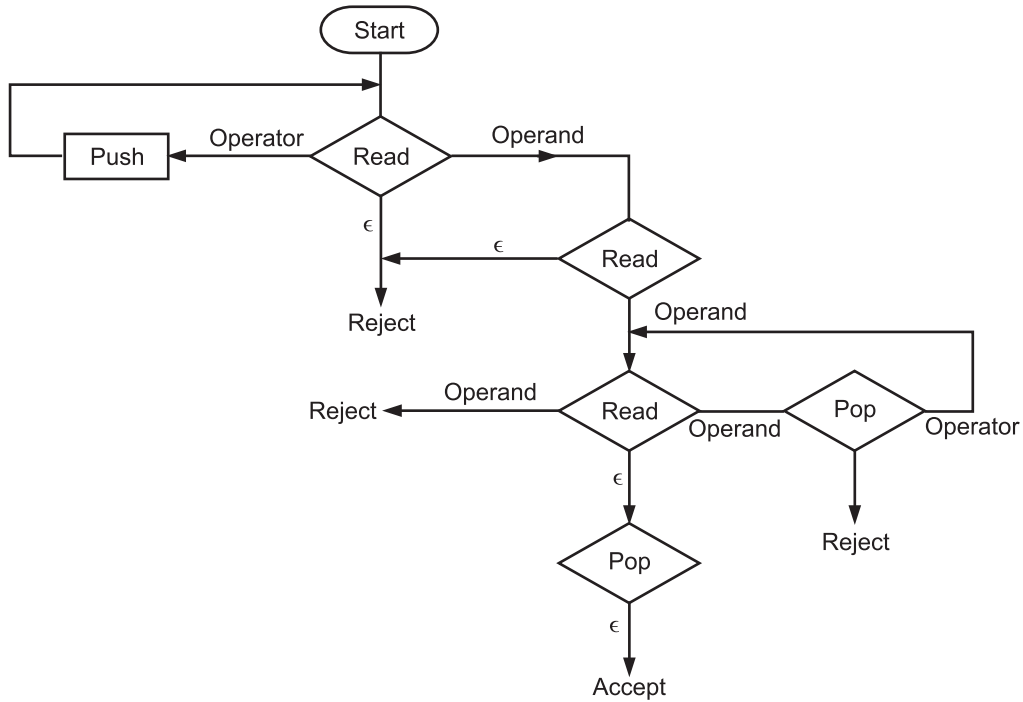
Simulation of string 012223:

$$\begin{aligned} (q_0, \underline{0}12223, R) &\vdash (q_0, \underline{1}2223, \underline{0}R) \\ &\vdash (q_1, \underline{2}223, \underline{1}0R) \\ &\vdash (q_2, \underline{2}23, \underline{1}0R) \\ &\vdash (q_3, \underline{2}3, \underline{1}0R) \\ &\vdash (q_3, \underline{3}, \underline{0}R) \\ &\vdash (q_4, \epsilon, R) \\ &\vdash (q_4, \epsilon) = \text{accept} \end{aligned}$$

Example 7: Construct PDA to check whether given string is correct prefix expression or not?

Solution: Logic:

- All operators are push into the stack.
 - When operands are read, pop operator from stack.
-

**Fig. 4.17**

PDA is

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$$

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

$$\Sigma = \{\text{operator}, \text{operand}\}$$

$$\Gamma = \text{operator}, z_0$$

 δ is

$$\delta(q_0, \text{operator}, z_0) = (q_0, \text{operator}, z_0)$$

$$\delta(q_0, \text{operator}, \text{operator}) = (q_0, \text{operator}, \text{operator})$$

$$\delta(q_0, \text{operand}, \text{operator}) = (q_1, \text{operator})$$

(By reading 1st operand do not change the stack)

$$\delta(q_1, \text{operand}, \text{operator}) = (q_2, \epsilon)$$

$$\delta(q_2, \text{operand}, \text{operator}) = (q_3, \epsilon)$$

$$\delta(q_3, \epsilon, z_0) = (q_3, \epsilon)$$

Simulation of string “+ * a b c”:

$$\delta(q_0, + * a b c, \underline{z_0}) \vdash (q_0, * a b c, + z_0)$$

$$\vdash (q_0, \underline{a} b c, * + z_0)$$

$$\vdash (q_1, b c, * + z_0)$$

$$\vdash (q_2, c, + z_0)$$

$$\vdash (q_3, \epsilon, z_0)$$

$$\vdash (q_3, \epsilon) \text{ string is accepted}$$

Example 8: Construct PDA to check whether given equation is in postfix form or not.

Solution: Logic:

- All operands are push into the stack.
- When operator is read, two operands are popped from the stack.

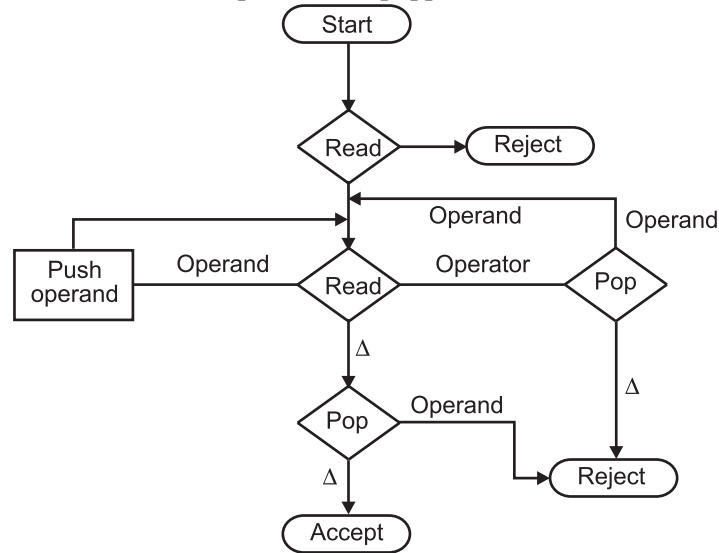


Fig. 4.18

PDA is
where

$$\begin{aligned}
 M &= (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi) \\
 Q &= \{q_0, q_1, q_2, q_3\} \\
 \Sigma &= \{\text{operator}, \text{operand}\} \\
 \Gamma &= \{\text{operand}, z_0\}
 \end{aligned}$$

δ is

$$\begin{aligned}
 \delta(q_0, \text{operand}, z_0) &= (q_1, z_0) \\
 \delta(q_1, \text{operand}, z_0) &= (q_1, \text{operand}, z_0) \\
 \delta(q_1, \text{operand}, \text{operand}) &= (q_1, \text{operand}, \text{operand}) \\
 \delta(q_1, \text{operator}, \text{operand}) &= (q_1, \epsilon) \\
 \delta(q_1, \epsilon, \text{operand}) &= (q_2, \epsilon) \\
 \delta(q_2, \epsilon, z_0) &= (q_2, \epsilon) = \text{accept}
 \end{aligned}$$

Simulation of string "a b * c +": The string is operand operand operator operand operator.

$$\begin{aligned}
 \delta(q_0, \underline{a} \text{ b * c +}, z_0) &\vdash (q_1, \underline{a} \text{ b * c +}, \underline{z_0}) \\
 &\vdash (q_1, \underline{b} \text{ * c +}, \underline{a} \text{ } z_0) \\
 &\vdash (q_2, \underline{*} \text{ c +}, \underline{b} \text{ a } z_0) \\
 &\vdash (q_1, \underline{c} \text{ +}, \underline{a} \text{ } z_0) \\
 &\vdash (q_1, \underline{+} \text{ c a } z_0) \\
 &\vdash (q_1, \epsilon, \text{a } z_0) \\
 &\vdash (q_2, \epsilon, z_0) \\
 &\vdash (q_2, \epsilon) = \text{accept}
 \end{aligned}$$

Example 9: Construct PDA to accept all those strings containing equal number of a's and b's.

Solution: $L = \{ab, abab, abba, aabb, bbaa, baba, \dots\}$

Logic:

- When a is read at state q_0 , push a and change the state to q_1 . If b is read, then push b and change the state to q_2 .
- At state q_1 , a is read, then push a. At state q_2 , b is read, then push b.
- When the PDA is at q_1 state means a is already read and current input symbol is b then pop a.
- When the PDA is at state q_2 and it reads a then pop b.
- When string is over and stack is empty, then language is accepted.

PDA is,

$$\delta(q_0, a, z_0) = (q_1, az_0)$$

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

$$\delta(q_1, b, z_0) = (q_2, bz_0)$$

$$\delta(q_0, b, z_0) = (q_2, bz_0)$$

$$\delta(q_2, b, b) = (q_2, bb)$$

$$\delta(q_2, a, b) = (q_2, \epsilon)$$

$$\delta(q_2, a, z_0) = (q_1, az_0)$$

$$\delta(q_2, \epsilon, z_0) = (q_3, \epsilon)$$

$$\delta(q_3, \epsilon, \epsilon) = (q_3, \epsilon) = \text{accept}$$

Simulation of string "abba":

$$\delta(q_0, \underline{a}bba, \underline{z}_0) \vdash (q_1, \underline{b}ba, \underline{a}z_0)$$

$$\vdash (q_1, \underline{b}a, \underline{z}_0)$$

$$\vdash (q_2, \underline{a}, \underline{b}z_0)$$

$$\vdash (q_2, \epsilon, \underline{z}_0)$$

$$\vdash (q_3, \epsilon, \epsilon) = \text{accept}$$

Example 10: Show that $\{a^n b^n \mid n \geq 1\} \cup \{a^m b^{2m} \mid m \geq 1\}$ cannot be accepted by a deterministic PDA. **OR** It is NPDA.

Solution: $L = \{ab, aabb, abb, aabbbb, \dots\}$

The union of a language containing strings of equal number of a's followed by equal number of b's or double b's for single a.

Logic:

- PDA push all a's.
- When it reads the first b, it recognize that a's are over. Then we either push 2nd b, or we pop 'a' to check equal a's and b's.

PDA is,

$$\begin{aligned}\delta(q_0, a, z_0) &= (q_0, Az_0) \\ \delta(q_0, a, A) &= (q_0, AA) \\ \delta(q_0, b, A) &= \{(q_1, A) (q_2, \epsilon)\} \\ \delta(q_1, b, A) &= (q_2, \epsilon) \\ \delta(q_2, b, A) &= (q_1, A) \\ \delta(q_2, b, z_0) &= \{(q_3, \epsilon) (q_2, \epsilon)\} \\ \delta(q_2, \epsilon, z_0) &= (q_3, \epsilon)\end{aligned}$$
Simulation of “abb”:

$$\begin{aligned}\delta(q_0, \underline{a}bb, \underline{z_0}) &\vdash (q_0, \underline{b}b, \underline{Az_0}) \\ &\vdash (q_1, b, Az_0) \\ &\vdash (q_2, \epsilon, z_0) \\ &\vdash (q_3, \epsilon) = \text{accept}\end{aligned}$$
Simulation of “ab”:

$$\begin{aligned}\delta(q_0, \underline{a}b, \underline{z_0}) &\vdash (q_0, \underline{b}, \underline{Az_0}) \\ &\swarrow \quad \searrow \\ (q_1, \underline{\epsilon}, \underline{AAz_0}) &\quad (q_2, \epsilon, z_0) \\ \swarrow \quad \searrow &\quad \searrow \\ \text{rejected} &\quad (q_3, \epsilon, \epsilon) \\ &\quad \text{accepted}\end{aligned}$$

The given PDA is NPDA.

Example 11: Construct PDA for language $L = \{0^n 1^m 2^{n+m} \mid n, m \geq 1\}$

Solution: Logic:

- Read all 0's at state q_0 and push A onto stack.
- Read all 1's at state q_1 and push B onto stack.
- When 2 is read at state q_1 , POP A and B from the stack by changing the state q_3 .
- At state q_4 , language is accepted.

$$\begin{aligned}\delta(q_0, 0, z_0) &= (q_0, Az_0) \\ \delta(q_0, 0, A) &= (q_0, AA) \\ \delta(q_0, 1, A) &= (q_1, BA) \\ \delta(q_1, 1, B) &= (q_1, BB)\end{aligned}$$

$$\begin{aligned}\delta(q_1, 2, B) &= (q_2, \epsilon) \\ \delta(q_2, 2, A) &= (q_3, \epsilon) \\ \delta(q_3, 2, A) &= (q_3, \epsilon) \\ \delta(q_3, \epsilon, \epsilon) &= (q_4, \epsilon)\end{aligned}$$

Example 12: Design a PDA for checking the acceptance of string $a^n b^n c^n$.

Solution: This is two stack problem. We require two stack for counting a and b, then b and c. So it is double check problem. If we are using a single stack, then we cannot solve such a problem because we can push all a's in the stack which will get popped on b. By the time we reach c, the stack will be empty and we do not have anything to compare. We only check equal number of a's and b's.

Using two stack all a's can be push into one stack and all b's can be push into other stack. When we read c, we can pop one symbol from each stack. When we finish reading input, both the stacks should be empty and then the string can be accepted.

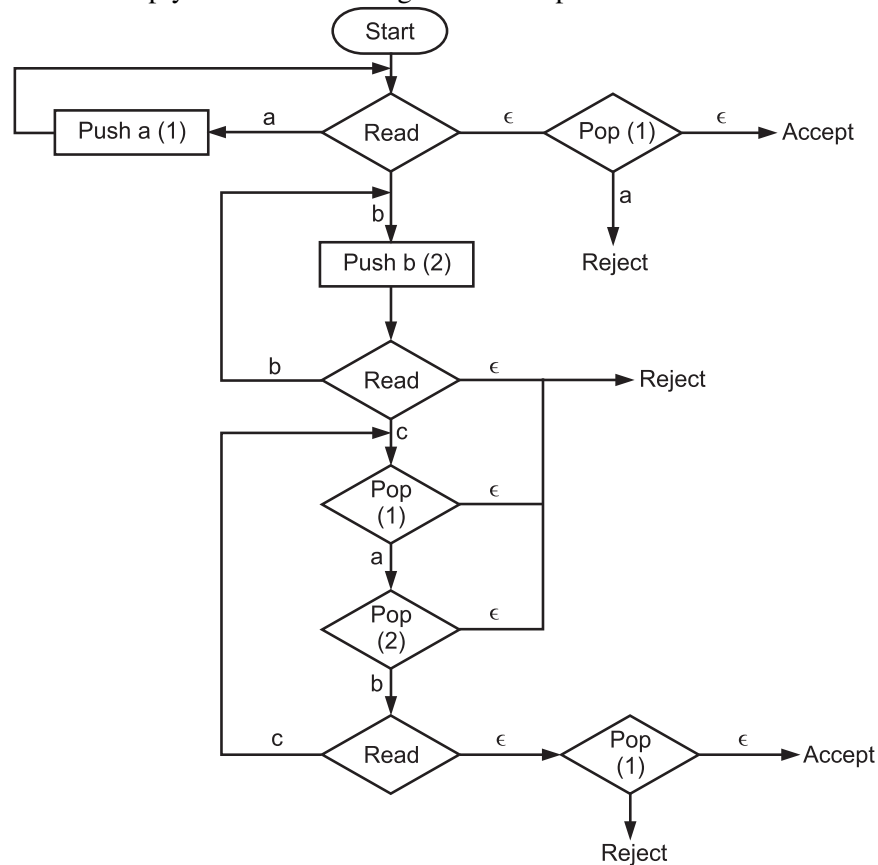


Fig. 4.19

Two stack PDA for $L = \{a^n b^n c^n \mid n \geq 0\}$

PDA with two stack is more powerful than PDA with one stack. But some languages (context-sensitive) are not accepted by PDA. In the above figure (1) represents stack 1 and (2) represents stack (2).

4.4 CREATING FILES CFG (IN GNF) TO PDA

[April 16]

- PDA is the machine accepting a context-free language. A context-free language is generated from context-free grammar. This section describes the process of generating a PDA from the CFG.
- **Theorem 1:** Given a language L generated by a particular CFG, there is a PDA that accepts exactly L .
- **Theorem 2:** Given a language L that is accepted by a certain PDA, there exists a CFG that generates exactly L .

Construction of PDA from the Given CFG:

- We will assume that the given CFG does not consist of any productions of the form $S \rightarrow \epsilon$.

Step 1: Convert the grammar into GNF.

Step 2: Say, if $S \rightarrow aSB \mid aB$ then

$$\begin{array}{ccc} \delta(q_0, & a, & S) = \{(q_0, SB) (q_0, B)\} \\ \downarrow & \downarrow & \downarrow \\ \text{State} & \text{Input} & \text{LHS} \\ & \text{symbol} & \text{non-terminal} \end{array}$$

Say, if $S \rightarrow a$ then $\delta(q_0, a, S) = (q_0, \epsilon)$.

EXAMPLES

Example 1: Construct a PDA for CFG

$$S \rightarrow aAA$$

$$A \rightarrow aS \mid bS \mid a$$

Solution: We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, A\}, \delta, q_0, S, \phi)$, where δ is defined as follows:

$$\delta(q_0, a, S) = \{(q_0, AA)\}$$

$$\delta(q_0, a, A) = \{(q_0, S) (q_0, \epsilon)\}$$

$$\delta(q_0, b, A) = \{(q_0, S)\}$$

Consider a string derived from CFG, "abaaaa" and find whether it is accepted by our constructed PDA by ID.

$$(q_0, abaaaa, S) \vdash (q_0, baaaa, AA)$$

$$\vdash (q_0, aaaa, SA) \vdash (q_0, aaa, AAA)$$

$$\vdash (q_0, aa, AA) \vdash (q_0, a, A) \vdash (q_0, \epsilon, \epsilon)$$

Therefore, the string which is accepted by CFG is also accepted by PDA.

Example 2: Construct a PDA equivalent to CFG, $S \rightarrow aSa \mid bSb \mid a \mid b$.

Solution: Converting CFG into GNF, we get,

$$\begin{aligned} S &\rightarrow aSA \\ S &\rightarrow bSB \\ S &\rightarrow a \\ S &\rightarrow b \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

We construct the PDA.

$M = (\{q_0\}, \{a, b\}, \{S, AB\}, \delta, q_0, S, \phi)$, where δ is defined as follows:

$$\begin{aligned} \delta(q_0, a, S) &= \{(q_0, SA) (q_0, \epsilon)\} \\ \delta(q_0, b, S) &= \{(q_0, SB) (q_0, \epsilon)\} \\ \delta(q_0, a, A) &= \{(q_0, \epsilon)\} \\ \delta(q_0, b, B) &= \{(q_0, \epsilon)\} \end{aligned}$$

Consider a string aababaa and find whether it is accepted by above PDA.

$$\begin{aligned} (q_0, aababaa, S) &\vdash (q_0, ababaa, SA) \\ &\vdash (q_0, babaa, SAA) \vdash (q_0, abaa, SBAA) \\ &\vdash (q_0, baa, BAA) \vdash (q_0, aa, AA) \\ &\vdash (q_0, a, A) \vdash (q_0, \epsilon, \epsilon) \end{aligned}$$

Example 3: Construct a PDA that accepts language as $S \rightarrow aS \mid aSbS \mid a$.

Solution: Convert CFG into GNF.

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow aSbS \\ S &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

We construct PDA.

$M = (\{q_0\}, \{a, b\}, \{S, B\}, \delta, q_0, S, \phi)$, where δ is defined as follows:

$$\begin{aligned} \delta(q_0, a, S) &= \{(q_0, S) (q_0, SBS) (q_0, \epsilon)\} \\ \delta(q_0, b, B) &= \{(q_0, \epsilon)\} \end{aligned}$$

Consider a string aaaba derived from CFG. We will see whether this string is accepted by above PDA or not.

$$\begin{aligned} (q_0, aaaba, S) &\vdash (q_0, aaba, S) \vdash (q_0, aba, SBS) \\ &\vdash (q_0, ba, BS) \vdash (q_0, a, S) \vdash (q_0, \epsilon, \epsilon) \end{aligned}$$

String "aaaba" is accepted.

Example 4: Show with example that if the language is accepted by CFG then the same language is accepted by PDA.

Solution: Consider CFG in CNF.

$$S \rightarrow SB$$

$$S \rightarrow AB$$

$$A \rightarrow CC$$

$$B \rightarrow b$$

$$C \rightarrow a$$

Let, $\Gamma = \{S, A, B, C\}$

$$\Sigma = \{a, b\}$$

The stack contains the symbol S and always contains non-terminals. There are two possibilities (NPDA): either we replace the removed non-terminal by two non-terminals or we do not replace non-terminal at all but we go to a read state to read the terminal from the tape.

Consider string "aab" generated by leftmost derivation.

$$S \Rightarrow AB \Rightarrow CCB \Rightarrow aCB \Rightarrow aaB \Rightarrow aab.$$

Now, we shall trace how the word "aab" can be accepted by PDA.

Leftmost Derivation	State	Stack	Tape
	Start	ϵ	aab
S	PUSH S	S	aab
	POP (S)	ϵ	aab
	PUSH B	B	aab
$S \Rightarrow AB$	PUSH A	AB (\because TOS = A)	aab
	POP (A)		aab
	PUSH C		aab
$A \Rightarrow CC$	PUSH C	CB	aab
	PUSH C	CCB	aab
	POP (C)	CB	aab
	Read a	CB	aab
	POP (C)	B	aab
	Read a	B	aab
	POP (B)	ϵ	aab
	Read b	ϵ	aab
	ACCEPT	ϵ	ϵ

Both stack and tape are empty. Thus language is accepted by empty stack or PDA, $N(M)$. Thus, we construct a PDA as shown in the Fig. 4.20.

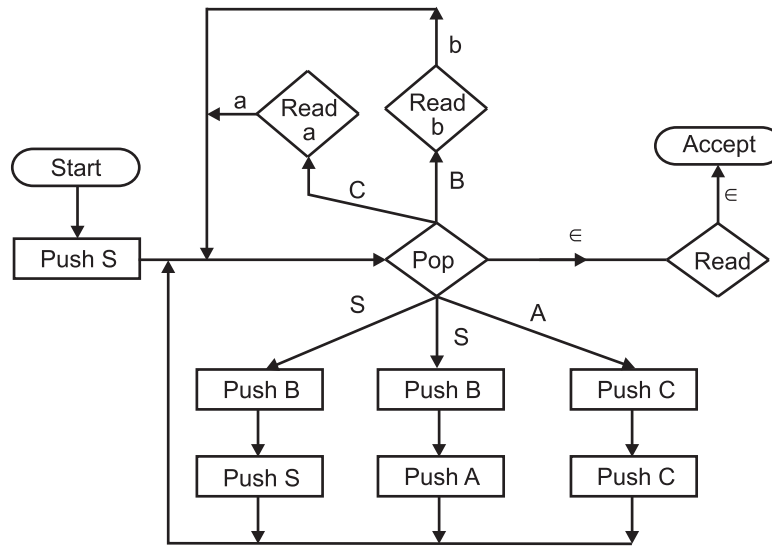


Fig. 4.20

Example 5: Construct a PDA to check whether the given expression is a valid postfix expression. Assume all operators to be binary.

Solution: If all operators are assumed to be binary, the postfix expression is \rightarrow operand 1, operand 2, operator e.g. $ab * c +$, $ab +$.

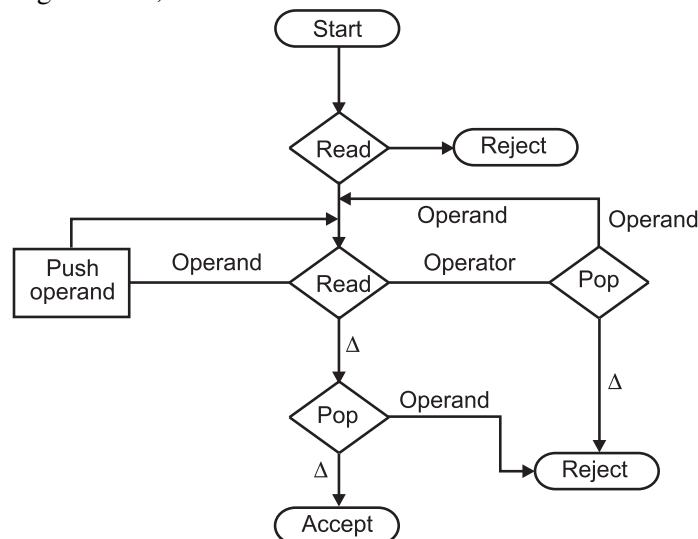


Fig. 4.21

Step 1 : Read the first symbol (operand).

Step 2 : Read next symbol and push it (operand).

Step 3 : Read next symbol, if it is operator then pop the symbol from the stack. If the symbol popped is operand then go back to step 2, else if symbol popped is Δ then go to Reject. If the next input symbol read is Δ , pop the symbol from the stack. If it is Δ , go to Accept, otherwise go to reject. The PDA is shown in the Fig. 4.21.

Example 6: Construct a PDA for a language in which all the strings have at least one occurrence of aa.

Solution:

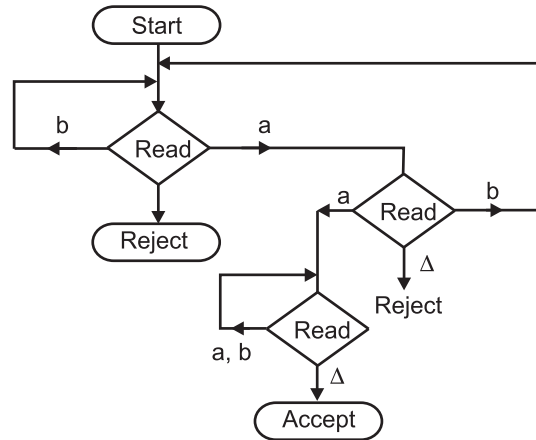


Fig. 4.22

Here we have a language. Start with a or b, when we read b, the next symbol is either a or b. But when we read a, then check for the next symbol. If it is 'b', then read again the next symbol. If it is 'a' (here we get two consecutive a's) then go for accept state, since the language is {aa, baa, baab, baaba, ...}

PDA is $\delta(q_0, a, z_0) = (q_1, Az_0)$

$\delta(q_1, a, A) = (q_2, \epsilon)$

$\delta(q_1, b, A) = (q_1, BA)$

$\delta(q_1, a, B) = (q_1, AB)$

$\delta(q_1, b, B) = (q_1, BB)$

Note: q_2 is the final state language accepted by final state.

EXAMPLES

Example 1: Construct PDA for $L = \{a^x b^y z^z \mid x = 2y + z, y, z \geq 1\}$

Solution: $\delta(q_0, a, z_0) = (q_0, az_0)$

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

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

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, c, a) = (q_2, \epsilon)$

$\delta(q_0, \epsilon, z_0) = (q_2, \epsilon, \epsilon) = \text{accept.}$

Example 2: Construct PDA for $L = \{a^n b^{2n} c^k \mid n \geq 1, k \geq 0\}$.

Solution:

$$\begin{aligned}\delta(q_0, a, z_0) &= (q_0, az_0) \\ \delta(q_0, a, a) &= (q_0, aa) \\ \delta(q_0, b, a) &= (q_1, a) \\ \delta(q_1, b, a) &= (q_2, \epsilon) \\ \delta(q_2, c, \epsilon) &= (q_3, \epsilon)\end{aligned}$$

↑
 q_3 is accept state.

This PDA is the language accepted by final state.

We check only a and double b's when c is read, language is accepted even though the stack is not empty.

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

- Which is the machine format of the context-free language?
 - Pushdown Automata (PDA)
 - Finite Automaton (FA)
 - Push Automata (PA)
 - None of the mentioned
- If a PDA being in a state with a single input and a single stack symbol gives a single move, then the PDA is called as deterministic pushdown automata?
 - Deterministic Finite Automaton (DFA)
 - Deterministic Pushdown Automata (DPDA)
 - Non-Deterministic Pushdown Automata (NPDA)
 - None of the mentioned
- The difference between FA and PDA is in,
 - Finite control
 - Reading head
 - Stack
 - Input tape
- Which of the following is not possible algorithmically?
 - NPDA to DPDA
 - NFA to DFA
 - RE to CFG
 - CFG to PDA
- A PDA can be formulated using how many tuples,
 - 7
 - 5
 - 6
 - 9
- The Instantaneous Description (ID) of PDA is,
 - the identity of that PDA
 - describes the identity of that PDA always nonzero
 - describe the configuration of PDA at a given instant (answer) always zero
 - All of the mentioned
- Which is the machine accepting a context-free language?
 - PDA
 - FA
 - DFA
 - NFA

8. In PDA which consists of a finite list of symbols.
 - (a) input tape
 - (b) stack
 - (c) finite control
 - (d) head
9. Which of the following languages is accepted by a NPDA but not by a deterministic PDA?
 - (a) Always a context-free language
 - (b) Always regular
 - (c) Never regular
 - (d) None of the mentioned
10. To declare a string accepted by a pushdown automata languages uses,
 - (a) accepted by the empty stack
 - (b) accepted by the final state
 - (c) Both (a) and (b)
 - (d) None of the mentioned

Answers

1. (a)	2. (b)	3. (c)	4. (a)	5. (a)	6. (c)	7. (a)	8. (b)	9. (c)	10. (c)
--------	--------	--------	--------	--------	--------	--------	--------	--------	---------

Q.II Fill in the Blanks:

1. A _____ consists of a 7-tuple M is a system $(Q, \cdot, \cdot, \cdot, q_0, Z_0, f)$.
2. A DFA can remember a finite amount of information, but a PDA can remember an _____ amount of information.
3. PDA is the machine format of _____.
4. The _____ control can be considered as a control unit of a PDA.
5. If a PDA being in a state with a single input and a single stack symbol gives more than one move for any of its transitional functions, then the PDA is called _____ pushdown automata.
6. Instantaneous Description (ID) describes the configuration of the PDA at a given _____.
7. A _____ is a temporary storage of stack symbols.
8. A context-free language is called non-deterministic context-free language if it is accepted by a _____.
9. PDA has a stack called the _____ pushdown store (abbreviated PDS) which is a read-write pushdown store as we add elements to PDS or remove elements from PDS.
10. A context-free language is called deterministic context-free language if it is accepted by a _____.
11. A pushdown automaton reads a given input string from _____ left to right and in each step, it chooses a transition by indexing a table by input symbol, current state, and the symbol at the top of the stack.

Answers

1. Pushdown Automata (PDA)	2. infinite	3. context-free language	4. finite
5. nondeterministic	6. instance	7. stack	8. NPDA
9. pushdown store (PDS)	10. DPDA	11. left to right	

Q.III State True or False:

1. A pushdown automaton (PDA) is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.
2. Every PDA accepts only CFL's.
3. PDA is the same as finite automata with the attachment of an auxiliary amount of storage as stack.
4. PDA accepts only non-regular sets.
5. ID remembers the information of the state and the stack content at a given instance of time.
6. A PDA is said to be a DPDA if all derivations in the design give only a single move.
7. In a PDA adding a symbol at the top of the stack is referred to as pushing a symbol onto the stack, and removing a symbol is referred to as popping an item from the stack.
8. PDA is the machine accepting a context-free language.
9. PDA which accepts the language that is produced by the context-free grammar G.
10. A pushdown automaton is called NDPA if one of the derivations generates more than one move.
11. A pushdown automaton can also manipulate the stack, as part of performing a transition. The manipulation can be to push a particular symbol to the top of the stack, or to pop off the top of the stack.
12. The PDA can be directly generated from the CFG.
13. A DPDA which accepts by empty stack cannot accept all Regular Languages.
14. Instantaneous Description (ID) is an informal notation of how a PDA computes an input string and makes a decision that string is accepted or rejected.
15. A language can be accepted by Pushdown automata using two approaches namely Acceptance by Empty Stack (on reading the input string from the initial configuration for some PDA, the stack of PDA gets empty) and Acceptance by Final State (to accept its input by the final state if it enters any final state in zero or more moves after reading the entire input).
16. For directly constructing a PDA from a given grammar, we first need to convert the grammar into Greibach Normal Form (GNF).

Answers

1. (T)	2. (T)	3. (T)	4. (T)	5. (F)	6. (T)	7. (T)	8. (T)	9. (T)	10. (T)
11. (T)	12. (T)	13. (T)	14. (T)	15. (T)	16. (T)				

Q.IV Answer the following Questions:**(A) Short Answer Questions:**

1. Define is PDA.
2. Construct an equivalent PDA for $S \rightarrow aB/bA$ context-free grammars.
3. Define DPDA.
4. What is NPDA?
5. Compare FA and PDA (any two points).
6. Which component used in PDA for temporary storage of symbols.

(B) Long Answer Questions:

1. What is PDA? Explain with the help of diagram.
2. How to construct a PDA using empty stack and final state methods?
3. Give an equivalent PDA for the following CFG:
 - (i) $S \rightarrow AB$
 $A \rightarrow BBla$
 $B \rightarrow ABla|b$
 - (ii) $S \rightarrow aAbblAB$
 $A \rightarrow bbB|blab$
 $B \rightarrow BbAlblaB$
 - (iii) $S \rightarrow AB$
 $A \rightarrow aAblab$
 $B \rightarrow cBdlcd$
 - (iv) $S \rightarrow 01AA|A10$
 $A \rightarrow 0B|0$
 $B \rightarrow |B00|0B|1$
 - (v) $S \rightarrow aBlbA$
 $A \rightarrow alaS|bAA$
 $B \rightarrow blbS|aBB$
4. Construct a PDA for the following languages:
 - (i) $\{a^n b^{2n} \mid n \geq 1\}$
 - (ii) $\{a^n b^m \mid n \geq 1\} \cup \{a^m b^{2n} \mid m \geq 1\}$
 - (iii) Set of all strings over $\{0, 1\}$ starting with "00" and containing even number of 0's in it.
 - (iv) $L = L_1 \cup L_2$
 $L_1 = \text{Set of all strings over } \{a, b\} \text{ having equal number of a's and b's}$
 $L_2 = \{b^n c^m \mid n > m, m \geq 0\}$
 - (v) $L = \{0^n 1^{2n+1} \mid n \geq 1\}$
 - (vi) $L = \{a^m b^n c^{n+2} d^m \mid n, m \geq 1\}$
 - (vii) All strings starting with 0 and having even number of 1's in it over $\{0, 1\}^*$.
5. Show that the DPDA and NPDA are not equivalent with an appropriate example.
6. Prove with an example that if the language is accepted by a CFG, then the same language is accepted by a PDA.
7. Write formal definition of DPDA and NDPA.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Different between PDA and FA.

[1 M]

Ans. Refer to Page 4.5.

2. Define PDA and construct PDA for $L = \{a^n b^m a^n \mid m, n \geq 1\}$. [5 M]

Ans. Refer to Sections 4.1 and 4.2.

3. Convert the following CFG into PDA: [4 M]

$$S \rightarrow 1S \mid 1S0S \mid 1$$

Also show that string "11101" is accepted by the PDA.

Ans. Refer to Section 4.4.

October 2016

1. Why is PDA more powerful than FA? [1 M]

Ans. Refer to Page 4.5.

2. Construct a PDA for $L = \{a^m b^n c^{n+1} d^m \mid n \geq 1, m \geq 1\}$. [1 M]

Ans. $\delta(q_0, a, 20) = (q_0, A 20)$

$$\delta(q_0, a, A) = (q_0, AA)$$

$$\delta(q_0, b, A) = (q_1, BA)$$

$$\delta(q_1, b, B) = (q_1, BB)$$

$$\delta(q_1, c, B) = (q_2, \epsilon)$$

$$\delta(q_2, C, B) = (q_2, \epsilon)$$

$$\delta(q_2, c, A) = (q_3, A)$$

$$\delta(q_3, d, A) = (q_4, \epsilon)$$

$$\delta(q_4, d, A) = (q_4, \epsilon)$$

$$\delta(q_4, 20) = (q_5, \epsilon)$$

3. Construct PDA equivalent to the given CFG: [5 M]

$$S \rightarrow AA$$

$$A \rightarrow 0A0 \mid A1 \mid 1$$

Ans. The given CFG is not CNF

Set $B \rightarrow 0$ and $C \rightarrow 1$

By replacing B and C in A, we get

$$A \rightarrow OAB \mid AC \mid 1$$

After removing the left recursion in A, we get

$$A \rightarrow OAB \mid 1 \mid OABA' \mid 1A'$$

$$A' \rightarrow C \mid CA'$$

The production of A is in GNF

By replacing the production of C in A', we get

$$A' \rightarrow 1 \mid 1A'$$

Now the production of A' are also in GNF.

By replacing the production of A in S we get

$$S \rightarrow OABA \mid 1A \mid OABA'A \mid 1A'A.$$

The grammar is

$$S \rightarrow OABA \mid 1A \mid OABA'A \mid 1A'A$$

$$A \rightarrow OAB \mid 1 \mid OABA' \mid 1A'$$

$$A' \rightarrow 1 \mid 1A'$$

$$B \rightarrow 0$$

$$C \rightarrow 1$$

April 2017

1. Define ID for PDA.

[1 M]**Ans.** Refer to Page 4.6.

2. Construct a PDA for $L = \{0^m 1^n 2^n 0^m \mid m \geq 1, n \geq 1\}$.

[5 M]**Ans.** Refer to Section 4.1, Examples.**October 2017**

1. Name the type of languages accepted by Pushdown Automata.

[1 M]**Ans.** Refer to Page 4.6.

2. Construct PDA for language: $L = \{a^m b^m \mid m > n \geq 1\}$

[5 M]**Ans.** Refer to Section 4.1, Examples.**April 2018**

1. State two differences between PDA and FA.

[1 M]**Ans.** Refer to Page 4.5.

2. Construct PDA that accept language as $S \rightarrow aS \mid aSbS \mid a$.

[4 M]**Ans.** Refer to Section 4.1, Examples.**October 2018**

1. Write a mapping of δ in PDA.

[5 M]**Ans.** Refer to Section 4.1.

2. Construct PDA for language:

[1 M]

$$L = \{a^n b^{2n+1} \mid n > 1\}.$$

Ans. Refer to Section 4.1, Examples.

3. Differentiate between FA and PDA.

[2 M]**Ans.** Refer to Page 4.5.**April 2019**

1. Construct PDA for $L = \{a^n b^{2n+1} \mid n > 1\}$

[4 M]**Ans.** Refer to Section 4.1, Examples.

2. Construct PDA equivalent to the given CFG:

[1 M]

$$S \rightarrow b A B \mid a B$$

$$A \rightarrow a A B \mid a$$

$$B \rightarrow a B B \mid b$$

Ans. Refer to Section 4.4, Examples.

■■■

Turing Machine

Objectives ...

- To study Basic Concepts in Turing Machine
- To learn Design of Turing Machine
- To understand Linear Bounded Automation (LBA)

5.0 INTRODUCTION

- We have seen various machines such as FA and PDA. To remove the limitations, we required a more powerful machine, called Turing machine.
- Turing machine is a simple mathematical model of a modern digital computer. This machine was introduced by Alan Turing in 1936.
- We can compare different machines which are shown in the Table 5.1.

Table 5.1

Language Defined By	Acceptor or Recognizer	Non-determinism = Determinism ?	Example of Application
1. Regular expression	Finite automata	Yes	Text editors
2. Context-free grammar	Pushdown automata	No	Programming language, statements, compilers
3. Type 0 grammar	Turing machine	Yes	Computers

- Turing machine can be constructed to accept a given language or to carry out some algorithm. Machine may take its own output as input for further operation, hence no distinction between input and output set.
- The machine can choose the current location and also decides whether to move left or right in the memory.
- Today, the turning machine has become the accepted formalization of an effective procedure. Turing machine is equivalent in computing power to the digital computer as we know it today and also to all the most general mathematical notion of computation.

5.1 BASIC CONCEPTS IN TURING MACHINE

[Oct. 16]

- In this section we will study basic concepts in Turing Machine (TM) like definition, TM model and design of TM.

5.1.1 Turing Machine Model

- Fig. 5.1 shows model of Turing machine. The Turing machine consists of
 - The **head** which can read or write a symbol and move left or right or stay in position, corresponding to the square cell read.
 - An **infinite tape**, which is divided into cells, on which the symbols from an alphabet set can be written (refer Fig. 5.1). Tape is infinite to right.
 - The **tape** contains the input alphabets with an infinite number of blanks at the left and the right hand side of the input symbols.

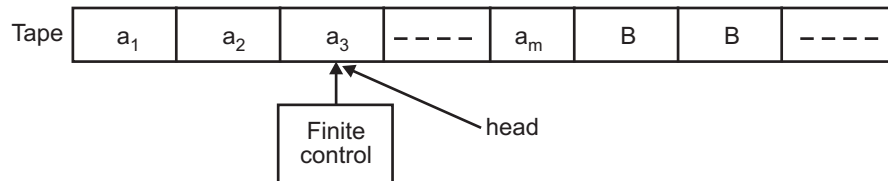


Fig. 5.1: Turing Machine Model

- Finite set of states, in one of which the machine can reside. In one move, TM
 - changes the state.
 - prints a symbol on the tape cell scanned, replacing the previous symbol.
 - moves the head to left (L) or right (R).

Working of Turing Machine:

- A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape.
- After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

5.1.2 Definition of Turing Machine

[April 17. Oct. 18]

- A TM can be formally defined as, M is a 7-tuple

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

where,

Q is a finite set of states.

Γ is a finite set of tape symbols.

$B \in \Gamma$ is a blank symbol.

Σ is a set of input symbols and $\Sigma \subseteq \Gamma - \{B\}$ (i.e. not including B).

δ is the next move function, a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$.

q_0 in Q is the start state.

$F \subseteq Q$ is set of final states.

Instantaneous Description (ID):

- An ID of TM is defined in terms of the entire input string and the current state. We denote ID of the TM, M by $\alpha q \beta$. Here q is the current state of M , is in Q .
- The $\alpha \beta$ is the string in Γ^* i.e. the contents of the tape upto the rightmost non-blank symbol or the symbol to the left of the head, whichever is rightmost.
- Consider the TM shown in the Fig. 5.2. Obtain its ID.

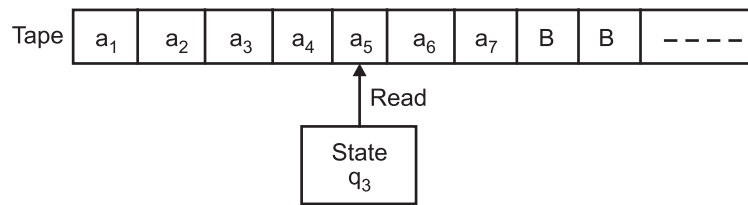


Fig. 5.2

- The present symbol under head is a_5 and the present state is q_3 . So a_5 is written to the right of q_3 . The non-blank symbols to the left of a_5 is a string $a_1 a_2 a_3 a_4$ which is written to the left of q_3 .
- The sequence of non-blank symbols to the right of a_5 is $a_6 a_7$. Thus, ID is $a_1 a_2 a_3 a_4 q_3 a_5 a_6 a_7$ then $\alpha = a_1 a_2 a_3 a_4$ and $\beta = a_5 a_6 a_7$ where head is scanning a_5 .

TM Moves:

- The $\delta(q, x_i)$ induces a change in ID of the TM. Such a change in ID is called a move.
- Let $x_1, x_2, x_3 \dots x_n$ is an input string to be processed and present symbol under tape head is x_i .
 1. Let $\delta(q, x_i) = (p, y, L)$

So the ID before processing the symbol x_i will be

$$x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n.$$

So after processing x_i , the ID will be

$$x_1 x_2 \dots x_{i-2} p x_{i-1} y x_{i+1} \dots x_n$$

This is represented as

$$x_1 x_2 \dots x_{i-1} q x_i \dots x_n \vdash x_1 x_2 \dots x_{i-2} p x_{i-1} y x_{i+1} \dots x_n.$$

2. If $\delta(q, x_i) = (p, y, R)$ then the change in ID is represented by

$$x_1 x_2 \dots x_{i-1} q x_i \dots x_n \vdash x_1 x_2 \dots x_{i-1} y p x_{i+1} \dots x_n.$$

Note: The symbol \vdash^* denotes the reflexive transitive closure of relation \vdash .

Language Acceptance by TM:

- The language accepted by M is,

$$L(M) = \{w \mid w \in \Sigma^* \text{ and } q_0 w \vdash \alpha p \beta, \text{ for some } p \text{ in } F \text{ and } \alpha \beta \text{ in } \Gamma^*\}.$$
- We assume that TM halts whenever the input is accepted. So TM has no next move when it enters in the final state. But if string is not accepted, it may go in infinite loop.

5.1.3 Design of TM**[Oct. 16, April 17, 18]**

- Turing machine is a simple mathematical model which computes two types of problems.
 - Problems on language recognition (Class of languages).
 - Problems on Arithmetical operation (Class of integer functions).
- Examples on Language Recognition:**
 - These examples check whether the string or word is accepted by language or not.
 - If language is accepted by Finite Automata, then it is also accepted by TM.
 - If language is accepted by pushdown automata, then it is also accepted by TM.
 - All regular sets and context free languages are accepted by TM.
 - Languages which are not regular set as well as not context free are also accepted by TM.
- All the following examples show that TM accept all types of regular or non-regular languages.

Example 1: Construct the TM which accept regular language $a^* b^*$.

Solution: $L = \{\epsilon, a, b, aa, bb, ab, aab, \dots\}$

Logic:

- Read all a's using state q_0 .
- If string is over, then accept the string, otherwise scan all b's using state q_1 . State q_1 will not accept any 'a'.

We design $TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

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

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B\}$ and δ is

Table 5.2

State	a	b	B
q_0	(q_0, a, R)	(q_1, b, R)	(q_2, B, R)
q_1		(q_1, b, R)	(q_2, B, R)
q_2	\rightarrow Final state		

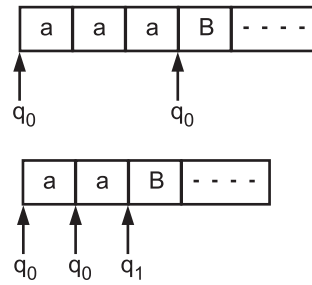


Fig. 5.3

The instantaneous description for “aabb” is

$$\begin{aligned}
 q_0 a a b b B & \vdash a q_0 a b b B \vdash a a q_0 b b B \\
 & \vdash a a b q_1 b B \vdash a a b b q_1 B \\
 & \vdash a a b b b q_1 B \vdash a a b b b q_2 \rightarrow \text{Final state}
 \end{aligned}$$

Example 2: Construct the TM which will accept the language starts with 0 and ends with 1.

Solution: $L = \{01, 001, 011, 01011, \dots\}$

It is regular language since it is represented with regular expression

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

Logic:

- Read first symbol 0 at state q_0 and change the state q_1 when only 0 is read. (1 is not the first symbol).
- At state q_1 read 0 without changing the state move to right.
- At state q_1 , when 1 is read, change the state to q_2 and move to right.
- At state q_2 , when 0 is read, change the state to q_1 and when 1 is read, state will not change, move to right till end of the string occurs.

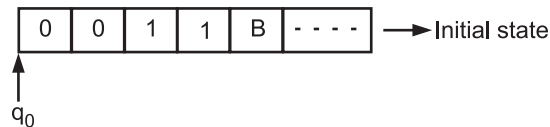


Fig. 5.4

TM is as follows:

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_3\})$$

δ is

Table 5.3

State	0	1	B
q_0	$(q_1, 0, R)$	—	—
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	—
q_2	$(q_1, 0, R)$	$(q_2, 1, R)$	(q_3, B, R)
q_3	Final state		

Instantaneous description for string "0011" is

$$\begin{aligned} q_0 0011 B &\vdash 0q_1 011 B \\ &\vdash 00 q_1 11 B \vdash 001q_2 1B \\ &\vdash 0011q_2 B \vdash 0011q_3 = \text{accept} \end{aligned}$$

Example 3: Design a TM, M to accept the context-free language $L = \{0^n 1^n \mid n \geq 1\}$.

Solution: Repeatedly, M replace the leftmost 0 by X, moves right to the leftmost 1, replacing it by Y, moves left to find the rightmost X, moves one cell right to leftmost 0 and repeats the cycle. When searching for a 1, M finds a blank instead, then M halts without accepting.

$$\begin{aligned} \text{Let,} \quad Q &= \{q_0, q_1, q_2, q_3, q_4\} \text{ and } \Sigma = \{0, 1\} \\ T &= \{0, 1, X, Y, B\} \text{ and } F = \{q_4\} \end{aligned}$$

1. State q_0 is entered initially and used when each replacement of a leftmost 0 by X and change the state to q_1 .
2. State q_1 is used to search right, skipping over 0's and Y's until it finds the leftmost 1. If M finds a 1, it changes it to Y entering state q_2 .
3. State q_2 searches left for X and enters state q_0 upon finding it, moving right, to the leftmost 0, as it changes the state.
4. As M searches right in state q_1 and if a B or X is encountered before a 1, then input is rejected, either there are too many 0's or input $\notin 0^* 1^*$.

From q_0 , scanning Y, state q_3 is entered to scan over Y's and check that no 1's remains. If the Y's are followed by a B, state q_4 is entered and acceptance occurs, otherwise rejected.

Let input string = 0011

$$\delta(q_0, 0) = (q_1, X, R) \rightarrow \text{first move}$$

$$\delta(q_3, B) = (q_4, B, R) \rightarrow \text{last move.}$$

Therefore δ function is as follows shown in the Table 5.4.

Table 5.4

State	0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	final state	—	—	—	—

A computation of TM, M for a string "0011" is as follows:

$$\begin{aligned} q_0 0011 &\vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y_1 \\ &\vdash q_2 X 0 Y_1 \vdash X q_0 0 Y_1 \vdash X X q_1 Y_1 \end{aligned}$$

$$\begin{aligned}
&\vdash \quad XX \ Y \ q_1 \ 1 \vdash XX \ q_2 \ YY \vdash X \ q_2 \ X \ YY \\
&\vdash \quad XX \ q_0 \ YY \vdash XX \ Y \ q_3 \ Y \vdash XXYY \ q_3 \\
&\vdash \quad XX \ YY \ B \ q_4
\end{aligned}$$

As TM enters in q_4 , the string '0011' is accepted.

Example 4: Design a TM to recognize all strings consisting of even number of 1's. Assume that the string is made up of only 1's.

Solution: We will construct TM as follows:

1. Let q_0 be the initial state. The machine M will enter q_1 after reading a '1' and it will replace 1 by B.
2. In state q_1 , on reading a 1, the machine goes back to q_0 after replacing that 1 with B.
3. q_0 will be the final state.

Thus, if number of 1's are even in input string, then after scanning all the symbols in the input, the machine will be in q_0 which is the final state and the string is accepted, otherwise reject.

Hence, TM $M = (\{q_0, q_1\}, \{1\}, \{1, B\}, \delta, q_0, B, \{q_0\})$, where δ function is as shown in the Table 5.5.

Table 5.5: Transition table for example 5.4

State	1	B
q_0	(q_1, B, R)	Accept
q_1	(q_0, B, R)	Reject

The ID for the word "1111" will be

$$\begin{aligned}
q_0 \ 1111 \ B &\vdash \quad B \ q_1 \ 111 \ B \\
&\vdash \quad BB \ q_0 \ 11 \ B \\
&\vdash \quad BBB \ q_1 \ 1 \ B \\
&\vdash \quad BBBB \ q_0 \ B \ \text{accept.}
\end{aligned}$$

The ID for the word "111" will be

$$\begin{aligned}
q_0 \ 111 \ B &\vdash \quad B \ q_1 \ 11 \ B \\
&\vdash \quad BB \ q_0 \ 1 \ B \\
&\vdash \quad BBB \ q_1 \ B.
\end{aligned}$$

Here, q_1 is not a final state, therefore the word "111" is rejected.

Example 5: Construct a TM for language $L = \{a^m b^n \mid n \geq m \text{ and } m \geq 1\}$.

Solution: The steps involved can be written as:

1. TM starts in state q_0 .
2. If the symbol a is read, then state q_0 changes to q_1 and replace symbol by X and move towards right then check for 1st occurrence of b; skipping a and Y.

3. Replace b by Y and change the state to q_2 and move towards left till we get first X then move towards right.
4. If symbol is 'a' then repeat the cycle from step 2, else if symbol is Y then move towards right until current symbol is not equal to b.
5. If current symbol is 'b' then it means that there is atleast one extra b then check whether remaining string consists of only b's followed by a blank symbol. Also if current symbol is 'B' then string is accepted as $m = n$.

We construct TM as follows:

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, X, Y, B\}, \delta, q_0, B, \{q_5\})$, where δ function is as shown in the Table 5.6.

Table 5.6

State	a	b	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	(q_1, a, R)	(q_2, Y, L)		(q_2, Y, R)	
q_2	(q_2, a, L)	(q_2, Y, L)	(q_0, X, R)		
q_3		(q_4, b, R)		(q_3, Y, R)	(q_5, b, R)
q_4		(q_4, b, R)			(q_5, b, R)
q_5	final state				

The TM process for string aabbb is as follows:

$$\begin{aligned}
 q_0 \text{ aabbb} &\vdash X q_0 \text{ abbb} \vdash X a q_1 \text{ bbb} \vdash X q_2 a Y \text{ bb} \\
 &\vdash q_2 X a Y \text{ bb} \vdash X q_0 a Y \text{ bb} \vdash XX q_1 Y \text{ bb} \\
 &\vdash XX Y q_2 \text{ bb} \vdash XX q_2 YY b \vdash X q_2 X YY b \\
 &\vdash XX q_0 YY b \vdash XX Y q_3 Y b \vdash XX YY q_3 b \\
 &\vdash XX YY b q_4 B \vdash XX YY b B q_5
 \end{aligned}$$

As TM enters in q_5 , the string is accepted.

Example 6: Construct a TM for language $L = \{a^n b^n a^n \mid n \geq 1\}$.

[Oct. 17]

Solution: The TM, M works as follows:

1. TM starts in state q_0 .
2. If symbol a is read, then state q_0 changes to q_1 and a changes to X. TM moves towards right to check first occurrence of b, skipping a's and Y's.
3. If symbol b is read, state q_1 changes to q_2 and replace b to Y, and moves towards right to check first occurrence of a, skipping b's and Z's.
4. If symbol a is read, state q_2 changes to q_3 and a is replaced with Z and moves towards left to check leftmost X.

5. If leftmost X is read at state q_3 , state changes to q_0 and X remains as it is and the cycle repeats from step 1 again.
6. At q_0 when symbol Y is read the state q_4 enters and in q_4 when B is read then string is accepted. So we construct TM as follows:

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \{a, b, X, Y, Z, B\}, \delta, q_0, B, \{q_5\})$, where δ function is shown in the Table 5.7.

Table 5.7

State	a	b	X	Y	Z	B
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_3, Z, L)	(q_2, b, 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, R)
q_5	final state					

Computation for string "aabbbaa" is as follows:

q_0 aabbbaa \vdash X q_1 abbaa \vdash X a q_1 bbaa \vdash X a Y q_2 baa \vdash X a Y b q_2 aa
 \vdash X a Y q_3 b Z a \vdash X a q_3 Y b Z a \vdash X q_3 a Y b Z a \vdash q_3 X a Y b Z a
 \vdash X q_0 a Y b Z a \vdash XX q_1 Y b Z a \vdash XXYY q_2 Z a
 \vdash XX YY Z q_2 a \vdash XX YY q_3 ZZ \vdash XX Y q_3 Y ZZ
 \vdash XX q_3 YY ZZ \vdash X q_3 X YY ZZ \vdash XX q_0 YY ZZ
 \vdash XX Y q_4 Y ZZ \vdash XX YY q_4 ZZ \vdash XX YY Z q_4 ZB
 \vdash XXYYZZ q_4 B \vdash XX YY ZZ B q_5

String is accepted.

Example 7: Design a TM to recognize well-formedness of parenthesis.

Solution: We construct TM, M as follows:

$M = (\{q_0, q_1, q_2, q_3\}, \{ (,) \}, \{ (,), X, Y, B \}, \delta, q_0, B, \{q_3\})$, where δ function is shown in the Table 5.8.

Table 5.8

State	()	X	Y	B
q_0	$(q_0, (, R)$	(q_1, X, L)	(q_0, X, R)	(q_0, Y, R)	(q_2, B, L)
q_1	(q_0, Y, R)	–	(q_1, X, L)	(q_1, Y, L)	–
q_2	–	–	(q_2, X, L)	(q_2, Y, L)	(q_3, B, R)
q_3	final state	–	–	–	–

Computation of string $(())$ is as follows:

$$\begin{aligned} & q_0(()) B \vdash (q_0()) B \vdash ((q_0)) B \vdash (q_1(X) B \\ & \vdash (Y q_0 X) B \vdash (Y X q_0) B \vdash (Y q_1 XX B \vdash (q_1 Y XX B \\ & \vdash q_1(Y XX B \vdash Y q_0 Y XX B \vdash YY q_0 XX B \vdash YY X q_0 X B \\ & \vdash YY XX q_2 B \vdash YY XX B q_3 \rightarrow \text{accept.} \end{aligned}$$

The string is accepted.

Example 8: Design a TM to accept $L = \{wcw^R \mid w \text{ is in } (0 + 1)^*\}$.

Solution: If w is 100 then word is 100 c 001.

1. Let q_0 be the start state, checking for c moving to the rightmost direction, skipping 0's and 1's and then changing the state to q_1 .
2. At q_1 , if input 0 is read, state q_2 goes left to find 0, if it gets 0 then q_3 will move right to get 0 or 1. If 1 is found then state changes to q_4 and q_4 moves left to find 1 and the process is repeated till in q_3 machine gets blank.
3. If in q_5 machine gets blank then accept the string, otherwise reject.

We construct the TM as follows:

TM, $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1, c\}, \{0, 1, c, X, B\}, \delta, q_0, B, \{q_6\})$, where δ function is as shown in the Table 5.9.

Table 5.9

State	0	1	c	X	B
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, c, R)		
q_1	(q_2, X, L)	(q_4, X, L)			
q_2	(q_3, X, R)		(q_2, c, L)	(q_2, X, L)	
q_3	(q_2, X, L)	(q_4, X, L)	(q_3, c, R)	(q_3, X, R)	(q_5, B, L)
q_4	—	(q_3, X, R)	(q_4, c, L)	(q_4, X, L)	
q_5	—		(q_5, c, L)	(q_5, X, L)	(q_6, B, R)
q_6	final state				

Computation for string "100c001" is as follows:

$$\begin{aligned} & q_0 100c001 \vdash 100 q_0 c 001 B \vdash 100 c q_1 001 B \vdash 100 q_2 c X 01 B \\ & \vdash 10 q_2 0 c X 01 B \vdash 10 X q_3 c X 01 B \vdash 10 X c X q_3 01 B \\ & \vdash 10 X c q_2 XX 1 B \vdash 1 q_2 0 X c XX 1 B \vdash 1 X q_3 X c XX 1 B \\ & \vdash 1 XX c XX q_3 1 B \vdash 1 XX c X q_4 XX B \vdash q_4 1 XX c XXX B \\ & \vdash X q_3 XX c XXX B \vdash XXX c XXX q_3 B \vdash XXX c XX q_5 X B \\ & \vdash^* q_5 XXX c XXX B \rightarrow \text{accept.} \end{aligned}$$

Example 9: Design a TM which will replace every occurrence of a substring '11' by '10' over $\{0, 1\}$.

Solution: We will construct TM as follows:

1. Let q_0 be the initial state. At q_0 when we read 1, change the state to q_2 without replacing the symbol. At q_0 when we read 0, state will not change.
2. At state q_1 , when we read 1 (substring 11 is found), replace 1 by 0 and again state changes to q_0 . When we read B go to state q_2 which is the final state.

TM is $M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_0, q_1\})$

where, δ is shown in Table 5.10.

Table 5.10

State	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	(q_2, B, R)
q_1	$(q_1, 0, R)$	$(q_0, 0, R)$	(q_2, B, R)
q_2	final state		

The ID for the word "011011" is

$$\begin{aligned}
 q_0 \ 011011B & \vdash 0q_0 \ 11011B \vdash 01q_1 \ 1011B \\
 & \vdash 010q_0 \ 011B \vdash 0100q_0 \ 11B \\
 & \vdash 01001q_1 \ 1B \vdash 010010q_0 \ B \\
 & \vdash 010010 \ q_2 = \text{accept}
 \end{aligned}$$

Example 10: Design a TM to recognize words in the language $\{a^n b^n c^n \mid n \geq 1\}$

Solution: $L = \{abc, aabbcc, \dots\}$

Logic:

1. At state q_0 , if we read a replace it by X and change the state to q_1 , move right.
2. At q_1 , skip all a's and y's, searching for b when the b is read it is replaced by Y and change the state to q_2 .
3. At q_2 , skip all b's and z's, when c is read it is replaced by z, state changes to q_3 and move to left.
4. At q_3 , move left by skipping all a's, b's, y's, z's. When X is read, then state changes to q_0 and move to right. The cycle is repeated.

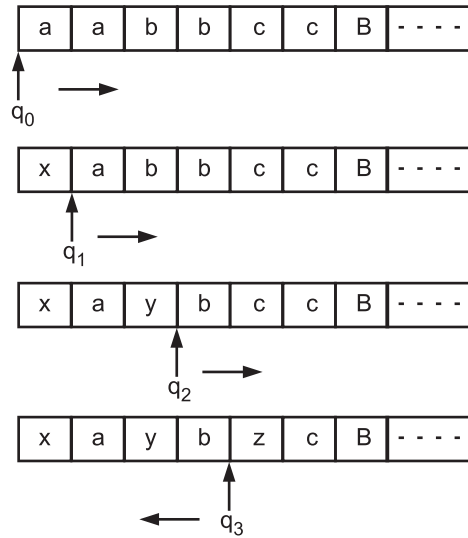


Fig. 5.5: Steps in simulating string

The TM is as follows:

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\} \{a, b, c\} \{a, b, c, X, Y, Z, B\}, \delta, q_0, B, \{q_5\})$$

where δ is:

Table 5.11

State	a	b	c	X	Y	Z	B
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, L)	—	—	(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, R)
q_5	final state						

Here when all a's are over, then machine enters in state q_4 to check that there are no more symbols remaining on the tape. So at q_4 skipping all Y's and Z's, when B is read machine enters into final state q_5 .

The TM process for string "aabbcc" is as follows:

$$\begin{aligned}
 q_0 \text{ aabbccB} & \vdash Xq_1\text{abbccB} \\
 & \vdash Xaq_1\text{bbccB} \vdash XaYq_2\text{bccB} \\
 & \vdash XaYbq_2\text{cc} \vdash XaYbq_3\text{ZcB} \\
 & \vdash XaYq_3\text{bZcB} \vdash Xaq_3\text{YbZcB} \\
 & \vdash Xq_3\text{aYbZcB} \vdash q_3\text{XaYbZcB}
 \end{aligned}$$

$$\begin{aligned}
&\vdash Xq_0 aYbZcB \\
&\vdash XXq_1 YbZcB \vdash XXYq_1 bZcB \\
&\vdash XXYq_2 ZcB \\
&\vdash XXYq_2 cB \vdash XXYq_3 ZB \\
&\vdash XXYq_3 ZZB \\
&\vdash XXYq_3 ZZB \\
&\vdash XXq_3 YYZZB \vdash Xq_3 XYYZZB \\
&\vdash XXq_0 YYZZB \\
&\vdash XXYq_4 YZZB \\
&\vdash XXYq_4 ZZB \\
&\vdash XXYq_4 ZZB \vdash XXYq_5 = \text{accept}
\end{aligned}$$

Example 11: Design a TM for language $\{ww^R \mid w \in (0 + 1)^*\}$

Solution: Logic:

If we read first 0, then read last 0 and If we read first 1 then read last 1.

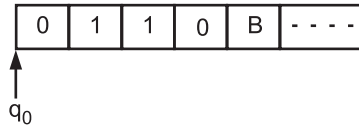


Fig. 5.6

The TM is

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\} \{0, 1\} \{0, 1, X, B\} \{\delta, q_0, B, F\})$ where δ is as:

Table 5.12

State	0	1	X	B
q_0	(q_1, X, R)	(q_4, X, R)	(q_6, X, R)	–
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_2, X, L)	(q_2, B, L)
q_2	(q_3, X, L)	–	–	–
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, X, R)	–
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	(q_5, X, L)	(q_5, B, L)
q_5	–	(q_3, X, L)	–	–
q_6	accept state		(q_6, X, R)	(q_6, B, R)

Consider string “0110”.

$$\begin{aligned}
q_0 0110B &\vdash Xq_1110B \\
&\vdash X1q_110B \vdash X11q_10B \\
&\vdash X110q_1B \vdash X11q_20B \\
&\vdash X1q_31XB \vdash Xq_311XB
\end{aligned}$$

$$\begin{aligned}
& \vdash q_3 X11XB \vdash Xq_0 11XB \\
& \vdash XXq_4 1XB \vdash XX1q_4 XB \\
& \vdash XXq_5 1XB \vdash Xq_3 XXXB \\
& \vdash XXq_0 XXB \vdash XXXq_6 XB \\
& \vdash XXXXq_6 B \vdash XXXXq_6 = \text{accept}
\end{aligned}$$

Example 12: Design a TM which recognize well formedness of parenthesis, over $\{ (, [, \{ , \} ,] ,) \}$.

[April 18]

Solution: $L = \{ \{ \}, [], (), [()], \{ () \}, \dots \}$

Logic:

1. At state q_0 , find the first symbol. The symbol is $\{$, $[$, or $($, skip all opening parenthesis till machine reads first leftmost closing parenthesis.
2. When machine reads $)$ then change it to Y and change the state to q_1 and move to left.
3. If machine reads $]$ then change to Y and state is q_2 , move to left.
4. If machine reads $\}$ then change to Y and state is q_3 and move to left.
5. At q_1 , check for $($ bracket.

If machine reads $($ parenthesis then change to X and state changes to q_0 .

6. At q_2 when $[$ is read then make X and change state to q_0 .
7. At q_3 when $\{$ is read then make X and change state to q_0 .
8. The TM is

$M = (\{q_0, q_1, q_2, q_3, q_4\} \{ (, [, \{ , \} ,] ,) \}, \{ (, [, \{ , \} ,] ,) , X, Y, B \}, \delta, q_0, B, \{q_4\})$

where δ is

Table 5.13

State	([{	}])	X	Y	B
q_0	$(q_0, (, R)$	$(q_0, [, R)$	$(q_0, \{, R)$	(q_3, Y, L)	(q_2, Y, L)	(q_1, Y, L)	(q_0, X, R)	(q_0, Y, R)	(q_4, B, L)
q_1	(q_0, X, R)						(q_1, X, L)	(q_1, Y, L)	
q_2		(q_0, X, R)					(q_2, X, L)	(q_2, Y, L)	
q_3			(q_0, X, R)				(q_3, X, L)	(q_3, Y, L)	
q_4	Final	state							

Consider string “ $\{ \{ () \} \}$ ”

$$\begin{aligned}
q_0 \{ \{ () \} \} B & \vdash [q_0 \{ () \} B \vdash [\{q_0 () \} B \\
& \vdash [\{ (q_0) \} B \vdash [\{q_1 (Y) \} B \\
& \vdash [\{Xq_0 Y \} B \vdash [\{XYq_0 \} B \\
& \vdash [\{XYq_3 Y \} B \vdash [\{Xq_3 YY \} B
\end{aligned}$$

$\vdash [\{q_3 \text{ XYY} \} B \vdash [q_3 \{ \text{XYY} \} B$
 $\vdash [XX_{q_0} \text{ YY} \} B \vdash [XXYY_{q_0} \} B$
 $\vdash [XXY_{q_2} \text{ YYB}$
 $\vdash [XX_{q_2} \text{ YYYYB} \vdash [X_{q_2} \text{ YYYYB}$
 $\vdash q_2 [XXYYYYB \vdash X_{q_0} \text{ XXXYYY} \vdash XXXYYYY_{q_0} B$
 $\vdash XXXYYYY_{q_4} = \text{accept}$

Example 13: Design TM for language $L = \{a^m b^n c^m \mid m, n \geq 0\}$

Solution: $L = \{\epsilon, b, ac, abc, abbc, aabbcc, \dots\}$

Logic:

- Let start state is q_0 . At q_0 , read leftmost a and make X move towards right. Skipping all b's, search for last 'c'.
- When at state q_1 'c' is read make it Y and move to left.
- Repeat this process till all a's and c's are over. We need to check only equal number of a's and c's. The string consists of any number of b's in between. The TM is as follows:

Table 5.14

State	a	b	c	X	Y	B
q_0	(q_1, X, R)	(q_6, b, R)				
q_1	(q_1, a, R)	(q_2, b, R)	(q_3, c, R)			
q_2		(q_2, b, R)	(q_3, c, R)			
q_3			(q_3, c, R)		(q_4, Y, L)	(q_4, B, L)
q_4			(q_5, Y, L)			
q_5	(q_5, a, L)	(q_5, b, L)	(q_5, c, L)	(q_0, X, R)		
q_6		(q_6, b, R)			(q_6, Y, R)	accept

Consider the string "aabcc"

$q_0 \text{ aabccB} \vdash X_{q_1} \text{ abccB} \vdash X_{q_1} \text{ bccB}$
 $\vdash X_{abq_2} \text{ ccB} \vdash X_{abcq_3} \text{ cB}$
 $\vdash X_{abccq_3} \text{ B} \vdash X_{abcq_4} \text{ cB}$
 $\vdash X_{abq_5} \text{ cYB} \vdash X_{q_5} \text{ abcYB}$
 $\vdash q_5 \text{ XabcYB} \vdash X_{q_0} \text{ abcYB}$
 $\vdash XX_{q_1} \text{ bcYB} \vdash XX_{bq_2} \text{ YB} \vdash XX_{bcq_3} \text{ YB} \vdash XX_{bq_4} \text{ cYB}$
 $\vdash XX_{q_5} \text{ bYYB} \vdash X_{q_5} \text{ XbYYB} \vdash XX_{q_0} \text{ bYYB}$
 $\vdash XX_{bq_6} \text{ YYB} \vdash XXYY_{q_6} \text{ B} \vdash XXYY_{q_6} = \text{accept}$

5.2 TYPES OF TM

[April 16]

- The different types of Turing machine are Multi-track TM, Two-way TM, Multi-tape TM, Non-deterministic TM.

5.2.1 Multi-track TM

- So far, we have discussed the single tap TM. Assume, single tape TM is divided into several tracks. Tape alphabet is consists of k-tuples of tape symbols.
- Single tape and multi-tape TM differs only with the set of tape symbols. Consider the tape symbols of multi-track TM is, Γ^* .
- First, a TM program for the subroutine is written, which will have an initial state and return state. After reaching the return state, there is a temporary halt.
- A new state is introduced by using a subroutine. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine and to return to the main program of TM, when return state is reached.
- The following examples illustrate the concept of multiple track TM using computation of unary numbers.

Use of TM for Computation of Unary Numbers:

- A language that is accepted by a TM is said to be recursively enumerable. In addition to being a language acceptor, the TM may be viewed as a computer of functions from integer to integer.
- The integers are represented in **unary**, the integer $i \geq 0$ is represented by a string 0^i .
- If the function has k argument i_1, i_2, \dots, i_k , then these integers are initially placed on the tape separated by 1's, as $0^{i_1} 1 0^{i_2} 1 \dots 1 0^{i_k}$. If the TM halts with a tape consisting of 0^m for some M, then we say that $f(i_1, i_2 \dots i_k) = m$, where f is the function of k arguments computed by this TM.

Example 1: Design a TM to subtract two unary numbers.

$$f(m, n) = m - n \text{ (proper subtraction)}$$

$$\text{i.e. } f(m - n) = \begin{cases} m - n & \text{if } m \geq n \\ 0 & \text{if } m < n \end{cases}$$

Solution: Unary system represents integers as shown in the Fig. 5.7.

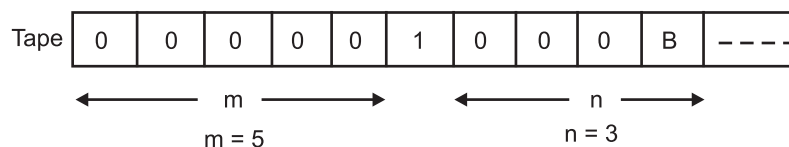


Fig. 5.7: Unary system represents integers

The TM, $M = (\{q_0, q_1, q_2 \dots q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_6\})$ defined below, started with $0^m 1 0^n$.

M repeatedly replaces its leading 0 by blank, then searches right for 1 followed by 0 and changes the 0 to 1. Next M moves left until it encounters a blank and then repeats the cycle. The repetitions end if

1. Rightmost 0's changes to 1 and paired with leftmost 0's, changes to B and leaving $m - n$ 0's on tape.
2. If $n \geq m$, then $m - n = 0$, replace all remaining 1's and 0's by B.

The function δ is as described below in the Table 5.15.

Table 5.15

State	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	final state		

Here, if in state q_0 a 1 is encountered instead of 0. In this, M enters in q_5 to erase the rest of the tape, then enters in q_6 and M halts. State q_4 is used for changing all 1's and B's until encountering a B. B changes back to 0, state q_6 is entered and M halts.

Computation for input 0010 ($m - n = 2 - 1 = 1$) is shown below.

$q_0 0010 \vdash B q_1 010 \vdash B 0 q_1 10 \vdash B 01 q_2 0 \vdash B 0 q_3 11 \vdash B q_3 011 \vdash q_3 B 011$
 $\vdash B q_0 011 \vdash BB q_1 11 \vdash BB 1 q_2 1 \vdash BB 11 q_2 \vdash BB 1 q_4 1 \vdash BB q_4 1 \vdash B q_4$
 $\vdash B 0 q_6 \rightarrow \text{accept.}$

If the input is 0100 then TM moves is shown below.

$q_0 0100 \vdash B q_1 100 \vdash B 1 q_2 00 \vdash B q_3 110$
 $\vdash q_3 B 110 \vdash B q_0 110 \vdash BB q_5 10 \vdash BBB q_5 0$
 $\vdash BBBB q_5 \vdash BBBBB q_6$

Here tape is blank i.e. output = 0.

Example 2: Design a TM to multiply two unary numbers.

Solution:

B	1	1	1	*	1	1	=	B
---	---	---	---	---	---	---	---	---

Let this is the initial configuration. The machine is as follows:

Table 5.16

State	1	*	=	X	Y	B
q_0	(q_1, X, R)	$(q_6, *, R)$	—	—	—	—
q_1	$(q_1, 1, R)$	$(q_2, *, R)$	—	—	—	—
q_2	(q_3, Y, R)	—	$(q_5, =, L)$	—	—	—
q_3	$(q_3, 1, R)$	—	$(q_3, =, R)$	—	—	$(q_4, 1, L)$
q_4	$(q_4, 1, L)$	—	$(q_4, =, L)$	—	(q_2, Y, R)	—
q_5	$(q_5, 1, L)$	$(q_3, *, L)$	—	(q_0, X, L)	$(q_5, 1, L)$	—
q_6	$(q_6, 1, R)$	—	$(q_7, =, R)$	—	—	—
q_7	final state					

Let q_0 is the initial state. At q_1 , second number has been read and states q_2, q_3, q_4 copy multiplier at the end. Once copying is over, state q_5 again retains the multiplier by replacing Y by 1 and the process continues till in q_0 machine gets * i.e. machine goes upto = and stays in the final state q_7 .

For input $11 * 11 =$ the computation is as follows:

$B q_0 11 * 11 = B \vdash B X q_1 1 ** 11 = B \vdash B X 1 q_1 * 11 = B$
 $\vdash B X 1 * q_2 11 = B \vdash B X 1 * Y q_3 1 = B \vdash^* B X 1 * Y 1 = q_3 B$
 $\vdash B X 1 * Y_1 q_4 = 1 B \vdash^* B X 1 * q_4 Y 1 = 1 B \vdash B X 1 * Y q_2 1 = 1 B$
 $\vdash B X 1 * Y Y q_3 = 1 B \vdash^* B X 1 * Y Y = 1 q_3 B \vdash B X 1 * Y Y = q_4 11 B$
 $\vdash B X 1 * Y q_4 Y = 11 B \vdash B X 1 * Y Y q_2 = 11 B$
 $\vdash B X 1 * Y q_5 Y = 11 B \vdash B q_5 X 1 * 11 = 11 B$
 $\vdash B X q_0 1 * 11 = 11 B \vdash B X X q_1 * 11 = 11 B$
 $\vdash B X X * q_2 11 = 11 B \vdash B X X * Y q_4 = 1111 B$
 $\vdash B X X * Y q_5 Y = 1111 B \vdash B X q_5 X * 11 = 1111 B$
 $\vdash B X X q_0 * 11 = 1111 B \vdash^* B X X * 11 q_7 = 1111 B$

5.2.2 Multi-tape TM

[Oct. 16]

- A multi-tape TM has finite set of states Q , initial state q_0 , a set of P of tape symbols, set of final states F , $F \subseteq Q$ and blank symbol $b \notin \Sigma$.
- Consider K -tapes, each divided into cells. The Fig. 5.8 represent multi-tape TM.

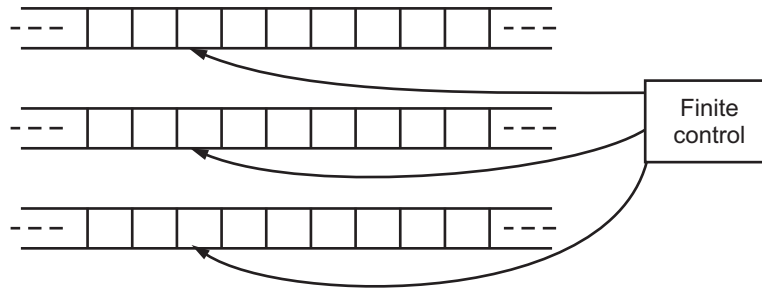


Fig. 5.8: Multi-tape TM

- In a move:
 - M enters a new state.
 - On each tape, a new symbol is written in the cell under the head.
 - Each tape head moves to the left or right or remains stationary. The head move perform independently. Some move to the left, some to the right and the remaining heads do not move.

5.2.3 Two-way TM

- Two-way infinite tape Turing machine having infinite tape is unbounded in both directions left and right.
- Two-way infinite tape Turing machine can be simulated by one-way infinite Turing machine.
- TM can move indefinitely in either direction.
- Example:

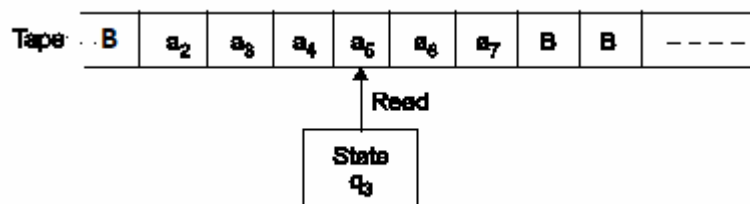


Fig. 5.9

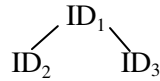
5.2.4 Non-Deterministic TM

[Oct. 17]

- A Non-deterministic Turing Machine (NTM) differs from the deterministic TM by transition function δ such that each state q and tape symbol X , $\delta(q, X)$ is the set of triples:

$$\{(q_1, X_1, D_1), (q_2, X_2, D_2), \dots, (q_n, X_n, D_n)\}$$
 where, n is any finite integer.
- All the examples which we have discussed, the transition function δ has a unique ID on each input symbol. So all are deterministic TM.

- Non-deterministic TM represents ID with tree structure.



- A string ω is said to be accepted by NTM if there exists a sequence of moves starting from the initial ID to an accepting condition.
 - If M_N is a non-deterministic TM, then there is a deterministic TM (DTM), M_D such that

$$L(M_N) = L(M_D)$$
 - So language accepted by NTM and DTM are same.

Examples of NTM:

- The language of all strings of 0's and 1's that have some string of length 100 that repeats, not necessarily consecutively. Formally this language is the set of strings of 0's and 1's of the form $\omega y x z$, where $|x| = 100$, ω, y, z are of any length.

5.3 LINEAR BOUNDED AUTOMATION (LBA) [April 16, 19, Oct. 16]

- The machine to accept context-sensitive language is Linear Bounded Automation (LBA).
- An LBA is a special type of Turing machine with restricted tape space. The name 'linear bounded' suggests that the machine is linearly bounded.

Difference between TM and LBA:

[Oct. 16, 18]

1. In TM the storage is not restricted on tape. Tape head can move to both directions at infinite storage. In LBA, the input string tape space is the only tape space allowed to use. So storage is restricted in size.
 2. In LBA, linear function is used to restrict (to bound) the length of the tape.
 3. In LBA, the computation is done between end markers. In TM, no endmarkers are present, end of string is considered when blank (B) occurs.
- LBA is a non-deterministic TM which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string.

Definition: LBA, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \varphi, \$, F)$

where

Q = Finite set of states

Σ = Input alphabet $U \{\varphi, \$\}$

Γ = Tape alphabet

q_0, F, b are same as TM basic model.

φ and $\$$ are two special symbols $\varphi, \$ \in \Sigma$.

- Both symbols are end markers. φ is called the left-end marker which is entered in the leftmost cell of the input tape and prevents the R/W head from getting off the left-end of the tape.

- The \$ is called the right-end marker which is entered in the rightmost cell of the input tape and prevents the R/W head from getting off the right end of the tape. Both the end markers should not appear on any other cell within the input tape, and the R/W head should not print any other symbol over both the end markers.

5.3.1 Basic model of LBA

- The Fig. 5.10 shows the model of Linear Bounded Automata.

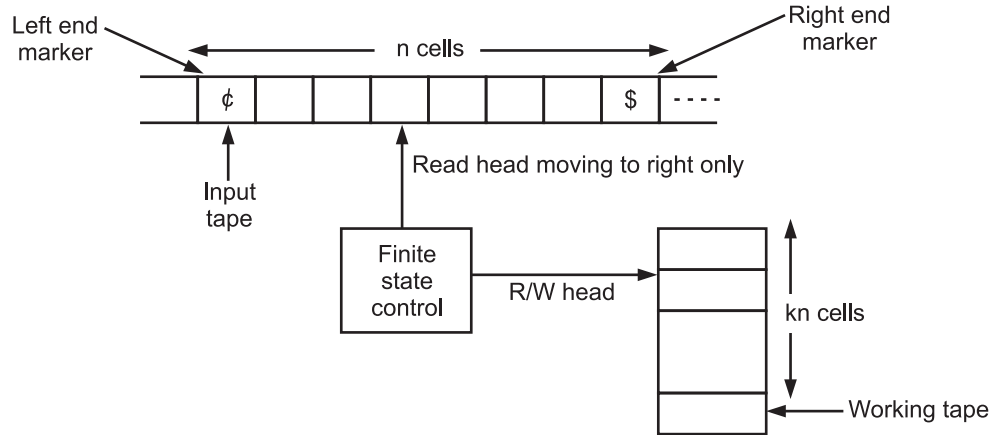


Fig. 5.10: LBA Model

- There are two tapes: input tape and working tape. On the input tape, the head never prints and never moves to the left. On the working tape the head can modify the contents in any way, without any restriction.
- The mapping function δ of LBA is denoted by $\rightarrow (q, w, k)$ where $q \in Q$, $w \in \Gamma$, k is some integer between 1 and n .
 - If R/W head moves to left, k changes to $k - 1$.
 - If R/W head moves to right, k changes to $k + 1$.
- The language accepted by LBA is defined as the set,

$$\{w \in (\Sigma - \{\phi, \$\})^* \mid (q_0, \phi w \$, 1) \vdash^* (q, \alpha, i) \text{ for some } q \in F \text{ and for some integer } i \text{ between } 1 \text{ to } n\}$$

5.3.2 Context-Sensitive Grammar (CSG)

- The Context Sensitive Grammar (CSG) is used to represent Context Sensitive Language (CSL).
- The CSG may have more than one symbol on the left hand side of their production rules. The language that can be defined by context-sensitive grammar is called CSL.
- The set of strings accepted by non-deterministic LBA is the set of strings generated by the context-sensitive grammars (CSG), excluding the null strings.
- If L is a context-sensitive language, then L is accepted by a LBA. The converse is also true.

- Examples of language accepted by LBA:

1. $L = \{a^n b^n c^n \mid n \geq 1\}$

2. $L = \{a^{n!} \mid n \geq 1\}$

EXAMPLES

Example 1: Design TM to check palindrome over $\{a, b\}$.

Solution: We have to check odd and even palindrome.

Table 5.17

State	a	b	X	B
q_0	(q_1, X, R)	(q_4, X, R)	(q_6, X, R)	–
q_1	(q_1, a, R)	(q_1, b, R)	(q_2, X, L)	(q_2, B, L)
q_2	(q_3, X, L)	–	(q_6, X, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, X, R)	–
q_4	(q_4, a, R)	(q_4, b, R)	(q_5, X, L)	(q_5, B, L)
q_5	–	(q_3, X, L)	(q_6, X, L)	–
q_6	Final state			(q_6, X, L)

Consider the string “aba”

$$\begin{aligned}
 q_0 \text{ aba B} &\rightarrow Xq_1 \text{ baB} \vdash Xbq_1 \text{ aB} \\
 &\vdash Xbaq_1 \text{ B} \vdash Xbq_2 \text{ aB} \\
 &\vdash Xq_3 \text{ bXB} \vdash q_3 \text{ XbXB} \\
 &\vdash Xq_0 \text{ bXB} \vdash XXq_4 \text{ XB} \\
 &\vdash Xq_5 \text{ XXB} \vdash q_6 \text{ XXXB} \rightarrow \text{accept}
 \end{aligned}$$

Example 2: Construct TM which will add three unary numbers.

Solution:

B	1	1	1	B	1	1	B	1	1	1	B
---	---	---	---	---	---	---	---	---	---	---	---

Logic:

1. Skip all 1's till B while moving to right.
2. Change B to 1 and move to right.
3. Skip all 1's till we get 2nd B.
4. Change B to 1 and move right to 3rd blank.
5. When 3rd B occurs move to left.
6. Two extra 1's are changed to blanks.

The TM is as follows:

Table 5.18

State	1	B
q ₀	(q ₀ , 1, R)	(q ₁ , 1, R)
q ₁	(q ₁ , 1, R)	(q ₂ , 1, R)
q ₂	(q ₂ , 1, R)	(q ₃ , B, L)
q ₃	(q ₄ , B, L)	–
q ₄	(q ₅ , B, L)	–
q ₅	← accept	

Example 3: Construct TM which will subtract two unary numbers.

Solution:

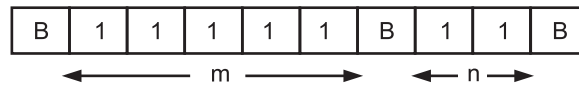


Fig. 5.11

$$f(m, n) = m - n \quad \forall m \geq n \text{ (proper subtraction)}$$

Logic:

1. Skip all 1's till B at state q₀.
2. When B occurs change the state and search for last B. Change state to q₂.
3. Change 1 to B and move to left at q₂.
4. Repeat process till all 1's are over.

Table 5.19

State	1	B
q ₀	(q ₀ , 1, R)	(q ₁ , B, R)
q ₁	(q ₁ , 1, R)	(q ₂ , B, L)
q ₂	(q ₃ , B, L)	(q ₇ , B, L)
q ₃	(q ₃ , 1, L)	(q ₄ , B, L)
q ₄	(q ₄ , 1, L)	(q ₅ , B, R)
q ₅	(q ₀ , B, R)	(q ₆ , 1, L)
q ₆	–	(q ₇ , B, L)
q ₇	← accept	

Output is,

B	B	B	1	1	1	B	B	B	B	---
---	---	---	---	---	---	---	---	---	---	-----

Example 4: Construct TM accepting language,

$$L = \{a^n, b^m, c^{m+n} \mid m, n \geq 0\}$$

Solution:

Logic:

1. Scan leftmost a or b and make X.
2. Scan rightmost c and make Y.
3. Find rightmost X.
4. Repeat the process.
5. q_3 is reject state and q_4 is used for checking extra a's or b's or c's.

Table 5.20

State	a	b	c	X	Y	B
q_0	(q_1, X, R)	(q_1, X, R)	–	–	(q_4, b, R)	–
q_1	(q_1, a, R)	(q_1, b, R)	(q_1, c, 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_3, c, L)	(q_0, X, R)	–	–
q_4	–	–	–	–	(q_4, Y, R)	(q_5, B, L)
q_5	\leftarrow accept					

Example 5: Construct TM for language

$$L = \{a^m b^n \mid n \geq m \text{ and } m \geq 1\}$$

[April 16]

Solution:

Table 5.21

State	a	b	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, L)	–
q_1	(q_1, a, R)	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	(q_2, a, L)	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	(q_3, b, R)	–	(q_3, Y, R)	(q_4, B, R)
q_4	\leftarrow accept				

Example 6: Design a TM which computes a language $L = \{0^{2n} 1^n \mid n \geq 1\}$

Solution:

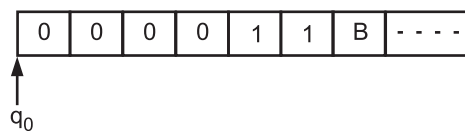


Fig. 5.12

Logic:

- At state q_0 , when 1st 0 is read, change the state q_1 and 0 to X, move right.
- At q_1 , when 0 is read, change the state to q_2 and 0 to X, move right.
- At q_2 skip all 0's till machine reads 1. State is changed to q_3 and 1 changes to Y.
- At q_3 , move to left, skipping all 0's till machine reads X. Change the state to q_0 .

This process repeats.

Table 5.22

State	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_4, Y, R)	–
q_1	(q_2, X, R)	–	–	–	–
q_2	$(q_2, 0, R)$	(q_3, Y, L)	–	(q_2, Y, R)	–
q_3	$(q_3, 0, L)$	–	(q_0, X, R)	(q_3, Y, L)	–
q_4	–	–	–	(q_4, Y, R)	accept

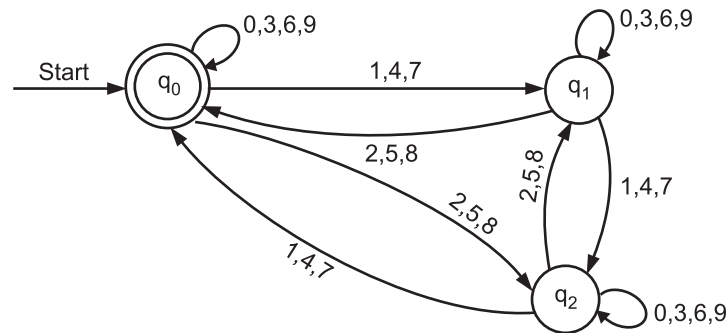
Consider instantaneous description for string “001”.

q_0 001B \vdash X q_1 01B
 \vdash XX q_2 1B
 \vdash X q_3 XYB \vdash XX q_0 YB
 \vdash XXY q_4 B \vdash XXY q_4 \rightarrow accept

Example 7: Design a TM that computes remainder when the number is divisible by 3.

Solution: First we draw finite automata for the machine and then we will design TM.

Finite automata is,

**Fig. 5.13**

Here we get 3 remainders 0, 1 and 2. So machine has 3 states where we get the outputs 0, 1 and 2.

Let we replace (0, 3, 6, 9) by X

(1, 4, 7) by Y

(2, 5, 8) by Z

The TM is (using above FA):

Table 5.23

State	X (0, 3, 6, 9)	Y (1, 4, 7)	Z (2, 5, 8)	B
q ₀	(q ₀ , X, R)	(q ₁ , Y, R)	(q ₂ , Z, R)	(q ₃ , B, R)
q ₁	(q ₁ , X, R)	(q ₂ , Y, R)	(q ₀ , Z, R)	(q ₄ , B, R)
q ₂	(q ₂ , X, R)	(q ₀ , Y, R)	(q ₁ , Z, R)	(q ₅ , B, R)
q ₃	–	–	–	(q ₆ , 0, R)
q ₄	–	–	–	(q ₆ , 1, R)
q ₅	–	–	–	(q ₆ , 1, R)
q ₆	Final state			

Consider the string “241” as input

$$\begin{aligned}
 q_0 \text{ 241B} &\vdash 2q_2 \text{ 41B} \vdash 24q_0 \text{ 1B} \\
 &\vdash 241q_1 \text{ B} \vdash 241Bq_4 \text{ B} \\
 &\vdash \underline{241}B1q_6 = \text{accept} \\
 &\quad \uparrow \quad \uparrow \\
 &\quad \text{number remainder}
 \end{aligned}$$

Example 8: Design a TM to compute 2's complement of a given binary number.

Solution: If input is 1011, its 2's complement is 0101. If input is 1011100, its 2's complement is 0100100. We clearly see that from the rightmost 1 leaving that bit every bit on its left is negated i.e. 0 becomes 1 and 1 becomes 0.

We construct TM as follows:

TM, $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_3\})$, where δ function is as shown in the Table 5.24.

Table 5.24

State	0	1	B
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 ₂	(q ₂ , 1, L)	(q ₂ , 0, L)	(q ₃ , B, R)
q ₃	final state		

Computation of string 1011 is as follows:

$$\begin{aligned}
 q_0 \text{ 1011 B} &\vdash 1 q_0 \text{ 011 B} \vdash 10 q_1 \text{ 11 B} \vdash 101 q_0 \text{ 1 B} \\
 &\vdash 1011 q_0 \text{ B} \vdash 101 q_1 \text{ 1 B} \vdash 10 q_2 \text{ 11 B} \vdash 1 q_2 \text{ 001 B} \\
 &\vdash q_2 \text{ 1101 B} \vdash q_2 \text{ B 0101 B} \vdash B q_3 \text{ 0101 B} \rightarrow \text{accept.}
 \end{aligned}$$

State q_0 moves the machine to the right end of the input. State q_1 moves the machine back till it reaches to rightmost 1. In q_2 machine moves back complementing each input symbol, till it reaches blank and we get 2's complement of string.

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

- Which is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars and was invented in 1936 by Alan Turing?
(a) Turing Machine (TM) (b) Finite Machine (FM)
(c) Grammar Machine (GM) (d) Language Machine (LM)
- The TM, is defined by how many tuples?
(a) 5 (b) 8
(c) 7 (d) 6
- A TM consists of,
(a) input tape (b) read-write (R/W) head
(c) finite control (d) All of the mentioned
- Which TM has multiple tapes where each tape is accessed with a separate head?
(a) Multi-tape (b) Multi-track
(c) nondeterministic TM (d) Two-way TM
- Which is a multi-track non-deterministic TM with a tape of some bounded finite length?
(a) Linear Finite Automaton (LFA) (b) Linear Bounded Automaton (LBA)
(c) Context Free Automaton (CFA) (d) None of the mentioned
- A multi-track TM can be formally described as,
(a) 6 tuples (b) 8 tuples
(c) 7 tuples (d) 5 tuples
- The mathematical notation for TM is,
(a) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ (b) $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \varphi, \$, F)$
(c) $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ (d) None of the mentioned
- The ID of the TM remembers the following at a given instance of time:
(a) The cell currently being scanned by the read-write head
(b) The state of the machine, and
(c) The contents of all the cells of the tape, starting from the rightmost cell up to at least the last cell, containing a non-blank symbol and containing all cells up to the cell being scanned
(d) All of the mentioned

9. The TM accepts all the language even though they are recursively,
 (a) context-free (b) regular
 (c) enumerable (d) All of the mentioned

Answers

1. (a)	2. (c)	3. (d)	4. (a)	5. (b)	6. (a)	7. (c)	8. (d)	9. (c)	
--------	--------	--------	--------	--------	--------	--------	--------	--------	--

Q.II Fill in the Blanks:

1. A _____ is a mathematical model consists of an infinite length tape divided into cells on which input is given.
2. In TM each cell can store only _____ symbol.
3. A _____ is a nondeterministic TM which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string.
4. A Turing Machine (TM) accepts a language if it enters into a final state for any input string w and a language is recursively _____ (generated by Type-0 grammar) if it is accepted by a TM.
5. The Turing machine was proposed by A.M. Turing in 1936 is the machine format of _____ language i.e., all types of languages are accepted by the TM.
6. The instantaneous description (ID) of a Turing machine remembers the contents of all cells from the rightmost to at least the leftmost, the cell currently being scanned by the read-write head and the state of the machine at a given _____ of time.
7. The Turing machine, in short TM, is defined by 7 tuples $M =$ _____.
8. In _____ multi-track TM a single tape head reads n symbols from n tracks at one step. It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.
9. The computation of a non-deterministic TM is a tree of configurations that can be reached from the _____ configuration.
10. According to the _____ hierarchy, type 0 language is called as unrestricted language (URG)
11. A linear bounded automaton can be defined as an 8-tuple $M =$ _____.

Answers

1. Turing Machine (TM)	2. one	3. Linear Bounded Automaton (LBA)	4. enumerable
5. unrestricted	6. instance	7. $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$	8. multi-track
9. start	10. Chomsky	11. $(Q, \Sigma, \Gamma, \delta, q_0, B, \epsilon, \$, F)$	

Q.III State True or False:

1. As an automaton, the Turing machine is the most general model which accepts type-0 languages.
2. In LBA because a linear function is used to restrict (to bound) the length of the tape.
3. The mechanical diagram of the Turing machine consists of the input tape, the finite control and the read-write head.
4. A TM can be formally described as a 8-tuples.
5. TM is a machine (automaton) capable of enumerating some arbitrary subset of valid strings of an alphabet; these strings are part of a recursively enumerable set.
6. The set of strings accepted by nondeterministic LBA is the set of strings generated by the context-sensitive grammars, excluding the null strings.
7. If the TM reaches the final state, the input string is accepted, otherwise rejected.
8. A language is recursive if it is decided by a Turing Machine (TM).
9. Multi-track TM, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks.
10. A Multi-tape Turing machine can be formally described as a 7 tuple.
11. In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. T
12. A linear bounded automaton is a multi-track deterministic Turing machine with a tape of some bounded finite length.
13. A context-sensitive language is equivalent to a linear bounded nondeterministic Turing machine, also called a linear bounded automaton.
14. All types of languages are accepted by TM.
15. Computer is an example of TM.
16. Text editor is an example of TM.
17. A TM has a tape of infinite length on which it can perform read and write operations.
18. For TM, tape head moves to only right side.
19. CSL's can be accepted by LBA.

Answers

1. (T)	2. (T)	3. (T)	4. (F)	5. (T)	6. (T)	7. (T)	8. (T)	9. (T)	10. (F)
11. (T)	12. (F)	13. (T)	14. (T)	15. (T)	16. (F)	17. (T)	18. (F)	19. (T)	

Q.IV Answer the following Questions:**(A) Short Answer Questions:**

1. Define Turing machine.
2. Define LBA.
3. What is two-way TM?
4. Define nondeterministic TM.
5. List problems for language recognizers.
6. What is multi-tape TM?
7. Define multi-track TM.
8. List types of TM
9. Compare multi-tape TM and multi-track TM (any two points).

(B) Long Answer Questions:

1. What is TM? How it works? Explain its model diagrammatically.
2. What is LBA? Describe in detail.
3. Construct a TM to accept the following languages:
 - (i) $\{0^n 1^n 0^n \mid n \geq 1\}$
 - (ii) $\{ww^R \mid w \text{ is in } (0 + 1)^*\}$
 - (iii) $L = (a + b)^* (abc)$
 - (iv) The set of strings with an equal number of 0's and 1's.
 - (v) $\{0^p 1^q 2^{p+q} \mid p, q \geq 1\}$
4. Design a TM to perform proper subtraction of two unary numbers.
5. Design a TM to check palindrome.
6. Design a TM which recognizes the following languages:
 - (i) $L = \{0^m 1^m 2^{m+2} \mid m \geq 1\}$
 - (ii) $L = \{a^n b a^n \mid n \geq 0\}$
 - (iii) $L = \{a^n b^n a^n \mid n \geq 1\}$
7. Construct a TM for accepting well-formed parenthesis over $\{(,)\}$.
8. Construct a TM which recognizes a language which contains equal number of a's and b's.
9. With the help of diagram basic model of LBA.
10. Write a shot note on: Language accepted by TM.
11. What is non-deterministic TM? Explain with example.
12. With the help of example explain multi-tape TM.

13. What is two-way TM? Describe with diagram and example.
14. What is multi-track TM? State its use. Also explain with example.
15. Construct TM for $L = \{anbm \mid n, m \geq 1\}$.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Define tuples of LBA. **[1 M]**
- Ans.** Refer to Section 5.3.
2. Explain the types of Turing Machine (TM). **[4 M]**
- Ans.** Refer to Section 5.2.

October 2016

1. Define Multi-tape Turing Machine. **[1 M]**
- Ans.** Refer to Section 5.2.2.
2. Construct a TM for a language L, where $L = \{a^m b^n c^n \mid m, n \geq 1\}$ **[5 M]**
- Ans.** Refer to Section 5.1.3, Examples.
3. Differentiate between TM and LBA. **[3 M]**
- Ans.** Refer to Page 5.20.

April 2017

1. Write the tuples of Turing Machine. **[1 M]**
- Ans.** Refer to Section 5.1.2.
2. Construct a TM for $L = \{wcw^R \mid w \in (a + b)^*\}$. **[5 M]**
- Ans.** Refer to Section 5.1.3, Examples.

October 2017

1. Define non-deterministic Turing Machine. **[1 M]**
- Ans.** Refer to Section 5.2.4.

April 2018

1. Design a TM to recognize well-formedness of parenthesis (). **[4 M]**
- Ans.** Refer to Page 5.1.3.

October 2018

1. Define Turing Machine (T.M.). **[1 M]**
- Ans.** Refer to Section 5.1.2.

-
2. State two differences between TM and LBA.

[1 M]

Ans. Refer to Page 5.20.

April 2019

1. Write the tuples of LBA.

[1 M]

Ans. Refer to Section 5.3.

2. Construct TM for language: $L = \{a^m b^n \mid n > m, m > 1\}$

[5 M]

Ans. Refer to Section 5.1.3, Examples.

■■■