THEORY OF COMPUTATION

 $\underline{\hat{o}}(q_0, a, Z) = (q_0, \widetilde{A}_1 \mathcal{A}_2) = (q_0, Z_0) \text{ with two states } (q_0, q_1)$

Dr. D. Jagadeesan

Recursively Enumerable

Context-Sensitive

JVK PUBLICATIONS

(JVK Printers) Vengikkal, Tiruvannamalai Tamil Nadu, India

THEORY OF COMPUTATION

Dr. D. Jagadeesan

The Apollo University, Chittoor, Andhra Pradesh

JVK PUBLICATIONS

(JVK Printers) Vengikkal, Tiruvannamalai Tamil Nadu, India

Theory of Computation

Copyright © 2024 JVK Publications

JVK Printers 1799, Sri Raghavendra Nagar, Vengikkal, Tiruvannamalai, Tamil Nadu, India - 606604

While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book. The authors and publishers have attempted to trace the copyrights holders of all material reproduced in this publication and apologize to copy right holders if permission to publish in this form bas not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, without either the prior written permission of the Publisher, or author.

Price: ₹150.00

First Edition, 2024 ISBN 978-93-5620-409-6



INDEX

Chapter	Contents	Page No	
1	Finite Automata		
	1.1 History	6	
	1.2 Finite State Systems	6	
	1.3 Applications of Finite Automata	7	
	1.4 Basic Terms and Definitions	8	
	1.5 Finite Automata without Output	10	
	1.6 DFA vs NFA	14	
	1.7 Problems for Finite Automata	14	
	1.8 Equivalence of NFA and DFA	17	
	1.9 Equivalence of NFA with and		
	without ε transitions	22	
	1.10 Minimization of DFA	28	
	1.11 Finite Automata with Output	32	
	1.12 Transforming Mealy Machine into		
	Moore Machine	35	
	1.13 Transforming Moore Machine into		
	Mealy Machine	40	
2	Regular Expression		
	2.1 Regular language	43	
	2.2 Regular Expression	43	
	2.3 Relationship between Finite		
	Automata and Regular Expression	46	
	2.4 Equivalence of Regular Expression		
	and Finite Automata	46	
	2.5 Regular Expression from DFA	51	
	2.6 Pumping Lemma	59	
	2.7 Closure Properties for Regular	62	
	Language		
3	Regular Grammar		
	3.1 Chomsky Hierarchy	64	
	3.2 Scope of Each type of Grammar	65	
	3.3 Regular Grammer	66	
	67		

Theory of Computation

3.6 Left linear Grammar into Right Linear Grammar 67 3.7 Equivalence of Regular Grammar and Finite Automata 69 3.8 Finite Automata to Right Linear Grammar 69 3.9 Right Linear Grammar to Finite Automata 74 3.10 Finite Automata to Left Linear Grammar 77 3.11 Left Linear Grammar to Finite Automata 80 4 Context Free Grammar 4.1 Motivation and Introduction 4.2 Formal Definition 4.3 Conversion of CFG into CFL 4.4 Conversion of CFG into CFL 4.5 Derivation Tree 4.6 Derivation Tree 4.7 Ambiguity 4.8 Simplification of CFG 4.9 Normal Forms 100 5 Pushdown Automata 108 5.1 Formal Definition 108 5.2 Model of PDA 109 5.4 Equivalence of Acceptance of PDA 109 5.4		3.5	Types of Regular Grammars	67
3.7 Equivalence of Regular Grammar and Finite Automata to Right Linear Grammar 69 3.8 Finite Automata to Right Linear Grammar 69 3.9 Right Linear Grammar to Finite Automata 74 3.10 Finite Automata to Left Linear Grammar 77 3.11 Left Linear Grammar to Finite Automata 80 4 Context Free Grammar 80 4.1 Motivation and Introduction 85 4.2 Formal Definition 85 4.3 Conversion of CFG into CFL 86 4.4 Conversion of CFG into CFG 87 4.5 Derivation Tree 92 4.7 Ambiguity 94 4.8 Simplification of CFG 95 4.9 Normal Forms 100 5 Pushdown Automata 108 5.1 Formal Definition 108 5.2 Model of PDA 108		3.6	Left linear Grammar into Right	
and Finite Automata693.8Finite Automata to Right Linear Grammar693.9Right Linear Grammar to Finite Automata743.10Finite Automata to Left Linear Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar854.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFG into CFL864.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Final State to Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.10Pumping Lemma129			Linear Grammar	67
3.8 Finite Automata to Right Linear Grammar 69 3.9 Right Linear Grammar to Finite Automata 74 3.10 Finite Automata to Left Linear Grammar 77 3.11 Left Linear Grammar to Finite Automata 80 4 Context Free Grammar 4.1 Motivation and Introduction 85 4.2 Formal Definition 85 4.3 Conversion of CFG into CFL 86 4.4 Conversion of CFL into CFG 4.5 Derivation Tree 92 4.7 Ambiguity 94 4.8 Simplification of CFG 95 4.9 Normal Forms 100 5 Pushdown Automata 100 5 Pushdown Automata 109 5.1 Formal Definition 108 5.2 Model of PDA 108 5.3 Acceptance of Acceptance of PDA 109 5.4 Equivalence of Acceptance of PDA 109 5.4 Equivalence of Acceptance of PDA 110 </td <td></td> <td>3.7</td> <td>Equivalence of Regular Grammar</td> <td></td>		3.7	Equivalence of Regular Grammar	
Grammar693.9Right Linear Grammar to Finite Automata743.10Finite Automata to Left Linear Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804Conversion of CFG into CFL864.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			and Finite Automata	69
3.9Right Linear Grammar to Finite Automata743.10Finite Automata to Left Linear Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804Conversion of CFG into CFL864.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		3.8	Finite Automata to Right Linear	
Automata743.10Finite Automata to Left Linear Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar804Context Free Grammar804Context Free Grammar804.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			Grammar	69
3.10Finite Automata to Left Linear Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar804.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFG into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		3.9	Right Linear Grammar to Finite	
Grammar773.11Left Linear Grammar to Finite Automata804Context Free Grammar804Context Free Grammar804.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			Automata	74
3.11Left Linear Grammar to Finite Automata804Context Free Grammar4.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		3.10	Finite Automata to Left Linear	
Automata804Context Free Grammar4.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			Grammar	77
4 Context Free Grammar 4.1 Motivation and Introduction 85 4.2 Formal Definition 85 4.3 Conversion of CFG into CFL 86 4.4 Conversion of CFL into CFG 87 4.5 Derivation 89 4.6 Derivation Tree 92 4.7 Ambiguity 94 4.8 Simplification of CFG 95 4.9 Normal Forms 100 5 Pushdown Automata 108 5.1 Formal Definition 108 5.2 Model of PDA 108 5.3 Acceptance by PDA 109 5.4 Equivalence of Acceptance of PDA 109 5.4 Equivalence of Acceptance of PDA 113 5.7 Equivalence of Acceptance of PDA 113 5.7 Equivalence of PDA and CFL 119 5.8 Deterministic PDA 128 5.9 Non- Deterministic PDA 128 5.10 Pumping Lemma 129		3.11	Left Linear Grammar to Finite	
4.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			Automata	80
4.1Motivation and Introduction854.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		C		
4.2Formal Definition854.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129	4			05
4.3Conversion of CFG into CFL864.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
4.4Conversion of CFL into CFG874.5Derivation894.6Derivation Tree924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1005Pushdown Automata1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
4.5Derivation of CFD Intervet4.5Derivation Of CFD Intervet4.6Derivation Tree924.74.7Ambiguity4.8Simplification of CFG954.94.9Normal Forms1005Pushdown Automata5.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDAfrom Empty Stack to Final State1105.5Equivalence of Acceptance of PDAfrom Final State to Empty Stack1135.7Equivalence of PDA and CFL5.8Deterministic PDA5.9Non- Deterministic PDA5.10Pumping Lemma129				
4.6Derivation924.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
4.7Ambiguity944.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
4.8Simplification of CFG954.9Normal Forms1005Pushdown Automata1085.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
4.9Normal Forms1005Pushdown Automata5.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				
5Pushdown Automata5.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			-	
5.1Formal Definition1085.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		4.9	Normal Forms	100
5.1Formar Definition1085.2Model of PDA1095.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129	5	Push	ndown Automata	
5.2Model of PDA1085.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129		5.1	Formal Definition	108
5.3Acceptance by PDA1095.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				108
5.4Equivalence of Acceptance of PDA from Empty Stack to Final State1105.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				109
from Empty Stack to Final State 110 5.5 Equivalence of Acceptance of PDA from Final State to Empty Stack 112 5.6 Design of PDA 113 5.7 Equivalence of PDA and CFL 119 5.8 Deterministic PDA 128 5.9 Non- Deterministic PDA 128 5.10 Pumping Lemma 129				
5.5Equivalence of Acceptance of PDA from Final State to Empty Stack1125.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				110
from Final State to Empty Stack1125.6 Design of PDA1135.7 Equivalence of PDA and CFL1195.8 Deterministic PDA1285.9 Non- Deterministic PDA1285.10 Pumping Lemma129		5.5		
5.6Design of PDA1135.7Equivalence of PDA and CFL1195.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129				112
5.7 Equivalence of PDA and CFL1195.8 Deterministic PDA1285.9 Non- Deterministic PDA1285.10 Pumping Lemma129		5.6		113
5.8Deterministic PDA1285.9Non- Deterministic PDA1285.10Pumping Lemma129			6	119
5.9Non- Deterministic PDA1285.10Pumping Lemma129			-	128
5.10 Pumping Lemma 129				128
				129
5.11 Closure Properties of CFL 130				130

6	Tur	Turing Machine				
	6.1	Formal Definition	131			
	6.2	Model of Turing Machine	131			
	6.3	Design of TM	133			
	6.4	Computable Languages and				
		Functions	140			
	6.5	Modification of Turing Machine	144			
7	Und	Undecidability				
	7.1	Properties of Recursive and				
		Recursively Enumerable Languages	150			
	7.2	Post Correspondence Problem				
		(PCP)	152			
	7.3	Rice Theorem	154			
	7.4	Halting Problem	156			

FINITE AUTOMATA

Formal languages and Automata Theory describes the basic ideas and models underlying computing. It suggests various abstract models of computation, represented mathematically.

1.1 History

Mathematicians and logicians made major contributions to the field of finite automata as early as the 20th century. The concept was first introduced by David Hilbert in 1927, but it was the work of Alonzo Church and Alan Turing in the 1930s that laid the theoretical foundation for finite automata as models of computation. In the mid-20th century, the renowned mathematician and computer scientist John von Neumann made substantial contributions to automata theory. The formalization of deterministic and nondeterministic finite automata emerged in the 1950s, with notable contributions from Michael O. Rabin and Dana Scott. Finite automata became integral to theoretical computer science, playing a crucial role in formal language theory and compiler design. The development of finite automata reflects a collaborative effort by pioneers in mathematics and computer science to understand the fundamental principles of computation and lay the groundwork for subsequent advances in the field.

1.2 Finite State systems

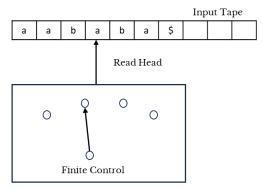
A finite automaton can also be thought of as the device shown below consisting of a tape and a control circuit which satisfy the following conditions:

- ✓ The tape has the left end and extends to the right without an end.
- ✓ The tape is dividing into squares in each of which a symbol can be written prior to the start of the operation of the automaton.
- ✓ The tape has a read only head.

- ✓ The head is always at the leftmost square at the beginning of the operation.
- ✓ The head moves to the right one square every time it reads a symbol.

It never moves to the left. When it sees no symbol, it stops and the automaton terminates its operation.

✓ There is a finite control which determines the state of the automaton and also controls the movement of the head.

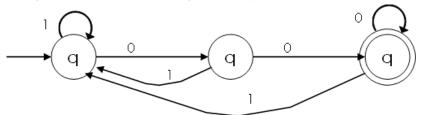


Finite Automaton

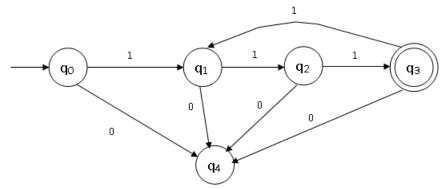
1.3 Applications of Finite Automata

- ✓ Finite automata are extensively used in compilers for lexical analysis, the first phase of language processing.
- ✓ It is useful in text processing and searching applications.
- ✓ Finite state machines are utilized to model and analyze network protocols.
- ✓ Finite state machines are used to design and model digital circuits.
- ✓ Finite automata contribute to pattern recognition tasks, where recognizing specific sequences or patterns in data is essential.
- ✓ Finite state machines are employed to model and control the behavior of robots.
- ✓ Finite state machines play a crucial role in designing the control units of digital systems within these circuits.
- ✓ Finite state machines and automata models are used in designing decision-making processes for AI agents.
- ✓ Finite state machines are commonly used in game development for modeling the behavior of non-player characters.

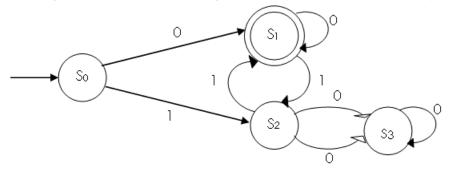
3. Design FA to accept the string that always ends with 00.



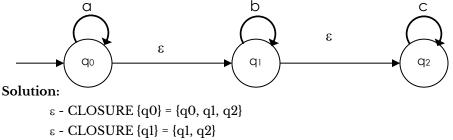
4. Design FA to check whether a given unary number is divisible by 3.



5. Design FA to check whether a given binary number is divisible by 3.



6. Obtain the ϵ closure of states q0 and q1 in the following NFA with ϵ transition.



Now we will show that

 $\delta'(p,a) = \delta(q0,wa)$

But,

$$\delta'(p,a) = \delta'(q,a) = \delta''(q,a)$$
 as $p = \delta''(q0,w)$

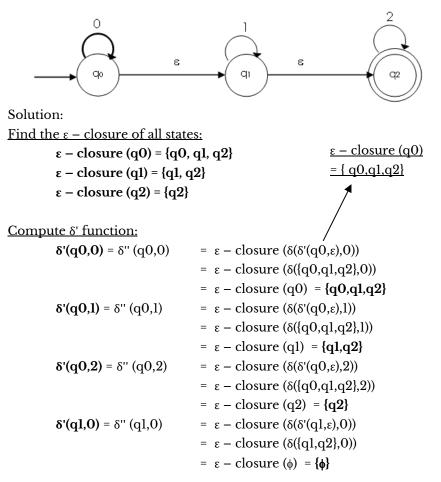
We have

 $\delta''(q,a) = \delta''(q0,wa)$

Thus, by definition of
$$\delta$$
"
 $\delta'(q0, wa) = \delta''(q0, wa)$

1.9.1 Problems for Converting NFA with ε into NFA without ε

1. Construct NFA without ε from NFA with ε.



REGULAR EXPRESSION

A regular expression plays a crucial role in describing languages. Let's explore what regular expressions are and how they relate to regular languages.

2.1 Regular Language

A language is called regular language if there exists a finite automaton that recognizes it. For example, finite automaton M recognizes the language L if $L = \{w \mid M \text{ accepts } w\}$.

2.1.1 Operations on Regular Languages

Let A and B be languages. We define regular language operations union, concatenation and closure as follows:

Union	$: A \cup B = \{x \mid x \in A \lor x \in B\}$
Concatenation	$: A \circ B = \{ xy \mid x \in A \land y \in B \}$
Closure	$: A^* = \{x1x2\ldots xk \mid k \ge 0 \land xi \in A, 1 \le i \le k\}$

2.2 Regular Expression

A regular expression is a string that defines a finite pattern of strings or symbols. Each pattern corresponds to a set of strings, serving as a name for that set.

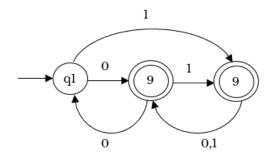
Regular expressions are used to describe languages, and they allow us to express rules for constructing valid strings within those languages.

2.2.1 Operations on Regular Expressions

Let Σ be an alphabet. The regular expressions over Σ and the sets that they denote are defined recursively as follows:

- \checkmark Ø is a regular expression and denotes the empty set {}.
- \checkmark ϵ is a regular expression and denotes the set { ϵ }.

- 2.5.1.2 Problems
 - 1. Covert the given finite automata into a regular expression using substitution method.



$$\mathbf{r} = r_{12}^3 + r_{13}^3$$

$$r_{ij}^{0} = \begin{cases} a1 + a2 + a3 + \dots + an & [\delta(qi, a) = qj] \text{ if } i \neq j \\ a1 + a2 + a3 + \dots + an + \varepsilon & [\delta(qi, a) = qj] \text{ if } i = j \end{cases}$$

<u>K= 0</u>

<u>K= 1</u>

$$r_{ij}^{k} = r_{ij}^{k-1} + r_{ik}^{k-1} (r_{kk}^{k-1})^{*} r_{kj}^{k-1}$$

$$r_{11}^{1} = r_{11}^{0} + r_{11}^{0} (r_{11}^{0})^{*} r_{11}^{0} = \varepsilon + \varepsilon (\varepsilon)^{*}\varepsilon = \varepsilon + \varepsilon = \varepsilon$$

$$r_{12}^{1} = r_{12}^{0} + r_{11}^{0} (r_{11}^{0})^{*} r_{12}^{0} = 0 + \varepsilon (\varepsilon)^{*} 0 = 0 + 0 = 0$$

$$r_{13}^{1} = r_{13}^{0} + r_{11}^{0} (r_{11}^{0})^{*} r_{13}^{0} = 1 + \varepsilon (\varepsilon)^{*}\varepsilon 1 = 1 + 1 = 0$$

$$r_{21}^{1} = r_{21}^{0} + r_{21}^{0} (r_{11}^{0})^{*} r_{11}^{0} = 0 + 0 (\varepsilon)^{*}\varepsilon = 0 + 0 = 0$$

$$r_{22}^{1} = r_{22}^{0} + r_{21}^{0} (r_{11}^{0})^{*} r_{12}^{0} = \varepsilon + 0 (\varepsilon)^{*}0 = \varepsilon + 00$$

$$r_{23}^{1} = r_{23}^{0} + r_{21}^{0} (r_{11}^{0})^{*} r_{13}^{0} = 1 + 0 (\varepsilon)^{*}1 = 1 + 01$$

REGULAR GRAMMAR

Language: "A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols."

Grammar: "A grammar can be regarded as a device that enumerates the sentences of a language."

A formal grammar is a quad-tuple G = (N, T, P, S)

where

N is a finite set of non-terminals

T is a finite set of terminals

P is a finite set of production rules of the form $\alpha\:A\:\beta\to\alpha\:\gamma\:\beta$

Where α , β , $\gamma \in (N \cup T)$ *, $A \in N \neq \epsilon$

 $S \in N$ is the start symbol

3.1 Chomsky Hierarchy (Type of Grammars)

Class	Grammars	Languages	Automaton	Rules
Type-0	Unrestricted	Recursively	Turing	$\alpha \rightarrow \beta$
	Grammar	Enumerable	Machine	$\alpha, \beta \in (N \cup T)*$
		Language		$\alpha\neq\epsilon$
Type-1	Context	Context	Linear	$\alpha \: A \: \beta \to \alpha \: \gamma \: \beta$
	Sensitive	Sensitive	Bounded	$\alpha, \beta, \gamma \in (N \cup T) *$
	Grammar	Language	Automaton	$A \in N \neq \varepsilon$
Type-2	Context-	Context-	Pushdown	$A \rightarrow \alpha$
	free	free	automaton	where $A \in N$
	Grammar	Language		$\alpha \in (N \cup T) \ast$
Type-3	Regular	Regular	Finite	$A \rightarrow \alpha B$
	Grammar	Language	automaton	$A \rightarrow B \alpha$
				$A \rightarrow \alpha$
				where A, $B \in N$ and
				$\alpha \in T^*$

CONTEXT FREE GRAMMAR

A context-free grammar is a collection of recursive rules employed to produce patterns of strings. A context-free grammar is capable of describing all regular languages and additional languages, but not all possible languages. Context-free grammars are an area of study in the fields of theoretical computer science, compiler design and linguistics.

4.1 Motivation and Introduction

Formal language theory and computer science use a context-free grammar (CFG) as a formalism to describe a language's syntax. A set of production rules define the replacement of symbols or non-terminals with sequences of other symbols and terminals.

A Context Free Grammar is consisting of four components. They are finite set of non-terminals, finite set of terminals, set of productions and start symbol.

4.2 Formal Definition of Context Free Grammars (CFG)

• A CFG is a mathematical object, G, with four components, G = (N, T, P, S)

where

N is a nonempty, finite set of non-terminal symbols

T is a finite set of terminal symbols

P is a set of grammar rules as per below form

 $\label{eq:alpha} \begin{array}{ll} A \to \alpha & \text{where } A \in N \text{ and } \ \alpha \in (N \cup T)^* \\ S \text{ is the start symbol } S \in N \end{array}$

• Example

Let G = ({S},{0,1, ϵ },P,S) be a CFG, where productions are S \rightarrow 0S0/1S1/ ϵ

PUSHDOWN AUTOMATA

A pushdown automaton (PDA) is a type of automaton, which is a mathematical model of computation. PDAs are used to recognize context-free languages, which are languages generated by context-free grammars. A PDA consists of Input tape, Finite control, Stack. The transitions of a PDA are determined by the current state, the symbol read from the input tape, and the symbol popped from the stack. Based on these factors, the PDA can change its state, push symbols onto the stack, or pop symbols from the stack. PDAs can be in one of three states: accepting, rejecting, or non-accepting. The PDA accepts the input string if, after processing the entire string, it is in an accepting state. Otherwise, it rejects the input string.

5.1 Formal Definition of Pushdown Automaton

A pushdown automaton consists of seven tuples

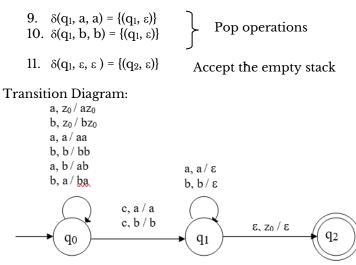
M = (Q, Σ , Γ , δ , q₀, Z₀, F), where

- Q Finite set of states
- Σ Finite input alphabet
- Γ Finite alphabet of pushdown symbols
- δ Transition function Q × (Σ ∪ {ε}) × Γ → Q×Γ
- q_0 start / initial state $q_0 \in Q$
- Z_0 start symbol on the pushdown $Z_0 \in \Gamma$
- F~ set of final states $F \in Q$

5.2 Model of PDA

Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

The PDA consists of a finite set of states, a finite set of input symbols and a finite set of push down symbols. The finite control has control of both the input tape and the push down store. The stack head scans the top symbol of the stack.



7. Design a PDA that accepts $L = \{ww^R : w \in (0+1)^*\}$ accepted by final state. (or) Design a PDA for even length palindrome.

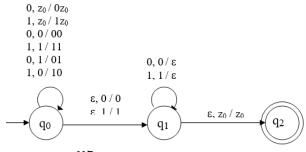
Solution:

Let M = (Q, Σ , Γ , δ , q₀, Z₀, F) be a PDA

The productions are:

-	° P -		~	
	1.	$\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$))	
	2.	$\delta(q_0, 1, z_0) = \{(q_0, 1z_0)\}$		
	3.	$\delta(q_0, 0, 0) = \{(q_0, 00)\}$	l	➤ Push operations
	4.	$\delta(q_0, 1, 1) = \{(q_0, 11)$	Ì	
	5.	$\delta(\mathbf{q}_0, 0, 1) = \{(\mathbf{q}_0, 01)\}$		
	6.	$\delta(q_0, 1, 0) = \{(q_0, 10)\}$	J	
			\leq	
	7.	$\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$	L	A
	8.	$\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$	ſ	Accept the separator 'ε'
			Ś	
	9.	$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$	Ļ	Pop operations
	10.	$\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$		
			-	
	11.	$\delta(q_1, \epsilon, z_0) = \{(q_2, z_0)\}$		Accept the Final State

Transition Diagram:



TURNING MACHINE

A Turing machine (TM) is a theoretical model of computation introduced by Alan Turing in 1936. It is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars.

A Turing Machine (TM) is a mathematical model which consists of

- An **infinite length tape** divided into cells; each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right.
- A head which reads the input tape.
- A state register stores the state of the Turing machine.
- 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.

6.1 Formal Definition of Turing Machine

A TM can be formally described as a 7-tuple M = (Q, Σ , Γ , δ , q_0 , B, F) where

 $\begin{array}{l} Q \text{ is a finite set of states} \\ \Sigma \text{ is the input alphabet} \\ \Gamma \text{ is the tape alphabet} \\ \delta \text{ is a transition function; } \delta \text{: } Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}. \\ q_0 \text{ is the initial state, } q_0 \in Q \\ B \text{ is the blank symbol, } B \in \Gamma \\ F \text{ is the set of final states, } F \in Q \end{array}$

6.2 Model of Turing Machine (TM)

Turing Machine has three components:

- i. Finite state control:
 - It is in one of a finite number of states at each instant, and is connected to the tape head.

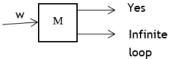
UNDECIDABILITY

Undecidability is a concept in mathematics and computer science that refers to situations where there is no algorithmic way to determine whether a given statement is true or false within a particular formal system. Undecidability is a concept in mathematics and computer science that refers to situations where there is no algorithmic way to determine whether a given statement is true or false within a particular formal system. This concept often arises in the study of formal languages, logic, and computational theory. Alan Turing famously proved the halting problem, a classic example of undecidability, in 1936. The halting problem asks whether a given program, when provided with a particular input, will eventually halt (i.e., stop running). Turing showed that it is impossible to write a general algorithm that can decide whether any arbitrary program will halt or run forever.

Recursive Language: A language is recursive if there exists a Turing Machine that accepts every string of the language and reject every string that is not in the language.

$$\xrightarrow{w} M \longrightarrow No$$

Recursively Enumerable Language: A language is recursive enumerable if there exists a Turing Machine that accepts every string of the language and does not accept strings that are not in the language. The strings that are not in the language may be rejected and it may cause the TM to go to an infinite loop.





Accepted here

Scan and Pay using bob world



*send payment proof to 7904647133 after verification of payment publisher will send the softcopy of book



JVK PUBLICATIONS

(JVK Printers) Vengikkal, Tiruvannamalai Tamil Nadu, India