IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⪤ Vol. 11, Issue 1, January 2022

DOI: 10.17148/IJARCCE.2022.11106

Introduction of Automata, Analyzation of Regular Language, Grammar and Computation

Barkha Gupta[1]

Programme Assistant, Department of Computer, Agriculture University, Jodhpur, Rajasthan, India[1]

Abstract: All real-world computers perform some sort of computations like mathematical models to solve their problems in a systematic manner. The essence of the automata theory is to help and develop mathematical and logical models that run efficiently. Since all the machines that implement logic or follows and predefined algorithm apply TOC (Theory of Computation). Thus, studying TOC gives learners an insight view of both computer hardware and software, its working and limitations.

Keywords: Theory of computation, Automata, Finite Automata, Turning Machine, Grammar, Regular Expression.

I. INTRODUCTION

Automata theory also known as theory of computation is a branch of computer science and a study of mathematical machine or systems called automata. Automata is derived from the word “Automaton” which is related to the word “Automation”. Automaton is a system where data are transformed and utilized for performing some processes without the intervention of human. It also deals with the logic of computations. In automata theory, we identify whether the problem can be solved or cannot be solved by a system and if problem can be solved then finding out the minimum effort in terms of time and space is required to solve that problem. Thus, theory of computation is concerned with how the problems can be solved using predefined algorithms and how efficiently that problem can be solved.

II. BASIC TERMINOLOGIES IN AUTOMATA THEORY

(i) **Symbol**- Basic and fundamental building blocks which can be any character, picture, token, alphabets or letter. They are also referred as character. Few examples are f, h, 7, 9 etc.

(ii) **Alphabet**- Collection of symbols and an finite non empty set of symbols. It is generally denoted by Sigma symbol ∑.
Few examples are ∑ = {a, b} and ∑ = {4,8}

(iii) **String/Word**- String is finite sequence of symbols. It is generally denoted by **w** and length of the string is denoted by **|w|**.

Few examples are	∑ = {a, b}	this is string of length 2
	∑ = {4,8.9}	this is string of length 3

(iv) **Language** – Language is a set of strings. It is generally denoted by ∑*. A language that can be formed from ∑ or nay possible subset of ∑ is a language.

III. BASIC OPERATIONS ON STRING

(i) **Lengthy of string**
Length of the string is denoted by |w|. Smallest length of string is of zero length (zero occurrence of string) represented by ε. This symbol ε is termed as epsilon.

$|w_1|$ = {a}	length of one string
$|w_2|$ = {ab}	length of two string
$|w_3|$ = {aaa}	length of three string and so on.

(ii) **Concatenation of string**
Concatenation of string implies joining the one string with the other string. Make sure that the first string will remain at the beginning and the second string will join after the first one.

Example |w1| = {abb} and |w2| = {cde}

DOI: 10.17148/IJARCCE.2022.11106

Concatenate w_1w_2 will give {abbcde}
|w1| + |w2| also represents concatenation and leads to same result.
$w_1w_2 \neq w_2w_1$ Both are not equal and hence concatenation is not commutative in nature.

(iii) **Reverse of string**

Reverse of string implies writing the string from right to left or from backward to frontward direction. It is denoted by w^R

Example |w1| = {abb}
The w^R = {bba}

(iv) **Prefix and Suffix of string**

Prefix of string is any subset of string started from left hand side and containing its left-hand characters and must be unique. Epsilon can also be included in prefix of string. It is denoted by p(w).

Example of prefix is let string is w = abaa
P(w) = { ε, a, ab, aba, abaa}
Hence, number of prefix string is N+1 where N denotes length of string.

Suffix of string is any subset of string started from right hand side and containing its right-hand characters and must be unique. Epsilon can also be included in prefix of string. It is denoted by s(w).

Example of suffix is let string is w = abaa
S(w) = { ε, a, aa, baa, abaa}
Hence, number of prefix string is N+1 where N denotes length of string.

(v) **Substring**

Substring is part of string. It extracts some characters from the string.
Example w = abbcdd

Valid Substring	**Invalid Substring**
bbc	abd
bc	abc
bcd	cba
ab	dcb

Total substring possible in any string is = n(n+1)/2 +1

(vi) **Kleene Closure**

Kleene closure is same a universal set in set theory which contains all the string including epsilon. It is denoted by $\sum^*$.

Here $2^3 \neq 8$. Here it implies string of length three.
Example 1:
$\sum$ = {a,b}
$\sum^0$ = { ε }
$\sum^1$ = {a, b}
$\sum^2$ = {aa, bb, ab, ba}
$\sum^*$ = contains all the strings so this defines

$\sum^* = \cup_{k=0}^{\infty}\{ w||w| = k\}$ K = 0 means supports null or epsilon string

$\sum^+ = \cup_{k=1}^{\infty}\{ w||w| = k\}$ K=1 means does not supports null or epsilon string

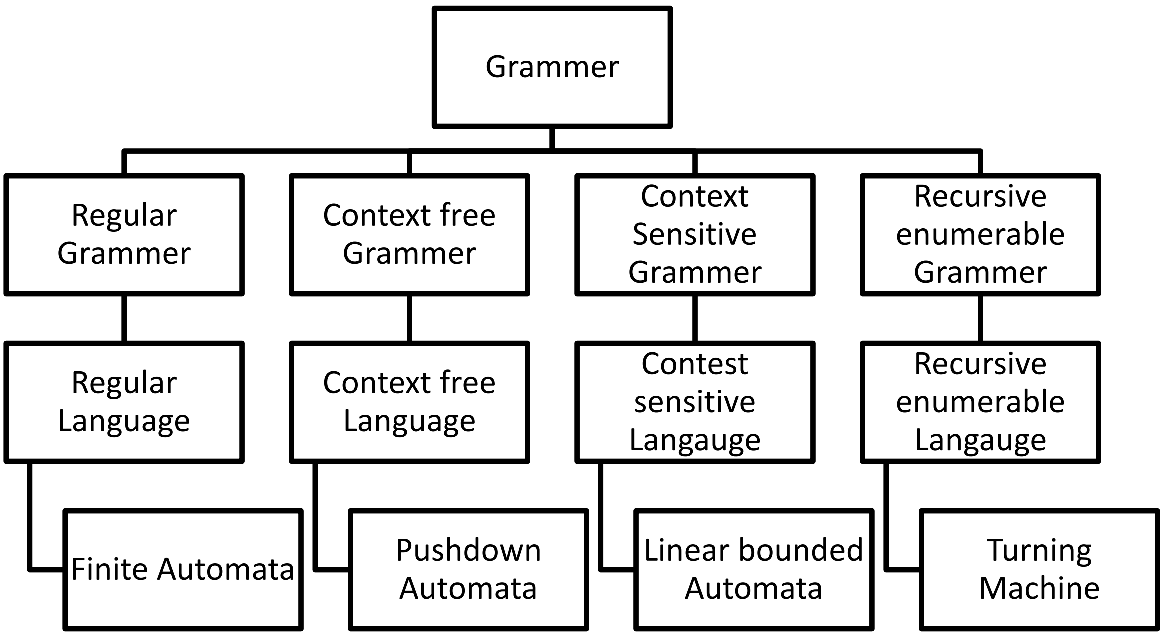
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

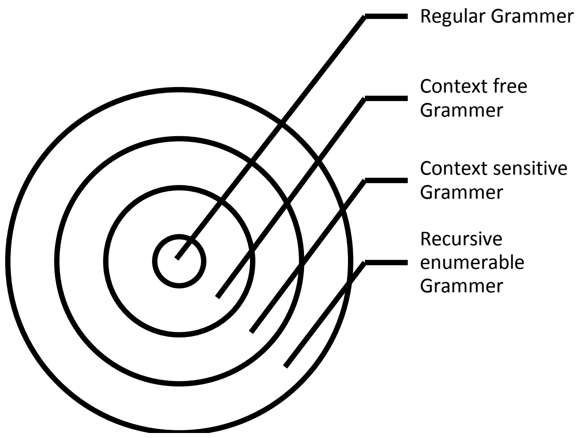
International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 1, January 2022

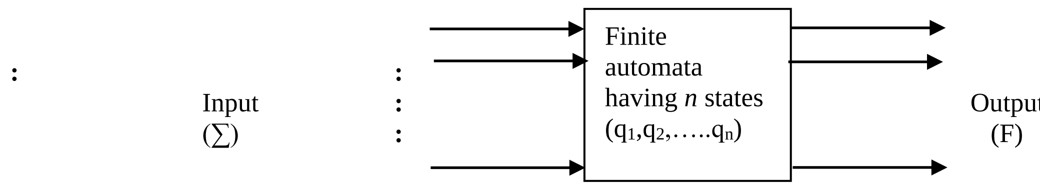
DOI: 10.17148/IJARCCE.2022.11106

IV. GRAMMAR





(i) **Finite Automata**– Finite Automata (FA) is the simplest machine to recognize patterns. The finite state machine or finite automata is an abstract machine that has five elements or tuples in it. It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol. The model of finite automata is shown below: -



Finite Automata is of two types

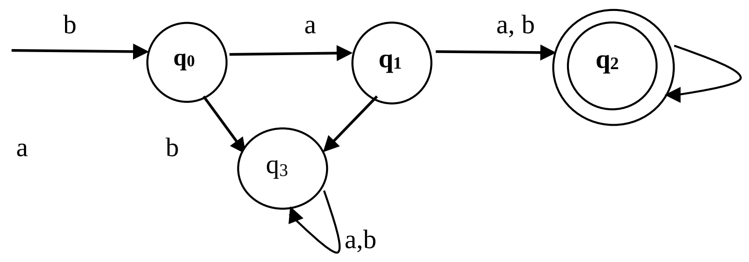
a) **DFA** – DFA is acronym of Deterministic Finite Automata. Here deterministic means knowing its current state and by giving any input or command, one can predict the next state then its is termed as deterministic. In DFA, null moves are not possible. In DFA, for each state there must be utmost one transition for each input. To make this machine deterministic, on every state it has to respond irrespective of fact that the string is accepted or rejected by the machine. DFA is having five tuples.

DFA = $\{Q, q_0, \sum, F, \delta\}$

Q = finite total number of states

q_0 = initial state
∑ = finite non-empty set of inputs
F = Set of final states
δ = transition function

Example 1 Design the DFA over ∑= {a,b} such that w='ba'.
So valid strings are { ba, baa, bab, ……}

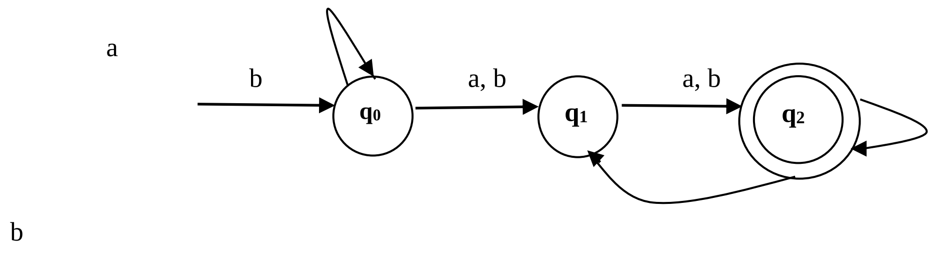


This DFA can also be represented in transition table that can be defined by transition function

	a	**b**
q_0	q_3	q_1
q_1	q_2	q_3
q_2	q_2	q_2
q_3	q_3	q_3

b) **NFA** – NFA is acronym of Non-Deterministic Finite Automata. In DFA by knowing its current state and by giving any command or input one cannot predict its forthcoming state. In NFA null moves are possible. In NFA, there can be more than one transition from a given possible input. It is described with 5 token-

NFA = {Q, q_0, ∑, F, δ}
Q = finite total number of states
q_0 = initial state
∑ = finite non-empty set of inputs
F = Set of final states
δ = transition function



This NFA can also be represented in transition table that can be defined by transition function

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

DOI: 10.17148/IJARCCE.2022.11106

V. APPLICATIONS OF FINITE AUTOMATA

(i) Word Processors programs
(ii) Spelling checkers
(iii) Lexical analyser
(iv) Text editor etc.

VI. REGULAR EXPRESSIONS

Regular expression is a way of representing regular language. An expression which is combinations of strings and operators. One way to describe regular language is DFA and NFA and another way is regular expressions.

(i) **Kleene closure** - Kleene closure is same a universal set in set theory which contains all the string including epsilon. It is denoted by *. Example is a* implies = ^, a, aa, aaa, …….. {^ implies null, length of 0 string}

(ii) **Positive closure -** It is denoted by $^+$ as a superscript (symbol written above the strings). It contains all the strings except epsilon. Example is r = a^+ then language can accept
L(r) = {a, aa, aaa, aaaa, ….. }

(iii) **Concatenation -** It is denoted by . (dot) symbol. Example is r = (ab + a).b then language will accept
L(r) = {abb, ab}

(iv) **Union -** It is denoted by +. It implies either-or option which means only one can be accepted. Example is r = a + b then language can accept either a or b.

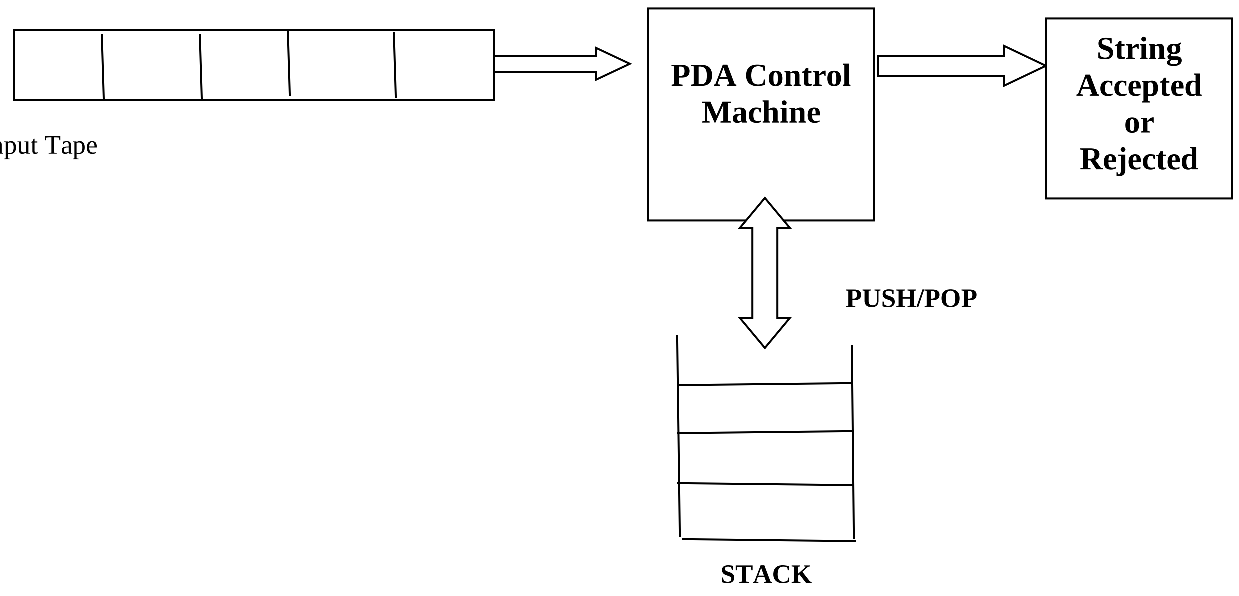
VII. PROPERTIES OF REGULAR EXPRESSION

(i) $\emptyset + R = R$
(ii) $\emptyset R + R\emptyset = \emptyset$
(iii) $\wedge R = R\wedge = R$
(iv) $\wedge^* = \wedge$ And $\emptyset^* = \wedge$
(v) $R + R = R$
(vi) $R * R^* = R^*$
(vii) $RR^* = R * R$
(viii) $(R^*)^* = R^*$
(ix) $\wedge + R\,R^* = \wedge + R^*R + R^*$
(x) $(PQ)^*\,P = P\,(QP)^*$
(xi) $(P + Q)^* = (P^*\,Q^*)^* = (P^* + Q^*)^*$
(xii) $(P + Q)\,R = PR + QR$
(xiii) $R(P + Q) = RP + RQ$

VIII. PUSHDOWN AUTOMATA

Pushdown automata is a finite automata that has extra memory element in it called as stack which helps pushdown automata to recognise context free grammar. Pushdown automaton has seven tokens

PDA = $\{Q, q_0, \sum, F, \delta, \Gamma, z_0\}$
Q = finite total number of states
q_0 = initial state
$\sum$ = finite non-empty set of inputs
F = Set of final states
δ = transition function
Γ = stack alphabet
z_0 = initial stack symbol



Pushdown automata does not have a transition table but it has a transition ID. This ID is an acronym of Instantaneous Description that computes an input string. It has three values in it (q,w, α).
Here q represents current state
w represents remaining inputs on the tape
α represents stack content which is at the top.
Turnstile Notation: ⊢ sign describes the turnstile notation and implies one move.

Pushdown automata is of two types Deterministic Pushdown Automata (DPDA) and Non-Deterministic Pushdown Automata (NDPA). Every NDPA can not be converted into DPDA. In terms of power DPDA > NDPA

In DPDA = $Q \times \sum \times \Gamma$ Q $\longrightarrow$

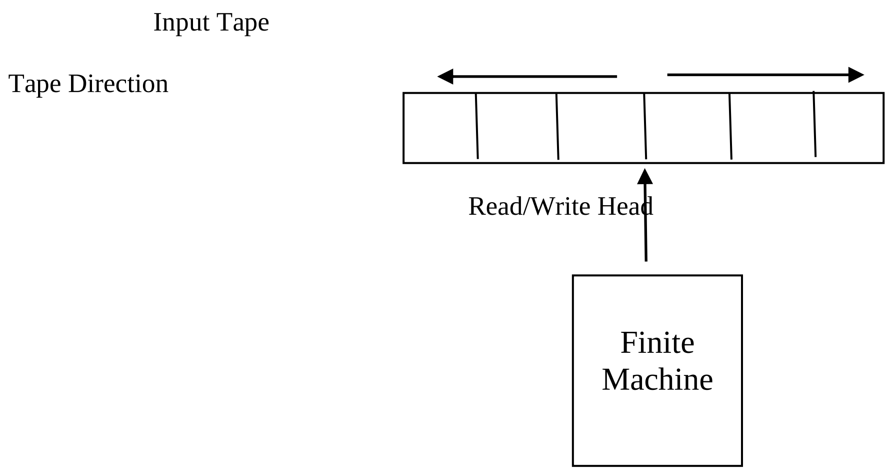
In NDPA = $Q \times \sum \times \Gamma$ 2^Q $\longrightarrow$

Example 1: Design a PDA for accepting a language $\{a^n b^n \mid n>=1\}$.

Representation in transition ID form
δ (q_0, a, z_0) = (q_0, az_0)
δ (q_0, a, a) = (q_0, aa)
δ (q_0, b, a) = (q_1, ε)
δ (q_1, b, a) = (q_1, ε)
δ (q_1, ε, z_0) = (q_f, z_0)

IX. AUTOMATA TURNING MACHINE

Turning machine in theory of computation is a mathematical model of computation that defines abstract computer which was invented in 1936 by Alan Turning. Turning machine is used to work and accept Recursive enumerable language. It is the most powerful machine in comparison with FA, PDA or LBA. It contains huge memory in terms of tape where user can read and write the string and one can also determines the movement of direction of the tape. Turning machine has seven tokens
TM = {Q, q_0, ∑, F, δ, Γ, B}
Q = finite total number of states
q_0 = initial state
∑ = finite non-empty set of inputs
F = Set of final states
δ = transition function
Γ = tape alphabet
B = denotes blank space, shows empty space on tape

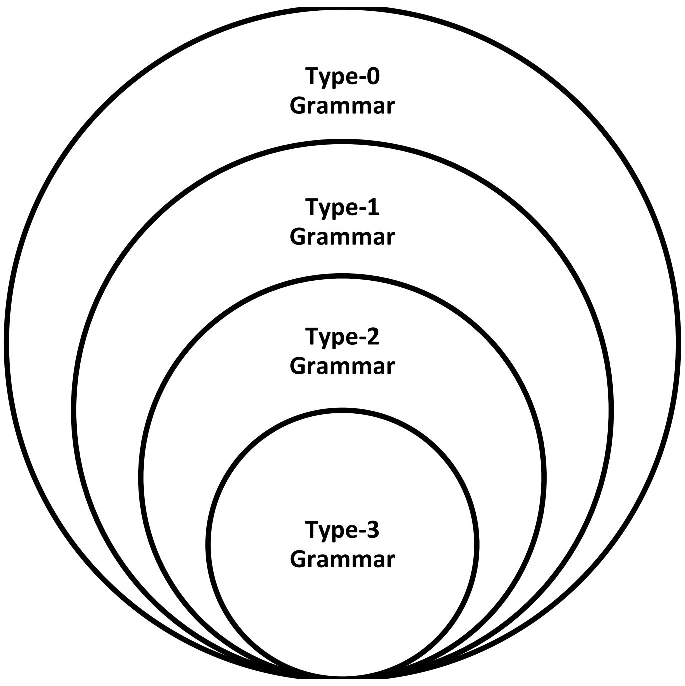


Example 1: Design Turning machine for L = {$a^n b^n$ | n>=1}
Here
Q = {q0,q1,q2,q3} where q0 is initial state.
T = {a, b,X,Y,B} where B represents blank.
∑ = {a, b}
F = {q3}

	a	b	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	(q_1, a, R)	(q2, Y, L)		(q_1, Y, R)	
q_2	(q2, a, L)		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	Halt/Stop

X. CHOMSKY HIERARCHY

Chomsky classify the grammar into four categories which are as follows:-



(i) **Type-0 Grammar** – Type-0 is an unrestricted grammar which accept all the languages. It is also known as phase structured grammar. It is used to generate recursive enumerable language which is accepted by Turning machine.

The mandatory condition for valid production in type-0 grammar is that at least one variable in left hand side should be in capital letter.

(ii) **Type-1 Grammar** – It is also known as context sensitive grammar or length increasing grammar or non-contracting grammar. It generates context sensitive languages which is accepted by linear bounded automata. The mandatory condition for valid production in type-1 grammar is that side of left-hand side should be less than or equal to size of right-hand side and right-hand side must contain at least one variable. Condition of NULL production can only be produced by start symbol and then this start symbol cannot be used in right hand side of any production.

(iii) **Type-2 Grammar** – It is also called as context free grammar. It generates context free languages which is accepted by pushdown automata. The mandatory condition for valid production in type-2 grammar is that the right-hand side can contain any number of terminals and even NULL allowed at right hand side but left-hand side must contain exactly one non-terminal letter and without having any terminal included in its production.

(iv) **Type-3 Grammar** – It is also known as regular grammar. It generates regular language which is accepted by finite automata. This restricts non-terminals on left-hand side and right-hand side can contain single terminal followed by single non-terminal.

XI. ACKNOWLEDGEMENTS

Author is grateful to Agriculture University, Jodhpur for encouraging writing in my field.

XII. REFERENCES

[1] John E. Hopcroft. Rajeev Motwani. Jeffre D. Ullman. Automata Theory Language & Computation. Third Edition. Publication Pearson Education India.
[2] Vivek Kulkarni. Theory of Computation. Publisher Oxford University Press. Illustrated Edition.
[3] Peter Linz. An introduction to Formal Languages and Automata. Publisher Jones & Bartlett. Student Edition.
[4] Michael Sipser. Theory of Computation. Publisher Thomson Press India Ltd.
[5] Rajeev Motwani. Jeffre D. Ullman. John E. Hopcroft. Introduction to Automata Theory, Languages and Computation. Third Edition. Publication Pearson Education India.
[6] B. Paramasivan. K. Mohaideen Pitchai. S. Dheenathayalan. Theory of Computation. Publishing Year 2016.
[7] Debidas Ghosh. Introduction to Theory of Automata, Formal Languages and Computation. Eastern Economy Edition.
[8] Adesh K. Pandey. An Introduction to Automata Theory & Formal Languages. Kataria SK & Sons Imprint.
[9] A. A. Puntambekar. Theory of Computation. Publisher Technical Publications. Edition First.