Chapter 1

Introduction

Formal languages are inevitable parts of Computer Science, Mathematics and Linguistics. Formal languages have their roots in Mathematics, and especially in symbolic logic. They are used as various codes needed in data encryption, transmission, and error correction. Their importance in various fields now-a-days is due to Noam Chomsky's investigation [25, 26, 27, 29], Marcel-Paul Schützenberger's algebro-combinatorial approach in formal language theory and works of Arto Salomaa [73]. In this respect the effect of programming languages should also be mentioned. The basis of every programming language lies in formal language theory. Since programming languages are flourishing day by day, together with them formal language is also attaining its importance in the field of Computer Science.
In this present work words over formal languages are studied using various mathematical tools or branches. As for example, matrix, theory of equations, analytical geometry, distance formulae, functions etc. are used . Various algorithms are also generated. Formal words are also investigated using finite state automata.

Formal language is a set of sequences of symbols that may be guarded by rules that are specific for a particular formal language. It is a model which can be used for both creation and identification of strings of a formal language if and only if those are the strings of the formal language. The formal language theory mostly deals with syntactical aspects of such languages. In Computer Science, formal languages are used as the basis for defining programming languages. In mathematical logic, formal languages are used to represent the axioms. It is believed that all the branches of Mathematics can be reduced to some grammatical operation of formal languages.

Finite-state automata are very constructive mathematical models of computation used to design both computer program and sequential logic circuits. Those can be in one or a finite number of states. The automata are in only one state at a time. They can be changed from one state to another when initiated by a triggering event or condition, this is called a transition. A particular FSA(Finite-state automaton) is defined by a list of its states, and the triggering condition for each transition. The automaton is represented as a directed graph known as state graph which consists of a finite set of vertices known as nodes, together with a set of directed links between pairs of vertices called arcs. Vertices are represented by circles and arcs by arrows. We can also represent an automaton with a state-transition table. As in the graph notation, the state-transition table represents the start state, the accepting states, and what transitions leave each state with which symbols. Finite state automata are divided into two parts. One is deterministic and the other is nondeterministic.
Deterministic FSA (DFSA) - There is one and only one state to which the automaton can have transition from its current state . Its behaviour during recognition is fully determined by the state it is in.
Nondeterministic FSA (NFSA)- A Nondeterministic finite automaton has the power to be in several states at once.
This distinction is relevant in practice, but not in theory, as there is an al-

gorithm namely the power set construction which can transform any NFSA into a more complex DFSA with identical functionality.

From the computational point of view, the use of finite-state machines is mainly motivated by considerations of time and space efficiency. Time efficiency is usually achieved by using deterministic automata. The output of deterministic machines depends, in general linearly, only on the input size and can therefore be considered as optimal from the point of time and space efficiency. Space efficiency is achieved with classical minimization algorithms for deterministic automata.

Parikh mapping or Parikh vector first introduced in [69] by R.J. Parikh (1966) plays a very significant role in the theory of formal languages. With the help of this tool properties of words can be expressed numerically. The Parikh vector is a mapping $\Psi : \Sigma^* \to \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}$ where
$\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ and $\mathbb{Z}$ is the set of natural numbers including 0, such that for a word w in Σ^*, $\Psi(w) = (|w|_{a_1}, |w|_{a_2}, |w|_{b_3} \cdots |w|_{a_n})$ with $|w|_{a_i}$ denoting the number of occurrences of the letter $a_i \in w$. For example, for the word $w = aabbc$ the Parikh vector is (2,2,1). The notion of Parikh matrix introduced in 2001 by Mateescu et al. [61] is an extension of Parikh mapping. A word is a finite or infinite sequence of symbols taken from a finite set called alphabet. A Parikh matrix can be associated with every word over an ordered alphabet and it is a triangular matrix. All the entries of the main diagonal of this matrix is 1 and every entry below the main diagonal has the value 0 but the entries above the main diagonal provide information on the number of certain sub-words in w.

Any word w over the n^{th} order alphabet $\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ has a unique Parikh Matrix. This matrix is given by $\Psi_{M_n}(w)$.

$\Psi_{M_n}(w)=$

$$\begin{pmatrix}
1 & |w|_{a_1} & |w|_{a_1a_2} & \cdots & |w|_{a_1a_2\cdots a_{n-2}} & |w|_{a_1a_2\cdots a_{n-1}} & |w|_{a_1a_2\cdots a_n} \\
0 & 1 & |w|_{a_2} & \cdots & |w|_{a_2\cdots a_{n-2}} & |w|_{a_2\cdots a_{n-1}} & |w|_{a_2\cdots a_n} \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & |w|_{a_{n-1}} & |w|_{a_{n-1}a_n} \\
0 & 0 & 0 & \cdots & 0 & 1 & |w|_{a_n} \\
0 & 0 & 0 & \cdots & 0 & 0 & 1
\end{pmatrix}_{(n+1)\times(n+1)}$$

Where $|w|_{a_i}$ is the number of occurrences of a_i in the word. Here $i \in [1, n]$. An interesting aspect of the Parikh matrix is that it has the classical Parikh vector as the second diagonal above the main diagonal. As such Parikh matrix of a word gives information about Parikh vector of the same.

Natural language is an integral part of our day-to-day lives. Now-a-days some machines have been developed which acts with voice command. So, by using this type of machines, people having little knowledge about the machines can operate them. The aim of Natural language processing is, in a sense, to make such machines . By understanding language processes in procedural terms, we can give computer systems the ability to generate and comprehend natural language. This would make it possible for computers to perform linguistic tasks like translation, process textual data and make it much easier for people to access computer-stored data.

1.1 Motivation:

Formal language is a very important topic for Computer Science, Mathematics and Linguistics. It is used to solve various mathematical problems. The use of formal language in Computer Science is deeply appreciated in making programming language. Some investigation in the field of formal language may benefit all the three subjects- Mathematics, Computer science and Linguistics.

Parikh matrix is a tool to arithmatise formal alphabets and words. Although Parikh Matrix of a word is achieving better results, it still faces some challenging problems, such as it is not injective. Two words may have the same corresponding Parikh matrix, but two words with the same Parikh vector have in many cases different Parikh matrices and thus the Parikh matrix gives more information about a word than the Parikh vector does. To overcome all these shortcomings, Parikh matrix has become a research interest in related fields in recent years. In recent decades scientists have developed many techniques to solve complex problems of words using Parikh matrix. In many branches of Mathematics, in geometry as well as in real analysis, complex analysis, functional analysis, topology etc. it has been found extremely convenient to have a notion of distance which is applicable to the elements of abstract sets. In this present work one kind of distance named as Stepping distance is introduced in Parikh matrix .

A finite state automaton is associated with state graph, whose nodes represent possible system of states, and whose arrows represent possible transitions from state to state. Every finite or infinite word can be represented distinctly by this type of graph. Finite-state automata have been used in many areas of computational linguistics. Their use can be justified by both linguistic and computational arguments. The automaton recognizes a set of strings.

The usefulness of an automaton for defining a language is that it can express an infinite set in a closed form. They often lead to a compact representation of lexical rules, or idioms and words, that appears as natural to linguists. Graphic tools also allow one to visualize and modify automata. This helps in correcting and completing a grammar.

From the computational point of view, the use of finite-state machines is mainly motivated by considerations of time and space efficiency. Time efficiency is usually achieved by using deterministic automata. The output of deterministic machines depends only on the input size. Space efficiency is achieved with classical minimization algorithms for deterministic automata. That is why in theoretical computer science finite automata are profoundly used. Finite-state automata the inevitable part of computational NLP, are used everywhere in the field of part-of-speech tagging, speech recognition, dialogue understanding, text-to-speech, and machine translation. Most of the research works of computational NLP are going on in these fields.

1.2 Statement of the problem:

Being the inevitable part of Computer Science, Mathematics and Linguistics the formal languages have become a subject of much interest among the practitioners of formal languages as well as researchers in this field. While going through the review of literature on this branch of study, it is observed that using various mathematical tools words over formal languages can be studied more robustly. Accordingly arithmatisation of formal languages presents itself as a matter of further in-depth study. It is identified that Parikh matrix, finite state automata, analytical geometry, distance formula can be befittingly utilised to formal languages to make it more arithmatised as well as more attractive. It is also observed that some algorithms can also be formed for finite state automata and for Parikh matrices of words. M-ambiguous words can also be found out from a given Parikh matrix by an algorithm.

Further, another matter of interest came to light that the relation between formal languages and natural languages can be established using

Parikh matrix.

All these matters naturally involve a thoughtful in-depth study on the areas presented in earlier paragraphs. The present study aims at throwing some light on these areas for enriching the knowledge set of formal languages.

1.3 Objectives:

The objectives of the present study are as follows:

1. To investigate on words over formal languages.
2. To make some algorithm to represent formal Languages in terms of finite state automata.
3. To investigate on Parikh matrices of words.
4. To make some algorithms on Parikh matrices of words for solving some existing problems.
5. To enrich the relation between the natural languages and formal language.

1.4 Data and Methodology:

Investigations are done on formal language, finite state automata, Parikh matrix and interrelation of natural languages with formal language. Importance of finite state automata in various fields of interest is analysed. Algorithms developed for finite state automata are studied and a new algorithm is developed. The definitions for various terms on Parikh matrices are generalised. Some theorems on Parikh matrices are investigated and extended to higher ordered alphabets. Some algorithms are generated for solving various challenges of Parikh matrices. For the study of natural and formal languages along the above mentioned lines, various mathematical notions and coding have been applied as and when necessitated.

1.5 Preliminary:

Some definitions used in present study are as under:

Formal Language: It is a set of sequences of symbols that may be guarded by rules that are specific for a particular formal language.

Formal grammar: A formal grammar G consists of [26]:

1. A finite set of nonterminal symbols N, that is disjoint with the strings formed from G.
2. A finite set of terminal symbols T, that is disjoint from N.
3. A finite set P of production rules $(T \cup N)^* N (T \cup N)^* \rightarrow (T \cup N)^*$ i.e. (left-hand side $\rightarrow$ right-hand side) where each side consists of a sequence of these symbols,
4. A start symbol S.

Regular expression: A regular expression, first developed by Kleene in 1956 [55] is a formula in a language that is used for specifying simple classes of strings. A string is any sequence of alphanumeric characters like letters, numbers, spaces, tabs, and punctuation.

Regular language: The class of languages that is definable by regular expressions are regular languages. Given a finite alphabet Σ, the following constants are defined as regular languages:

1. Empty set ϕ: denoting the set ϕ.

2. Empty string ϵ: denoting the set containing only the 'empty' string, which has no characters at all.

3. Literal character $'x'$ in Σ: denoting the set containing only the character $'x'$.

The following operations over given regular languages L and M are defined to produce more regular languages:

1. Concatenation: LM denoting the set $\{\alpha\beta : \alpha \in L \text{ and } \beta \in M\}$. α in set described by language L and β in set described by M. For example $\{abc, d\}\{e, fg\} = \{abce, abcfg, de, dfg\}$.

2. Alternation: $L|M$ denoting the union of sets described by L and M. For example, if L describes $\{ab, cd\}$ and M describes $\{cd, fg, h\}$, language $L|M$ describes $\{ab, cd, fg, h\}$.

3. Kleene star: This is the set of all strings that can be made by concatenating any finite number (including zero) of strings from set described by L. For example, $\{`0', `1'\}^*$ is the set of all finite binary strings (including the empty string),and $\{ab, c\}^* = \{\epsilon, ab, c, abc, abab, cab, cc, ababab, abcab, \cdots\}$.

Turing Machine: A Turing machine is a device that manipulates symbols on a strip of tape according to a table of rules.

Chomsky hierarchy: The Chomsky Hierarchy, as originally defined by Noam Chomsky [25, 27], comprises four types of languages and their associated grammars and machines. The Chomsky hierarchy consists of the following levels:

1. Type-0 grammars or unrestricted grammars : They are the superset of all formal grammars. They generate those languages if and only if the languages can be recognized by a Turing machine. These languages are

also known as the recursively enumerable languages. The example is any computable function.

2. Type-1 grammars or context-sensitive grammars: They generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with A a nonterminal and α, β , and γ strings of terminals and nonterminals. The strings α and β may be empty, but γ must be nonempty. The rule $S \rightarrow \epsilon$ where S is the start symbol and ϵ is the empty string is allowed if S does not appear on the right side of any rule. The corresponding machine is linear bounded automaton. The example is $a^n b^n c^n$.

3. Type-2 grammars or context-free grammars: They generate the context-free languages. These are defined by rules of the form $A \rightarrow \gamma$ with A a nonterminal and γ a string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context-free languages are the theoretical bases for the syntax of most programming languages. The corresponding machine is nondeterministic pushdown automaton. The example is $a^n b^n$.

4. Type-3 grammars or regular grammars: They generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single non terminal. The rule $S \rightarrow \epsilon$ is also allowed here if S does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. The corresponding machine is deterministic or nondeterministic finite-state acceptor. The example is a^*.

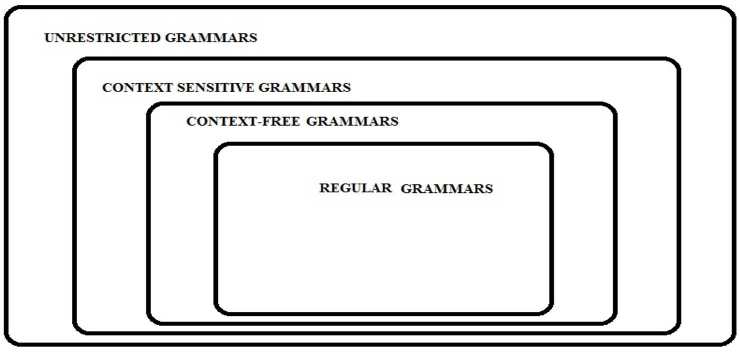

Figure 1.1: Chomsky hierarchy

Finite state automata: A finite-state automaton or a finite-state machine is a mathematical tool used to describe processes involving inputs and outputs. It is a mathematical model used to design computer programs and digital logic circuits.

State transition table: An automaton can be represented with a state-transition table. Like finite state automata, the state-transition table represents the start state, the accepting states, and what transitions leave each state with which symbols [16].

Input word: An automaton reads a finite string of symbols $a_1, a_2, a_3, \ldots, a_n$, where $a_i \in \Sigma$, which is called an input word. The set of all words is denoted by Σ^*.

Run: A run of the automaton on an input word $w = a_1, a_2, a_3, \ldots, a_n \in \Sigma$, is a sequence of states $q_0, q_1, q_2, \ldots, q_n$, where $q_i \in Q$ such that q_0 is the start state and $q_i = \delta(q_{i-1}, a_i)$ for $0 < i \leq n$. In words, at first the automaton is at the start state q_0, and then the automaton read symbols of the input word in sequence. When the automaton reads symbol a_i it jumps to state $q_i = \delta(q_{i-1}, a_i)$ and q_n is said to be the final state of the run.

Deterministic and Nondeterministic finite state automata: Finite state automata are divided into two parts- Deterministic and Nondeterministic finite state automata. In deterministic automata, the transition from one state to another is unique for each possible input. In non-deterministic automata the transition from one state to another is not unique; an input can lead to one, more than one or no transition for a given state.

Throughout this study $\mathbb{Z}$ will denote the set of integers i.e. the set of natural numbers including 0. Some definitions are given below based on the study done in [19, 17, 18, 15]:

Ordered alphabet: An ordered alphabet is a set of symbols
$\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ where the symbols are arranged maintaining a relation of order $(<)$ on it. For example if $a_1 < a_2 < a_3 < \cdots < a_n$ then we use notation: $\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$.

Word: A word is a finite or infinite sequence of symbols taken from a finite set called an alphabet. Let $\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ be the alphabet. The set of all words over Σ is Σ^*. The empty word is denoted by λ.

$|w|_{a_i}$: Let $w \in \Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ be a letter. The number of occurrences of a_i in a word $w \in \Sigma^*$ is denoted by $|w|_{a_i}$.

Sub -word: A word u is a sub- word of a word w, if there exist words $x_1 \cdots x_n$ and $y_0 \cdots y_n$, (some of them possibly empty), such that $u = x_1 \cdots x_n$ and $w = y_0 x_1 y_1 \cdots x_n y_n$. For example if $w = abaabcac$ is a word over the alphabet $\Sigma = \{a, b, c\}$ then $baca$ is a sub-word of w. Two occurrences of a sub-word are considered different if they differed by at least one position of some letter. In the word $w = abaabcac$, the number of occurrences of the word $baca$ as a sub-word of w is $|w|_{baca} = 2$.

prefix and suffix of a word: Prefix: If $w = uy$ for some y, then u is a prefix of w. Suffix: If $w = xv$ for some x, then v is a suffix of w.

Parikh vector: The Parikh vector is a mapping $\Psi : \Sigma^* \to \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}$ where $\Sigma = \{a_1, a_2, a_3, \cdots, a_n\}$ and $\mathbb{Z}$ is the set of integers i.e. natural numbers including 0, such that for a word w in Σ^*, $\Psi(w) = (|w|_{a_1}, |w|_{a_2}, |w|_{b_3} \cdots |w|_{a_n})$

with $|w|_{a_i}$ denoting the number of occurrences of the letter $a_i \in w$. For example, for the word $w = abaabcac$ the Parikh vector is $(4, 2, 2)$.

Triangle matrix: A triangle matrix is a square matrix $m = (m_{i,j})_{1 \le i,j \le n}$ such that

1. $m_{i,j} \in \mathbb{Z}(1 \le i, j \le n)$

2. $m_{i,j} = 0$ for all $1 \le j \le i \le n$

3. $m_{i,i} = 1(1 \le i \le n)$.

Parikh matrix: Let $\Sigma = \{a_1 < a_2 < a_3 < \cdots < a_n\}$ be an ordered alphabet, where $n \ge 1$. The Parikh matrix mapping, denoted by Ψ_{M_n}, is the morphism $\Psi_{M_n} : \Sigma^* \to M_{n+1}$ defined as follows:

if $\Psi_{M_n}(a_q) = (m_{i,j})_{1 \le i,j \le n+1}$ then for each $1 \le i \le (n+1)$, $m_{i,i} = 1$, $m_{q,q+1} = 1$ and all other elements of the matrix $\Psi_{M_n}(a_q)$ are zero.

Parikh matrix of a word: Let $\Sigma = \{a_1 < a_2 < a_3 < \cdots < a_n\}$ be an n^{th} ordered alphabet. The Parikh matrix of $a_1, a_2, a_3, \cdots, a_n$ are as follows:

$$\Psi_{M_n}(a_1) = \begin{pmatrix} 1 & 1 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}_{(n+1)\times(n+1)} ,$$

$$\Psi_{M_n}(a_2) = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}_{(n+1)\times(n+1)} ,$$

$$\ldots, \qquad \Psi_{M_n}(a_n) = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}_{(n+1)\times(n+1)} .$$

Any word w over the n^{th} order alphabet has a unique Parikh Matrix. This matrix is given by $\Psi_{M_n}(w) =$

$$\begin{pmatrix} 1 & |w|_{a_1} & |w|_{a_1a_2} & \cdots & |w|_{a_1a_2\cdots a_{n-2}} & |w|_{a_1a_2\cdots a_{n-1}} & |w|_{a_1a_2\cdots a_n} \\ 0 & 1 & |w|_{a_2} & \cdots & |w|_{a_2\cdots a_{n-2}} & |w|_{a_2\cdots a_{n-1}} & |w|_{a_2\cdots a_n} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & |w|_{a_{n-1}} & |w|_{a_{n-1}a_n} \\ 0 & 0 & 0 & \cdots & 0 & 1 & |w|_{a_n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}_{(n+1)\times(n+1)} .$$

M- ambiguous words: Two words $\alpha, \beta \in \Sigma^*, (\alpha \neq \beta)$ over the same alphabet Σ may have the same Parikh matrix. Then the words are called amiable or M-ambiguous.

The words $baaabaa$ and $ababaaa$ have the same Parikh matrix $\begin{pmatrix} 1 & 5 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$

So these two words are amiable.

M- unambiguous words: A word w is said to be M-unambiguous if there is no word $w^{'}$ for which $\Psi_{M_n}(w) = \Psi_{M_n}(w^{'})$.

Ratio property of words over ternary sequence: Let $\Sigma = \{a < b < c\}$ be a ternary alphabet. Two words $w_1 < w_2$ over $\Sigma = \{a < b < c\}$ are said to satisfy the ratio property, written $w_1 \backsim_r w_2$,

if $\Psi_{M_3}(w_1) = \begin{pmatrix} 1 & p_1 & p_{1,2} & p_{1,3} \\ 0 & 1 & p_2 & p_{2,3} \\ 0 & 0 & 1 & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{4\times 4}$ and $\Psi_{M_3}(w_2) = \begin{pmatrix} 1 & q_1 & q_{1,2} & q_{1,3} \\ 0 & 1 & q_2 & q_{2,3} \\ 0 & 0 & 1 & q_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{4\times 4}$
satisfy the conditions $p_i = s.q_i$, $(i = 1, 2, 3)$, $p_{i,i+1} = s.q_{i,i+1}$, $(i = 1, 2)$ where s is a constant.

Weak-ratio property of words over ternary sequence: Let $\Sigma = \{a < b < c\}$ be a ternary alphabet. Two words $w_1 < w_2$ over $\Sigma = \{a < b < c\}$ are said to satisfy the weak ratio property, written $w_1 \backsim_{wr} w_2$,
if $\Phi_{M_3}(w_1) = \begin{pmatrix} p_1 & p_{1,2} & p_{1,3} \\ p_{2,1} & p_2 & p_{2,3} \\ p_{3,1} & p_{3,2} & p_3 \end{pmatrix}_{3\times 3}$ and $\Phi_{M_3}(w_2) = \begin{pmatrix} q_1 & q_{1,2} & q_{1,3} \\ q_{2,1} & q_2 & q_{2,3} \\ q_{3,1} & q_{3,2} & q_3 \end{pmatrix}_{3\times 3}$
and satisfy the condition $p_i = s.q_i$, $(i = 1, 2, 3)$, where s, $(s > 0)$ is a constant.

*****