STUDY OF FORMAL MULTILANGUAGES AND THEIR ALGEBRA

By

AHMED IBRAHIM ISAH

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA NIGERIA

JULY, 2015

STUDY OF FORMAL MULTILANGUAGES AND THEIR ALGEBRA

By

Ahmed Ibrahim ISAH

N.C.E. (COE KAFANCHAN)

B.Sc., M.Sc. (ABU)

Ph.D./SCI/45288 /2012 - 2013

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF DOCTOR OF PHILOSOPHY IN MATHEMATICS

DEPARTMENT OF MATHEMATICS

FACULTY OF SCIENCE

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA.

July, 2015

DECLARATION

I declare that the work in this dissertation entitled "STUDY OF FORMAL MULTILANGUAGES AND THEIR ALGEBRA" has been performed by me in the Department of Mathematics under the supervision of Professor D. Singh, Dr. A. M. Ibrahim and Professor J. Singh. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other Institution.

ISAH, Ahmed Ibrahim

Name of Student	Signature	Date

CERTIFICATION

This dissertation entitled "STUDY OF FORMAL MULTILANGUAGES AND THEIR ALGEBRA" by Ahmed Ibrahim ISAH meets the regulations governing the award of the degree of Doctor of Philosophy of the Ahmadu Bello University, Zaria and is approved for its contribution to knowledge and literary presentation.

_____ _____

Prof. D. Singh Date

Chairman, Supervisory Committee

_____ _____

Dr. A. M. Ibrahim Date

Member, Supervisory Committee

_____ _____

Prof. J. Singh Date

Member, Supervisory Committee

_____ _____

Prof. M. S. Audu Date

External Examiner

_____ _____

Prof. B. Sani Date

Head, Department of Mathematics

_____ _____

Prof. Z. Hassan Date

Dean, Postgraduate School

DEDICATION

This Dissertation is dedicated to my late parents, Mal. Ibrahim Abdurrahman and Mrs. Khadija Ibrahim.

ACKNOWLEDGEMENTS

I remain grateful and indebted to Allah for giving me the strength to carry out this research to its logical conclusion (Alhamdulillah).

I wish to express my sincere appreciation and warm gratitude to my hard working supervisor Professor Dasharath Singh, for his patience, guidance, motivation, enthusiasm, constant encouragement, timely advice, constructive criticisms and invaluable contribution without which this dissertation would not have been successfully completed. His logical ways of thinking have been a great value to me. May God reward him abundantly.

I wish to express my sincere and warm gratitude to my other supervisors, Dr. A. M. Ibrahim and Professor J. Singh for their assistance towards the completion of this dissertation. I specially wish to thank and recognize the contribution of Dr. A. M. Ibrahim for his advice, suggestions, encouragement and mentoring throughout the programme. May God reward you abundantly.

I wish to express my deep sense of gratitude to the Head of Department Prof. B. Sani for his advices and suggestions. I am also thankful to the Postgraduate Coordinator Dr. A. Yahaya, and the former seminar coordinator Dr. A. Mohammed for their contributions.

I am as well grateful to Prof. S. B. Junaidu (Director of Institute of computing and ICT), Prof. B. K. Jha, Dr. A. O. Ajibade, Dr. H. M. Jibril, Dr. H. G. Dikko, Dr. A. A. Obiniyi, Mal. A. T. Imam, Mal. A. Salisu and Mal U. Shehu for their encouragement.

My special appreciation goes to Mal. A. J. Alkali and Mrs. I. A. Onyeozili for their constant encouragement and academic support throughout the programme.

I am grateful to all other lecturers in the Department of Mathematics, Ahmadu Bello University, Zaria, and my colleagues, who have directly or indirectly con-

tributed during the course of this research.

Also, worthy of mention is Prof. J. N. Singh, Barry University, Miami FL, USA, for his constant support by way of providing a number of relevant research materials during the course of this research.

I wish to extend my appreciation to my relatives and friends who in one way or the other contributed to the success of this work.

I would ever remain grateful to my late parents, Mal. Ibrahim Abdurrahman and Mrs. Khadija Ibrahim, for laying solid foundation on which I have built up my dreams. My gratitude also goes to my entire family, particularly my wives Mrs. Rabi'at Ahmad and Mrs. Hauwa'u Ahmad, and my lovely children, for their constant support, encouragement and patience.

ABSTRACT

In this Dissertation, besides describing some algebraic structures of a language defined over a string, monoids of equivalence classes of a partition and that of all partitions of a language are developed. Further, a new operation is defined over a regular language and its characteristic properties are described. The notion of *sorts* is elaborated and applied to modify and extend the extant theory of multisets. Finally, the concept of a multilanguage is developed, and various operations over languages are extended to multilanguages. Prefix, suffix and substring relations and their algebraic properties are formulated. It is shown that multilanguages of prefixes, suffixes and substrings of a string form a bounded distributive lattice. It is further shown that a prefix (suffix or substring) relation gives rise to a multiset relational structure which always has a unique maximal element and, only (prefix or suffix) relational structure has also a unique minimal element, and the graph of each of these relations is a directed acyclic graph for $n \geq 2$.

Contents

Cover Page i

Fly leaf i

Title page ii

Title Page ii

Declaration iii

Certification iv

Dedication v

Dedication v

Acknowledgement vi

Abstract viii

1 GENERAL INTRODUCTION 1

1.1 Introduction 1

1.2 Statement of the problem 6

1.3 Justification 7

1.4 Aim and Objectives 7

1.5 Methodology 7

1.6 Organization of the Dissertation 8

2 LITERATURE REVIEW 9

3 FORMAL LANGUAGES, THEIR OPERATIONS, AND SOME ALGEBRA 24

3.1 **Basic Definitions** 24

3.1.1 Concatenation of strings 25

3.1.2 Orderings on strings 26
3.2 **Operations on languages** 28
3.2.1 Concatenation of languages 28
3.2.2 Concatenation Closure 29
3.2.3 Mirror image 30
3.2.4 Prefix, Suffix and Substring Closures 32
3.2.5 Quotients 33
3.2.6 Homomorphism 34
3.2.7 Inverse homomorphism 35
3.2.8 Substitution 35
3.2.9 Recursion 36
3.3 **Some Algebraic Structures of Languages** 39
3.3.1 Monoids of equivalence classes of a partition of a language . . 42
3.3.2 Monoids of partitions of a language 43
3.3.3 Some algebraic structures of a language defined over a string . 45

4 GRAMMARS AND AUTOMATA 48

4.1 **Grammars: Chomsky's Hiererchy** 48
4.1.1 Regular (or Type 3) grammars 51
4.1.2 Context-free (or Type 2) grammars 52
4.1.3 Context-Sensitive (or Type 1) grammars 53
4.1.4 Unrestricted (or Type 0) grammar 53
4.2 **Automata** 57
4.2.1 Finite Automata (or Finite-State Accepters) 57
4.2.2 Pushdown automaton 71
4.2.3 Linear bounded automata 75
4.2.4 Turing machines 77

5 FORMAL MULTILANGUAGES AND THEIR ALGEBRA 82

5.1 **Multisets** 82

5.1.1 Multisets of Functional Approach 83
5.1.2 Multisets of Equivalence Relation Approach 85
5.2 **Formal Multilanguages** 93
5.3 **Basic Definitions** 94
5.4 **Operations on Multilanguages** 94
5.4.1 Concatenation of Multilanguages 95
5.4.2 Concatenation Closure of a Multilanguage 95
5.4.3 Union, Intersection, Difference and Complement 96
5.5 **Algebra of Multilanguages** 97
5.5.1 Prefix, Suffix and Substring in a Multilanguage 97
5.5.2 Lattices of Prefixes, Suffixes and Substrings of a String in a Multilanguage 98
5.5.3 Prefix, Suffix and Substring Closures in a Multilanguage . . . 100
5.5.4 Prefix, Suffix and Substring Relations in a Multilanguage . . . 101
5.5.5 Monoids of Prefixes, Suffixes and Substrings of a nonempty String in a Multilanguage 107

6 SUMMARY, CONCLUSION AND RECOMMENDATIONS 109

6.1 Summary 109
6.2 Conclusion 110
6.3 Recommendations 111

References 113

List of Figures

4.2.1 Example of a finite automaton accepting string 59

4.2.2 Transition diagram for an example of a DFA 60

4.2.3 Transition diagram for an automaton M_1 used to construct an automaton for length product 63

4.2.4 Transition diagram for an automaton M_2 used to construct an automaton for length product 64

4.2.5 Transition diagram for an automaton M recognizing the length product of the languages L_1 and L_2 65

4.2.6 Transition diagram for an example of an NFA 67

4.2.7 Transition diagram for an example of an ε−NFA 69

5.5.1 Graph for prefix relation (A) and Graph for substring relation (B) . 105

5.5.2 Hasse diagram for prefix relation (A) and Hasse diagram for the substring relation (B) 106

CHAPTER ONE

GENERAL INTRODUCTION

1.1 Introduction

A language is formal if it is provided with a mathematically rigorous representation of its alphabet *of symbols* and *formation rules specifying which strings of symbols count as well-formed.* In this sense, historically the first *formal language* description is due to Gottlob Frege in 1879 with the introduction of first-order logic.

In 1921, David Hilbert put forward a new proposal for the foundation of classical mathematics which has come to be known as Hilbert's program (Tiplea, 2005). It calls for a formalization of all mathematics in axiomatic form, together with a proof that this axiomatization of mathematics is consistent.

In the 1930s Alan Turing studied an abstract machine (Turing Machine) that had all the capabilities of today's computers (concerning what they could compute). His goal was to describe precisely the boundary between what a computing machine could do and what it could not do. Turing machines help us understand what we can expect from software.

In the late 1930's and the 1940's, Emil Post (Hughes, 2007) devised symbol manipulation systems in the form of rewriting rules (precursors to Chomsky's grammars) which showed their equivalence to Turing machines.

Noam Chomsky in the 1950s, building on earlier work such as David Hilbert, Alan Turing and Emil Post initiated the field of *formal language theory* in order to give a precise characterization of the structure of natural languages. The goal was to define the syntax of languages using simple and precise mathematical rules. Much earlier, the Norwegian mathematician Axel Thue studied sequences of binary symbols and abstracted to interesting mathematical properties, such as not having the same substring three times in a row. His work influenced Emil Post, and others to study the mathematical properties of strings, and collections of strings.

Soon after the advent of modern electronic computers, people realized that all forms of information (whether numbers, names, pictures, or sound waves) can be represented as strings. A collection of strings is a language. This idea became central to computer science. Subsequently, studies were conducted on the properties of languages and language generating systems, such as grammars. Every programming language can be precisely described by a grammar. Moreover, the grammar allows us to write a computer program (called the *syntax analyzer in a compiler*) to determine whether a string of statements is syntactically correct in the programming language. Later it was found that the syntax of programming languages can be described using one of the Chomsky's grammatical models called *context-free grammars*. Chomsky classified formal languages into what is known as the *Chomsky hierarchy* based on the expressive power of their generative grammar as well as the complexity of their recognizing automaton (Jäger and Rogers, 2012). In its classical formulation, the Chomsky hierarchy has four levels of increasing complexity: regular, context free, context sensitive, and computably enumerable languages. Context-free grammars and regular grammars provide a good compromise between expressivity and ease of parsing, and are widely used in practical applications (Grune and Jacobs, 1990).

Subsequent work in formal linguistics shows that, this four folds distinction is too coarse grain to pin down the level of complexity of natural languages along this domain. Therefore, several refinements have been proposed, such as, levels that extend the class of context-free languages called *mildly context sensitive languages* and one that further delimit the regular languages called *the sub-regular hierarchy* ((Joshi et al., 1990) and (Rogers and Pullum, 2007)).

Chomsky suggested a series of simplifications and abstractions to the empirical domain of natural language (Chomsky, 1956). The immense success of this framework, which lies in influencing not just linguistics to this day, but also theoretical computer science, and more recently, molecular biology, suggests that these abstractions were well chosen, preserving essential aspects of the structure of natural languages. The field provides a measuring stick for linguistic theories that sets a minimal limit of descriptive adequacy.

Formal languages have their origin in symbolical notation, axiomatics in mathematics, and especially in combinatory and symbolic logic. Subsequently, it was found important in various codes needed in data encryption, transmission, and error-correction; in particular, in construction of various mathematical models of automaton and computation.

A formal language is a set of strings over a fixed alphabet (Kari, 2013). A language is finite if it contains a finite number of strings. A word (or string) is a finite sequence of items (called *symbols or letters*) chosen from a specified finite set called the *alphabet*. For any alphabet Σ, a language over Σ is a set of strings over Σ. The members of a language are also called the *words* of the language. That is, a formal

language is a set of strings of symbols that may be constrained by rules that are specific for a particular formal language. The field of formal language theory studies primarily the purely syntactical aspects of such languages, that is, their internal structural patterns.

In computer science and mathematics, which do not usually deal with natural languages, the adjective *formal* is often omitted as redundant. Formal language theory rarely concerns itself with particular languages (except as examples), but is mainly concerned with the study of various types of formations to describe languages. For instance, a language can be given as:

1. Those strings generated by some formal grammar;
2. Those strings described or matched by a particular regular expression;
3. Those strings accepted by some automaton, such as Turing machine or finite state automaton;
4. Those strings for which some decision procedure (an algorithm that asks a sequence of related YES/NO questions) produces the answer YES.

Typically questions asked about such formalisms include:

1. What is their expressive power? That is, can formalism X describe every language that formalism Y can describe? Can it describe other languages?
2. What is their recognizability? That is, how difficult is it to decide whether a given string belongs to a language described by formalism X?
3. What is their comparability? That is, how difficult is it to decide whether two languages; one described in formalism X and one in formalism Y, or in X again, are actually the same language?

This is why a formal language theory has featured out to be a major application area of computability theory and complexity theory.

In linguistics, formal languages are used for the scientific study of human language. In computer science, formal languages are used among others as the basis for defining programming languages and therefore, in the development of compilers since compilers are computer programs formulated to transform source code written in a programming language into another computer language. It is also used for describing formalized versions of subsets of natural languages in which the strings of the language represent concepts that are associated with particular meanings or semantics. In computational complexity theory, decision problems are typically defined as the sets of the formal languages, and complexity classes are defined as the sets of the formal languages that can be parsed by machines with limited computational power. In logic and the foundations of mathematics, formal languages are used to represent the syntax of axiomatic systems, and mathematical formalism is the philosophy that all of mathematics can be reduced to the syntactic manipulation of formal languages in this way.

The theory of formal languages provides the theoretical underpinning for the study of programming languages, and is also important in such areas as data compression and computer networks. Essentially, it blends algebraic techniques with abstract models of computing devices. The mathematical theory of formal languages grew with the need of formalizing and describing the process linked with the use of computers and communication devices. Recently, formal languages have been as well applied in soft sciences, like biology and economics.

In addition, we invariably encounter with systems which contain objects with repeated elements or attributes (for example, groups of people, systems of elementary particles and having two or more elements with the same property). We need a (formal) mathematical structure to model this kind of data. In the recent years such mathematical structures have been developed which are in general called *multiset-based structures.*

A Multiset (mset, for short) is an unordered collection of *objects* in which, unlike a standard (Cantorian) set, duplicates or multiples of objects are admitted (Singh et al., 2007). In other words, an mset is a collection in which objects may appear more than once and each individual occurrence of an object is called its *element.* All duplicates of an object in an mset are indistinguishable. The objects of an mset are the distinguishable or distinct elements of the mset. The distinction made between the terms *object* and *element* does enrich the multiset language. The set of all objects of an mset is called its *root set*, denoted A^*. Accordingly, sets are merely special instances of multisets.

Besides in mathematics, multisets are being extensively used in theoretical computer science, biosystems (such as membrane computing and DNA computing), economics, social sciences and so on (see (Singh et al., 2007) for example).

1.2 Statement of the problem

Despite a number of applications of formal language theory and that of multisets illustrated above, multiset model of formal languages and their algebra have not been adequately developed as yet.

1.3 Justification

Studying formal languages via multisets would further comprehend various applications in mathematics, computer science and so on.

1.4 Aim and Objectives

The aim of this Dissertation is to develop a multiset model of formal languages and formulate their algebra.
The objectives are to:

1. Comprehensively and critically study the development of formal language theory.

2. Comprehensively study the fundamentals of formal languages, grammars, and automata, and develop some new concepts.

3. Formulate some algebraic structures of formal languages.

4. Study the theory of multisets and develop some new results.

5. Develop a multiset model of formal languages.

1.5 Methodology

In order to achieve the above outlined objectives, an up-to-date critical review of the existing literature was conducted, some new concepts were introduced, and related results were obtained.

1.6 Organization of the Dissertation

As chapter one covers the introductory part of the Dissertation, chapter two consists of a critical review of existing literature. In chapter three, syntax of formal languages, their operations and some algebra were presented. Chapter four was devoted to grammars and automata. In chapter five multilanguages were formulated. Finally, chapter six consists of the summary of the main results obtained and some directions for future research.

CHAPTER TWO

LITERATURE REVIEW

Introduction

Noam Chomsky's attempts in 1956 to formulate a general theory of the syntax of natural languages made a great impact on the development of formal language theory. A melting pot of ideas then developed with a convergence of thought between linguists, mathematicians, logicians, and newly born computer scientists.

Chomsky (1956) classified grammars into the following four classes by imposing restrictions on the production rules, known as the *Chomsky hierarchy*: unrestricted grammars (Type 0), context-sensitive grammars (Type 1), context-free grammars (Type 2), and regular grammars (Type 3).The *Chomsky hierarchy* turned out to be extremely useful basis for the understanding and use of a computer science model which enabled programmer to accomplish meaningful linguistic goals systematically.

On the class of natural languages on the hierarchy, Chomsky (1957) pointed out that as English has centre embedding constructions: involving two dependent elements a and b that are not adjacent, and that may contain another instance of the same construction between the two parts, such as *neither-nor* constructions, it is not regular.

Harrow (1978) presented a series of examples of languages, and provides relatively simple but rigorous proofs of their proper containments in various classes. Among the examples discussed (and the methods of proof outlined) are: a language which is context-free but not regular (myhill-nerode theorem), context-sensitive but not

context-free (uvwxy theorem or pumping lemma), a language but not type 0 or alternatively a function which is total but not recursive.

Huybregts (1984) and Shieber (1985) argued that Swiss-German is not context-free as the dependencies between verbs and their objects were unbounded in length. In fact, it has crossing dependencies between objects and verbs, and the number of these interlocked dependencies is potentially unbounded. Since context-free grammars can handle an unbounded number of interlocked dependencies only if they are nested, Swiss-German cannot be context-free. On the same lines, Postal and Langendoen (1984) observed that English was not context-free. In view of the fact that the aforesaid constructions were seen to extend to almost all other natural languages, it became imperative to design grammars formalisms that could be suitable for doing linguistics and also be in the vicinity of context-free grammars along with preserving its computational tractability.

Joshi (1985) described a list of properties that an extension of the context-free languages should have if it is to be of practical use for linguistics: It should contain all context-free languages, it can describe a limited number of types of cross-serial dependencies, its membership problem has polynomial complexity, and all languages in it have constant growth property. He called classes of languages having these properties *mildly context-sensitive* as they extend the context-free languages and slightly absorbing some context-sensitive languages.

Joshi et al. (1990) described the relationship between several different grammar formalisms belonging to the class of *Mildly context-sensitive grammars* namely, Tree Adjoining Grammars (TAGs), Combinatory Categorical Grammars (CCGs), Linear Indexed Grammars (LIGs), and Head Grammars (HGs) and further showed their equivalent as they described the same class of languages.

Knuth (1992) emphasized the need to allow a strings to repeat in a language, and presented the concept of Multilanguage which is a language but its objects may repeat. Thus, if we regard a language as a set of strings, a Multilanguage is a multiset of strings. However, various language operations were not extended to Multilanguage.

Cai and Paige (1995) partly disagreed that *hashing* is essential to solve a large class of language processing problems efficiently e.g., *symbol table formation and maintenance*, *grammar manipulation*, *basic block optimization*, and *global optimization.* They provided an efficient alternative compiler methodology without *hashing* or *sorting.* They presented a new theoretically efficient hash-free solution to an assortment of programming language problems based in large part on an efficient multiset discrimination methods for data types built up from basic data types implemented using directories and from sequence constructors. The generic nature of these methods and their wide ranging successful applications to language processing problems demonstrate a basic algorithm design tool for solving problems in various other areas of computer science. Moreover, the replacement of solutions based on hashing with solutions based on multiset discrimination illustrates a fundamental principle of algorithm improvement. Nevertheless, though the hash-free solution presented have better theoretical performance than their predecessors, further investigation comparing the hash-free alternatives with their conventional hash-based counterparts would reveal whether they have any computational advantage.

Ilie (1997) observed that the generative process in a context grammar is based on most important dual linguistic operations both in natural and artificial languages: *insertion of a word in a given context (pair of words)* and *adding a context to a given*

word. Any derivation in a context grammar is a finite sequence of such operations, starting from a finite set of words, simple enough to be considered as *primitive well formed words (axioms)*. He presented a brief survey of some of the most important recent results in the area of contextual languages, specially appropriate from natural language point of view, in which the concept of context-sensitivity plays a central role. He pointed out that the linguists seem to finally agree that the natural languages are not context-free; several languages (such as Bambara dialect, Swiss, German, etc.) contain convincing non-context-free features and this is probably true for most natural languages. Thus, in order to obtain formal grammars in order to model (specific constructions in) natural languages, we need to look for classes of grammars up-able of generating non-context-free languages. The idea of *mildly context-sensitive* families of languages that keep the generative power under control was presented.

Bilska et al. (1997) realized that formal language and automata theory course has lagged behind in integrating hands-on materials into lectures and assignments even though students learned algorithms better if they had hands-on experience. As such, they developed a set of tools written in Java for experimenting with concepts in formal languages and automata theory. These include JFLAP (Java Formal Languages and Automata Package), P*áté* (Parsing and Transforming Engine) and Pumping Lemma. JFLAP which is an instructional software tool that covers topics in FLAT (Formal Language and Automata Theory) in a visual and interactive manner, allows one to construct automata, push down automata and Turing machines, and run traces on input strings. P*áté* contains two parts for experimenting with grammars; a *brute force parser* that shows the derivation and parse tree for an input string, and a *transformer* for converting a context-free grammar to CNF (Chomsky Normal Form). Pump Lemma allows one to experiment with the regular pumping lemma to determine whether the language is regular or not. The tools

are of practical nature, during lecture they can be demonstrated using a computer, and with input from students, problems can be solved. Students can as well use the tools to reproduce examples treated in the class, and can further create their examples. Despite aiding the learning of Formal languages and automata theory course, the tools are time consuming and have only a smaller coverage of topics in Formal Language and Automata theory course.

Lloyd (1998) observed that, despite multisets are recognized as being important in specification but are not normally provided as a fully-fledged data type in programming language design. In order to reduce this mismatch, he proposed an interesting way of introducing multisets into declarative programming languages. Note that, declarative programming is a programming paradigm that expresses the logic of computation without describing its control flow. He presented the idea in the context of Escher programming language, which is a general purpose, higher order declarative programming language that integrates the best feature of both functional and logic programming languages. He started by distinguishing multisets as intensional and extensional, and expresses that extensional multisets can be implemented with standard programming language data types, and that in higher order functional programming language an intensional multiset can be defined along with its range of operations. Suitable operations on multisets (such as multiset cardinality, multiset membership, msubset, multiset equality, etc.) are defined and implemented in a declarative programming language. He further gave a variety of programming examples to illustrate the utility of his approach. This approach to multisets is compared with their treatment in the $\mathbb{Z}$ specification language and other approaches in programming languages. The advantage of this approach is its flexibility of handling multisets, which attract more complex implementation.

Hung and Rodger (2000), in consideration of the significance of interaction and visualization in the learning of automata theory, considered and improved the work of Bilska *et al.* (1997) on JFLAP and P*áté* by integrating into JFLAP regular expressions and the conversion of NFA (Non-deterministic finite automata) to regular expressions. They also added steps to the conversion of automata to grammars, and make an expanded help section in html format. With these developments in JFLAP one can convert any representation of a regular language into another representation, and with help one can either create a new representation or watch an animation step through its creation. In P*áté*, they included the visualization and an animation of a parse tree for unrestricted grammars, improved interaction in both the parsing and grammar transformation, and an expanded help section in html format is also produced. Moreover, they further made the file format for both JFLAP and P*áté* the same, making it easier to switch between the two.

The addition of regular expressions to JFLAP allows one to take one representation of a regular language and convert it to any of the other representations and even back again. As for the development in P*áté*, one can continue the visualization of parse trees as they learn different types of grammars. The increased interaction in both tools allows one focus on the steps of algorithms which makes it faster than the previous version. However, several topics in Formal language and automata theory course were not covered by this version of JFLAP which negatively affects the understanding of the course.

Nederpelt (2002) observed that mathematicians usually expressed mathematical ideas in writing through books, papers etc., based on natural language mixed with mathematical symbols and formulas. He called such method of communicating mathematical ideas as *Common Mathematical Language (CML)* which on the basis of formal language theory has no prescribed syntax. He thinks it is useful to have a

formal language for communicating mathematical knowledge. He proposed a formal language for expressing mathematics called *Weak Type Theory (WTT)* and claim that it is close to the usual way in which mathematicians express themselves in writing. The theory has a precise syntax consisting of a grammar and a derivation system based on linguistic categories such as nouns and adjectives. Constants are treated as *first class citizens* as are binders like Σ, $\cup$ or $\forall$. Notions such as assumption, declaration of a fresh variable, definition and statement (covering notions like theorem and step in proof) are incorporated in the theory. The theory which is relatively easy to use, provide a uniformity of expression and has a high degree of reliability in the sense that it gives a true image of what the mathematician intends to say. However, the theory is still very weak as regards to correctness, as its rules only concern linguistical correctness. All information concerning the real essence of correctness, like its logical mathematical aspects of truth, trustworthiness and applicability, are disregarded. Hence, the familiar Common mathematical language found in books, papers is usable and good enough for continuous utilization. The proposed Weak Type Theory (WTT) could be adopted by mathematician as a second language.

Jantzen (2003) noted that the interest in multisets and subsets of commutative monoids have increased in the recent years. In standard formal language theory, already some results have been obtained for commutative strings and languages. In comparison, the arbitrary multiset grammars are in some sense equivalent to variants of *vector replacement* systems. He showed that the family sem = mREG = mCF of regular multiset languages is closed under applications of finite and iterated multiset-union, finite and iterated multiset-intersection, and multiset-subtraction. It is further shown that for the class sem$\mathbb{Z}^k$ of semilinear subsets of $\mathbb{Z}^k$, or that of $\mathbb{N}^k$, sem:= $\bigcup_{k\geq 0} sem(\mathbb{N}^k)$ verifying that the component-wise maximum, minimum or non-negative subtraction of pairs of elements from two semilinear sets is again

semilinear, and that the iterated application can be replaced by a fixed finite application of multiset-intersection, respectively multiset-subtraction. Finally, some open questions about closure properties are solved, and the result that the family mMON is not closed with respect to multiset-intersection is verified.

Chesnevar et al. (2003) considered the importance of educational software for learning formal language and automata theory (FLAT) course which are core to the computer science curricula. They distinguished two categories of FLAT educational software available via the internet and reviewed various tools under each. The two categories presented were multi-purpose and single-purpose software tools for FLAT. Based on their practical experience, they analyzed and discussed the trade-offs between different FLAT software tools under these categories. They further suggested the use of multi-purpose software tools such as *Minerva, DEM (Deus ex machine)* and *JLAF* as both provide a unified view of different kinds of automata.

Nederhof et al. (2003) presented a formalism called *partially ordered multiset context-free grammars (poms-CFG)*, along with an Earley-style parsing algorithm. It shares with regular right-part grammars, an interleaving operator that can be thought of as generalization of context-free grammars with partially ordered right-hand sides. This serve as infrastructure for obtaining relatively stricter complexity bounds for more expressive context-free formalism intended to express free or multiple word-orders, such as ID/LP grammars. The class of Poms-CFG allows compact representation of languages with free word order. It is shown that by recasting ID/LP grammars with Poms-CFG one can generate refined bounds on ID/LP parsing complexity and this lead to polynomial complexity in practice. The effective width of attested ID/LP grammars is small, yielding effectively polynomial time complexity for ID/LP grammar parsing.

They further reduced ID/LP grammars to poms-grammars thereby getting finer-grained bounds on the parsing complexity of ID/LP grammars.

Perrin (2003) in his paper; *Automata and formal languages* explored the theory of automata and formal languages; starting from its inception to the development of its various components. The Chomsky hierarchy was partly presented with emphasis on context-free and recursively enumerable languages. He emphasized that a language is context-free if it can be accepted by some pushdown automata, and recursively enumerable if recognized by some Turing machine. The links of the theory with other mathematical fields including semigroups, groups, logic, etc., were highlighted. He finally presented the applications of formal language and automata to software design with the conception of compilers, text processing including text compression, natural language processing, and genome analysis.

Chakraborty (2003) outlined steps for constructing a corresponding finite automaton for a given regular expression via NFA-ε and NFA. The procedure adopted is based on the fact that given a regular expression, in most cases, it can be broken into natural subparts and an automaton is constructed for each subpart, and then joined together to obtain a finite automaton corresponding to the given regular expression. It is further shown that a language is regular if and only if it is accepted by a finite automaton.

Meduna and Vitex (2004) presented a fragment of the formal language theory, which deals with strings, languages and operations on them. A number of language operations are introduced together with their properties which they divide into two groups: *insertion* and *deletion* operations. New algorithms that construct finite

automata accepting languages resulting from some of these operations are described and it is demonstrated that the family of regular languages is closed under these operations.

Cavalcante et al. (2004) observed that over the years, several software tools such as JFLAP were developed in order to change the traditional FLAT (Formal language and automata theory) course from a theoretical presentation to a more visual and interactive approach, which proved insufficient as the developed JFLAP does not cover most of the topics in FLAT. Considering this, they presented an improved version of JFLAP of wider coverage, and named it JFLAP 4.0. This help in addressing the problem of coverage with the existing JFLAP of (Hung and Rodger, 2000). In JFLAP 4.0 existing topics were improved and new topics such as Hierarchy of formal languages, L-systems, and LL and SLR parsing were integrated. With JFLAP 4.0 Multi-tape Turing machine for up to five tapes can be created and with three tapes, one can create a universal Turing machine. Also two finite automatons can be compared for equivalence. This allows students to check their design against another solution that may appear to be different. Here several automata can be combined in the same window, and this allows one to build a new automaton faster by using parts from other automaton. Moreover, with JFLAP 4.0 one machine could be loaded and run on several inputs. The next machines loaded would use the same inputs, so that the grader would not have to reenter them. Nevertheless, grading could still take a long time for a large course as each machine has to be loaded and run. With these new developments, JFLAP 4.0 can be used along with automata theory textbooks to create hands-on FLAT course. It can also be used with almost all topics in Formal language and automata theory courses. The tool is more visual and easier to understand by users than the previous version. However, JFLAP 4.0 is a bit slower as only one input could be tested at a time on grammar, with the result being either a derivation or a parse tree.

Cavaliere et al. (2007) introduced a general model of random context multiset grammars as well as the concept of multiset random context checkers and transducers. They showed how recursively enumerable sets of finite multisets can be generated using these models of computing. They further presented antiport P system and showed how several results for multiset grammar could easily be extended to antiport P system.

Rodger et al. (2007) considered the significance of introducing instructional software such as JFLAP to the teaching of FLAT (Formal language and automata theory) course, which allows students to experiments with concepts that would be difficult and tedious to do on paper, and receive immediate feedback on problem solving. For the purpose of increasing interaction in FLAT course, they presented an improved version of JFLAP which they named JFLAP 6.0. The new topics integrated in the improved JFLAP 6.0 include *transducers* and the *pumping lemma*. (A transducer is a finite automaton that generates output.) JFLAP 6.0 provides experimentation with two types of transducers; *mealy machines* for generating output based on transitions and *moore machines* for generating output based on states. Adding these to JFLAP expands its applications as mealy machines can model problems in many disciplines. In addition pumping lemma games for both the regular pumping lemma and the context-free pumping lemma were also integrated. Both the topics allow students to experiment with them and receive immediate feedback, which certainly enhance the understanding of Formal language and automata theory courses. Moreover, in JFLAP 6.0 Multi-run input has been added to grammars, this is done in order to overcome the shortcoming of the previous version where only one input could be tested at a time on a grammar. With multi-run input one can run several inputs at the same time, and then select one at a time from the results to view

the derivation or parse tree. Batch grading support has also been added in order to allow one to load and produce output for multiple machines at once thereby creating an interface for grading a large number of JFLAP files. This is done to overcome the shortcoming of time consuming of grading in respect of a large course experienced in JFLAP 4.0.

Worthington (2009) treated automata and formal languages from a bialgebraic perspective: one that includes both algebraic and coalgebraic structures, with appropriate interactions between the two. That is, taking automata as certain representation objects of algebras and formal languages as elements of dual algebras of coalgebras. These enable them to show that there is an adjunction between the category of *algebraic* automata and the category of deterministic automata.

Wang (2009) observed that linguist and formal language theories play significant roles in the theory of computing. This intimates him to analyzed and presented ways in which linguistics improve the understanding of programming languages and their work products i.e., software, and how formal language theory enhance the study of natural languages. He further developed a universal language processing model and the deductive grammar of English toward the formalization of Chomsky's universal grammar in linguistics. He presented a wide range of applications of the deductive grammar of English in language acquisition, comprehension, generation, and processing in cognitive informatics, computational intelligence, and cognitive computing. Comparative analyses of natural and programming languages were presented, indicating that the syntax of programming languages is far more complicated than those of natural languages. But the semantic of programming languages is much simpler than that of natural languages, and suggested that for achieving such simple and precise semantics in programming languages, a set of very

complex and rigorous syntax and semantical rules has to be adopted.

Rodger et al. (2009) noted that an instructional software JFLAP was developed and modified several times over the years in order to provide a more visual, interactive and applied experience with theoretical concepts in the teaching/learning of formal languages and automata theory course. In their efforts to further improve the tool, they investigated its effectiveness in enhancing the learning process, and what additional value it adds to the formal language and automata theory course. They executed this by making a two-year study with fourteen universities using the tool in formal language and automata theory courses. Based on the feedback from students, faculty and other users, they presented the result of their study and make some changes and additions to JFLAP to improve its effectiveness for better understanding of formal language and automata theory course. The changes and additions made include new algorithms for experimenting with concepts such as the *user-control parser* which enable students try to derive a string from a grammar on their own. Others are the *CYK (cocke-younger-kasami) parser* which is substantially faster than the existing bruce-force parser. The pumping lemma was expanded to create more interaction, where either the user can go first or JFLAP can go first, contrary to the previous version where the user always starts the game.

Baulch (2010) observed that automata theory is intimately related to the theories of directed and topological graphs, semigroup theory and recursion theory. Considering the fruitful calculus of relators in group theory, he presented a theory of strongly connected automata with a similar flavor, where he investigated sufficient conditions for a *cycle structure* to completely determine the automaton. He further developed algorithms for computing with such invariants. Language-theoretic analogues of these automata-theoretic conditions are also developed.

Sassaman et al. (2011) noticed that computer security is often regarded as a never-ending arms race between attackers seeking to exploit weak points in software and defenders scrambling to defend regions of an ever shifting battlefield. As such they presented an approach to improve the security of complex, composed systems based on formal language theory, and showed how this approach leads to advances in input validation, security modeling, attack surface reduction, and ultimately software design and programming methodology. They also introduced a formalization of an exploit development technique and the *parse tree differential attack*. Their work actually provides a foundation for verifying critical implementation components with considerably less burden to developers than is offered by the current state of the art.

Jäger and Rogers (2012) noted that intensive research undertaken in the field of formal language theory besides providing a measuring stick for linguistic theories that steps a minimal limit of descriptive adequacy, have immensely contributed to theoretical computer science and more recently molecular biology. They observed that the mildly context-sensitive and sub-regular languages were proposed with the observation that the four-fold distinction is too coarse grain to pin down the level of complexity of natural languages. They further recapitulated the characteristic properties of four classical levels of the Chomsky hierarchy including the two proposed languages and relevancy or otherwise to the analysis of natural languages.

Ahmadi et al. (2014) in consideration of the applications of algebraic structures in many branches of sciences such as mathematics, computer science and finite machines, using the idea of free monoid in formal language theory, presented a class of algebraic structures consisting of monoids, semigroups and cyclic groups for every integer n.

In view of the diverse applications of both formal language theory and multisets, the study of algebra of languages, and multiset in formal language theory is being undertaken. This Dissertation aims to study multiset model of a formal language and develop some algebra of languages.

CHAPTER THREE

FORMAL LANGUAGES, THEIR OPERATIONS, AND SOME ALGEBRA

Formal languages (or simply languages) were developed in the 1950s by Noam Chomsky in an endeavor to systematically describe natural languages with an objective to make passage easier from one language to another using computers. It has as well found a number of applications in computer science such as compiler construction for high-level programming languages, etc.

In this chapter, formal languages and their operations are described and exemplified in sections 3.1 and 3.2. While in 3.3, some algebra of formal languages are formulated.

3.1 Basic Definitions

Most of the definitions given are standard and mimicked from various sources, particularly from (Hopcroft et al., 2001), (Ruohonen, 2009), (Gallier, 2010) and (Kari, 2013).

Definition 3.1.1 Alphabet

An alphabet, denoted Σ (or more specifically Σ_T, particularly in the context of defining Grammars) is a nonempty set of distinguishable symbols, called *terminal* symbols or simply *terminals*. An element of an alphabet is called a *letter* or a *character*. An alphabet needs not be finite. However, for our purposes, we shall assume it to be finite.

Definition 3.1.2 Strings

A string is a finite ordered set of symbols from the alphabet. It is also called a

(finite) *sequence,* a *word*, or a *sentence* depending on context. A string u consisting of k symbols $(k > 0)$ is said to be of length k, denoted $|u| = k$. In the event, an empty string (i.e., a string of length 0), denoted $\varepsilon, \lambda, \Lambda$ or 1, is admitted, the system can have strings of length $k \geq 0$.

Strings can also be defined as functions (Gallier, 2010). For any integer $n \geq 1$, let $[n] = \{1, 2, \ldots, n\}$.

A string u of length n over a fixed Σ is a function

$$u : [n] \longrightarrow \Sigma.$$

In case $n = 0$, the special string $u : [0] \longrightarrow \Sigma$ having length 0 (having no symbol), called the *empty* or *null string* is obtained.

Given a string $u : [n] \longrightarrow \Sigma$ of length $n \geq 1$, let $u(i)$ denote the i^{th} letter in the string u. Then a string u of length n can also be represented as

$$u = u_1 \, u_2 \ldots u_n, u_i \in \Sigma, \quad i = \overline{1, n}.$$

For example, let $\Sigma = \{a, b\}$ and $u : [5] \longrightarrow \Sigma$ be defined such that $u(1) = a$, $u(2) = b$, $u(3) = b$, $u(4) = a$ and $u(5) = a$, then

$$u = abbaa.$$

3.1.1 Concatenation of strings

Let $u : [m] \longrightarrow \Sigma$ and $v : [n] \longrightarrow \Sigma$ be two strings over Σ, then concatenation (or simply catenation) of u and v, denoted $u.v$ or uv, is the string $uv : [m + n] \longrightarrow \Sigma$,

defined as

$$uv(i) = \begin{cases} u(i) & \text{if } 1 \le i \le m \\ v(i-m) & \text{if } m+1 \le i \le m+n \end{cases}$$

It is immediate to see that the following holds:

$u\varepsilon = \varepsilon u = u$ and $u(vw) = (uv)w$.

Note that catenation, in general, is not commutative i.e., $uv \neq vu$.

For example, let $u = aab$, $v = bb$ over $\Sigma = \{a, b\}$, then $uv \neq vu$.

Given a string u over Σ and $n \ge 0$, u^n is defined as follows:

$$u^n = \begin{cases} \varepsilon & \text{if } n = 0, \\ u^{n-1}u & \text{if } n \ge 1. \end{cases}$$

Clearly, $u^1 = u$, $\varepsilon^n = \varepsilon$, $\varepsilon u = u\varepsilon$ and $u^n u = uu^n$, for all $n \ge 0$.

3.1.2 Orderings on strings

Strings are ordered by applying lexicographic ordering. The lexicographic ordering, abbreviated $\le$ or $\le_{lex}$, is defined as follows:

Let R be a linear ordering relation on a set X and let $T = X \cup X^2 \cup \cdots \cup X^n$ i.e., the set consisting of strings on X of length less than or equal to a fixed n. A string of length m can be represented as an ordered $m-tuple$. Let $(a_1, a_2, \ldots, a_p)$ and $(b_1, b_2, \ldots, b_q)$, $p \le q$, be two strings. Now, $(a_1, a_2, \ldots, a_p) \le_{lex} (b_1, b_2, \ldots, b_q)$ if any one of the following holds:

1. $(a_1, a_2, \ldots, a_p) = (b_1, b_2, \ldots, b_q)$

2. $a_1 \neq b_1$ and $a_1 R b_1$ in X

3. $a_i = b_i, i = \overline{1,k}$ $(k < p)$, and $a_{k+1} \neq b_{k+1}$ and $a_{k+1} R b_{k+1}$ in X

4. If none of the above, then $(b_1, b_2, \ldots, b_q) \leq_{lex} (a_1, a_2, \ldots, a_p)$, for example, $t_0 w \leq_{lex} t_0$.

Let $a_1 a_2 \ldots a_m$ and $b_1 b_2 \ldots b_n$ be two different strings on a partially ordered set Σ. Let $k = min(m, n)$. Then, $a_1 a_2 \ldots a_m \leq_{lex} b_1 b_2 \ldots b_n$ if and only if $(a_1, a_2, \ldots, a_k) < (b_1, b_2, \ldots, b_k)$, or $(a_1, a_2, \ldots, a_k) = (b_1, b_2, \ldots, b_k)$ and $m < n$.

This is a partial ordering on Σ^*. In other words, in order to determine the ordering of two different strings, the longer string is truncated to the length of the shorter string and then compared lexicographically as above. Note that a partial ordering $\leq$ is obtained by adding equality to the strict ordering $<$. However, if Σ is linearly ordered, then the lexicographic ordering on Σ^* is a linear ordering.

Example 3.1

Let $\Sigma = \{b_1, \ldots, b_k\}$ such that $b_1 < b_2 < \cdots < b_k$, $u, w \in \Sigma^*$, then the lexicographic ordering, denoted $\preccurlyeq$, is defined as:

$$u \preccurlyeq w \begin{cases} \text{if } w = uq \text{ for some } q \in \Sigma^*, \text{ or} \\ \text{if } u = pb_i q, w = pb_j r, \text{ and } b_i < b_j \text{ for some } p, q, r \in \Sigma^*. \end{cases}$$

In fact, lexicographic ordering of Σ^* is monotonic and linear (see (Tremblay and Manohar, 1997), (Rosen, 1999) and (Gallier, 2010) for details).

The lexicographic ordering of the strings $abbb$, a, ab, b, bb, aa, aaa over $\Sigma = \{a, b\}$, $a < b$, is a, b, aa, ab, bb, $aaa, abbb$ and also, lexicographic ordering of Σ^* itself is $[\, \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots]$, etc.

3.2 Operations on languages

Let Σ^* denote the set of all (finite) strings over Σ and $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ the set of all nonempty strings. It may be observed that catenation is a binary operation on Σ^* which is associative and has ε as an identity. Thus, $(\Sigma^*, ., \varepsilon)$ or $(\Sigma^*, .)$ is a free monoid, and $(\Sigma^+, .)$ is a free semigroup generated by Σ.

For any alphabet Σ, let Σ^k denote the set of all strings of length k over Σ. For example, let $\Sigma = \{a, b\}$, then $\Sigma^0 = \{\varepsilon\}$, $\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$ and so on. Note that $\Sigma = \{a, b\}$ is an alphabet and $\Sigma^1 = \{a, b\}$ is the set of all strings of length 1. Moreover, $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ i.e., $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$.

Definition 3.2.1 Formal languages (or simply languages)
A language consists of a finite or infinite set of strings or sentences that are defined by finite means only. A language can also be considered as a subset of the free monoid on a fixed alphabet i.e., the set of all strings with catenation as the operation. However, as the language consisting of a free monoid is too large, it is not found suitable for practical applications. In the following, a language L will be considered as a subset of Σ^* i.e., $L \subseteq \Sigma^*$.
It may be noted that ε is a string of length 0, $\varnothing$ is a language having no string, and $\{\varepsilon\}$ is a language having ε as the single member.

3.2.1 Concatenation of languages

The *catenation* of the languages L_1 and L_2 is $L_1L_2 = \{ w_1w_2 | \ w_1 \in L_1$ and $w_2 \in L_2\}$. Catenation of languages is associative because catenation of strings is associative.

The n^{th} catenation of the language L is $L^n = \{ w_1 \dots w_n | \ w_1, \dots, w_n \in L\}$. Moreover, $L^1 = L$, $L^0 = \{\varepsilon\}$.

The languages $\varnothing$ and $\{\varepsilon\}$ are zero and unit elements with respect to catenation i.e., for any language $L, L\varnothing = \varnothing L = \varnothing$ and $L\{\varepsilon\} = \{\varepsilon\}L = L$.

For example, given $\Sigma = \{0, 1\}$, let $L_1 = \{01, 1\}$, $L_2 = \{00, 10\}$, $L_3 = \{w \in \Sigma^* | w$ has even number of $0's\}$ and $L_4 = \{w | w$ starts with a 0 and the rest of the symbols are $1's\}$, then $L_1L_2 = \{0100, 0110, 100, 110\}$ and $L_3L_4 = \{w |$ w has an odd number of $0's\}$.

3.2.2 Concatenation Closure

The catenation closure of a language L, denoted L^*, is defined as the set obtained by catenating strings of L in all possible ways i.e.,

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

This is also called *Kleene Star* or *Kleene closure*.

The positive closure L^+ of L is

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

It contains the empty string only if it is already in L.

Observe that $L^* = L^+ \cup \{\varepsilon\}$, $L^+ = LL^* = L^*L$.

$L^* = L^+$ iff $\varepsilon \in L$. Also, $\varnothing^* = \{\varepsilon\}$ and $\varnothing^+ = \varnothing$.

For any two languages L_1 and L_2 over an alphabet Σ, the union $L_1 \cup L_2$ of L_1 and L_2 is the language $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1$ or $w \in L_2\}$.

The intersection $L_1 \cap L_2$ of L_1 and L_2 is the language $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}$.

The difference $L_1 - L_2$ of L_1 and L_2 is the language $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \notin L_2\}$. The difference is also called the *relative complement*.

The *complement* $\overline{L}$ of a language L is $\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$. Thus $\overline{L} = \Sigma^* - L$.

3.2.3 Mirror image

The *mirror image* of a string is obtained by reversing the order of its elements. The mirror image of a string u, denoted u^R, is defined inductively as follows:

$$\varepsilon^R = \varepsilon$$

$$(ua)^R = au^R$$

where $a \in \Sigma$ and $u \in \Sigma^*$. Also, for $u, v \in \Sigma^*, (uv)^R = v^R u^R$, and when $u_i \in \Sigma$, $(u_1 \ldots u_n)^R = u_n \ldots u_1$.

That is, the mirror image of the string $u = a_1 a_2 \ldots a_n$ is the string $u^R = a_n \ldots a_2 a_1$.

u is called a *palindrome* if $u = u^R$, e.g., $u = abba$ is a palindrome.

The mirror image of a language L is the language $L^R = \{u^R | u \in L\}$.

Definition 3.2.2 Prefix, Suffix and Substring

Let Σ be an alphabet and $u, w \in \Sigma^*$.

u is a *substring* of w iff there exist $p, q \in \Sigma^*$ such that $w = puq$. In other words, a substring (also, called *segment*, *subword* or *factor*) of a string is any sequence of

consecutive symbols that appear in the string.

The term *substrings of a string* represent the set of all substrings of a string.

u is a *prefix* of w iff there is some $q \in \Sigma^*$ such that $w = uq$. That is, a prefix of a string u is a substring of u that occurs at the beginning of u.

The term *prefixes of a string* represent the set of all prefixes of a string.

u is a *suffix* of w iff there is some $p \in \Sigma^*$ such that $w = pu$. That is, a suffix of a string u is a substring that occurs at the end of u.

The term *suffixes of a string* represent the set of all suffixes of a string.

Observe that a prefix is any sequence of leading symbols of the string, and a suffix is any sequence of trailing symbols of the string. Moreover, a substring of a string is a prefix of a suffix of the string, equivalently, a suffix of a prefix of the string.

u is a proper prefix (suffix or substring) of w iff u is a prefix (respectively, suffix or substring) of w and $u \neq w$.

Note that prefixes and suffixes both are substrings, but the converse need not hold.

Example 3.2

Let $w = abbbaa$ be a string over $\Sigma = \{a, b\}$. Its prefixes, suffixes and substrings are as follows:

Prefixes: $\{\varepsilon, a, ab, abb, abbb, abbba, abbbaa\}$.

Suffixes: $\{\varepsilon, a,\ aa,\ baa,\ bbaa,\ bbbaa,\ abbbaa\}$.

Substrings: $\{\varepsilon, a, b, ab, ba, bb, abb, bbb, abbb, bba, bbba, abbba, aa, baa, bbaa, bbbaa, abbbaa\}$.

The *border* of a string is both its prefix and suffix, e.g., aba is a border of $ababa$.

3.2.4 Prefix, Suffix and Substring Closures

Let L be a language.

Prefix closure of L:

$pref_L(v) = \{u|\ v = uy;\ v \in L, u, y \in \Sigma^*\}$, and

$$pref(L) = \cup_{v \in L}\, pref_L(v) = \{u|\ v = uy;\ v \in L, u, y \in \Sigma^*\}.$$

Suffix closure of L:

$suff_L(v) = \{u|\ v = xu;\ v \in L, x, u \in \Sigma^*\}$, and

$$suff(L) = \cup_{v \in L}\, suff_L(v) = \{u|\ v = xu;\ v \in L, x, u \in \Sigma^*\}.$$

Substring closure of L:

$subs_L(v) = \{u|\ v = xuy;\ v \in L, x, u, y \in \Sigma^*\}$, and

$$subs(L) = \cup_{v \in L}\, subs_L(v) = \{u|\ v = xuy;\ v \in L, x, u, y \in \Sigma^*\}.$$

Example 3.3

Let $L = \{ab, aba\}$ over $\Sigma = \{a,\ b\}$. Then

$$pref(L) = \{\varepsilon, a, ab, aba\},$$

$$suff(L) = \{\varepsilon, b, ab, a, ba, aba\}, \text{ and}$$

$$subs(L) = \{\varepsilon, a, b, ab, ba, aba\}.$$

A language is said to be *prefix, suffix or substring closed* if $pref(L) = L$, $suff(L) = L$ or $subs(L) = L$, respectively.

For instance, $L = \{\varepsilon, a, b\}$ is both prefix, suffix and substring closed.

The prefix, suffix and substring closure operators of L are idempotent:

$$pref(pref(L)) = pref(L),\ suff(suff(L)) = suff(L),\ subs(subs(L)) = subs(L).$$

Definition 3.2.3 Superstring

Given a language L of n strings over an alphabet Σ, a *superstring* of L is a single string that contains every string of L as a substring. Thus, catenation of the strings of L in any order gives a trivial superstring of L.

Example 3.4

Let $L = \{aabb, abaa, abbba\}$ over $\Sigma = \{a, b\}$. Then $aabbabaaabbba$, $abaabbba$ and $aabbbabaa$ are some of the superstrings of L.

3.2.5 Quotients

The right quotients of the languages L_1 and L_2 are defined as $L_1/L_2 = \{u_1 |\ u_1u_2 \in L_1$ for some string $u_2 \in L_2\}$. That is, the right quotient L_1/L_2 is the language containing all strings obtained by removing from the end of $L_1's$ strings a suffix that belongs to L_2.

The left quotient of the languages L_1 and L_2 are defined as $L_1 \backslash L_2 = \{u_2 |\ u_1u_2 \in L_2$

for some string $u_1 \in L_1\}$ i.e., remove from strings of L_2 prefixes belonging in L_1 in all possible ways.

Example 3.5

Let $L_1 = \{abaa,\ a\}$, $L_2 = \{aa,\ baa\}$, $L_3 = \{a^*ba^*\}$. Then $L_1/L_2 = \{ab,\ a\}$,

$L_2/L_2 = \{\varepsilon,\ b\}$, $L_3/L_1 = \{a^*,\ a^*ba^*\}$ and $L_3/L_3 = \{a^*\}$.

$L_1 \backslash L_2 = \{a\}$ and $L_1 \backslash L_3 = \{a^*,\ a^*ba^*\}$.

3.2.6 Homomorphism

Let Σ and Δ be two alphabets. A homomorphism from Σ to Δ is a function $h:\ \Sigma^* \longrightarrow \Delta^*$ that assigns to each letter $a \in \Sigma$ a string $h(a) \in \Delta^*$.

Example 3.6

$h(a) = 0$ and $h(b) = 01$ give a homomorphism from $\Sigma = \{a,\ b\}$ to $\Delta = \{0,\ 1\}$.

Homomorphisms are applied to strings by coding each letter separately and catenating the results together: $h(a_1a_2 \ldots a_n) = h(a_1)h(a_2) \ldots h(a_n)$.

For instance, using the homomorphism h above, we have

$h(ba) = 010$, $h(babba) = 01001010$ and $h(\varepsilon) = \varepsilon$.

Homomorphisms are applied to languages as well:

The homomorphic image $h(L)$ of the language L consists of all homomorphic images of the strings of L: $h(L) = \{h(w) \mid w \in L\} = \bigcup_{w \in L} \{h(w)\}$.

Our sample homomorphism h gives

$h(\{b,\ ab,\ bbb\}) = \{01,\ 010,\ 010101\}$,

$h(b^*) = h(\{\varepsilon,\ b,\ bb,\ bbb,\ \ldots\}) = \{\varepsilon,\ 01,\ 0101, 010101,\ \ldots\} = (01)^*$,

$h(a^*b,\ bb) = \{0^*01,\ 0101\}$, for example.

3.2.7 Inverse homomorphism

Let h be a homomorphism from Σ to Δ. The inverse homomorphism h^{-1} is defined on languages of alphabet Δ. If $L \subseteq \Delta^*$ then language h^{-1} consists of all strings over Σ that are mapped by homomorphism h into L i.e., $h^{-1}(L) = \{w \mid h(w) \in L\}$.

If L contains only one string we may simply write $h^{-1}(w)$ instead of $h^{-1}(\{w\})$.

Example 3.7

Given homomorphism $g(a) = 01$, $g(b) = 011$, $g(c) = 101$, and $g(d) = 1$.

we have $g^{-1}(011101) = \{bc\}$, $g^{-1}(01101) = \{ac,\ ba\}$,

$g^{-1}(010) = \varnothing$ and $g^{-1}((10)^*1) = \{d,\ ca^*\}$.

3.2.8 Substitution

Let Σ and Δ be alphabets. A *string substitution* or simply a *substitution* is a mapping f that maps letters in Σ to languages in Δ^*. That is, given a letter $a \in \Sigma$, we have $f(a) = L$, where $L \subseteq \Delta^*$. For the empty string ε, $f(\varepsilon) = \varepsilon$ and, for string $s \in L \subseteq \Sigma^*$,

$$f(sa) = f(s)f(a).$$

Example 3.8

$f(a) = \{0,\ 11\}$, $f(b) = \{0^*10^*\}$ is a substitution from $\Sigma = \{a,\ b\}$ to $\Delta = \{0,\ 1\}$.

The substitution is called *finite* if $f(a)$ is a finite language for each $a \in \Sigma$. The substitution f above is not finite, and substitution s given by $s(a) = \{000,\ 11\}$,

$s(b) = \{01,\ \varepsilon\}$ is finite.

A substitution s is also applied to strings by applying it to each letter separately and catenating the results (which are languages) together: $s(a_1a_2\ldots a_n) = s(a_1)s(a_2)\ldots s(a_n)$.

Example 3.9

Using our sample substitutions s and f, we have

$s(ba) = \{01,\ \varepsilon\}\{000,\ 11\} = \{01000,\ 0111,\ 000,\ 11\}, f(aba) = \{0,\ 11\}\ \{0^*10^*\}\{0,\ 11\}, f(\varepsilon) = \{\varepsilon\}$.

String substitutions may be extended to entire languages as well viz.,

$$f(L) = \cup_{s\epsilon L} f(s).$$

For instance, $f(ba,\ aba) = f(ba) \cup f(aba) = \{0^*10^*\}\{0,\ 11\} \cup \{0,\ 11\}\{0^*10^*\}\{0,\ 11\}, f(ba^*) = \{0^*10^*\}\{0, 11\}^*$.

3.2.9 Recursion

The process of *recursion* is concerned with defining the value of a function by using other values of the same function. Such functions are called *recursively defined functions* and constitute an important class of *computable* functions. A commonly cited example of a recursive function is the Fibonacci function:

$$fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{otherwise.} \end{cases}$$

In the context of strings, the set Σ^* of strings over the alphabet Σ can be recursively defined by taking the empty string belonging to Σ^* and generating new strings of Σ^* with symbols from Σ.

This brief introduction to recursion is largely confined to numberic functions, in particular, functions: $\mathbb{N}^n \longrightarrow \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers, including zero viz., $[0, 1, 2, \ldots]$. Note that such a function is *total* unless it is defined on a subset of $\mathbb{N}^n$, in that case, it is called *partial*. In fact, the codomain $\mathbb{N}$ of a numeric function can be as well generalized to $\mathbb{R}$, the set of real numbers. In the following, a function of n variables $x_1, x_2, \ldots, x_n$ will be denoted as $f(x_1, x_2, \ldots, x_n)$.

The following definitions are required to formally define the notion of *recursive functions*.

Initial functions

Z: $\mathbf{Z}(x) = 0$	*Zero function*
$Suc(x) = x + 1$	*Successor function*
$\Pi_i^n : \Pi_i^n (x_1, x_2, \ldots, x_n) = x_i$	*Projection function (also called the generalized identity function)*

Composition

The operation of composition is the usual one defined for functions.

Recursion

Formally, the process for defining a function $f(x_1, x_2, \ldots, x_n, y)$ of $n+1$ variables by using two other functions $g(x_1, x_2, \ldots, x_n)$ and $h(x_1, x_2, \ldots, x_n, y, z)$ of n and $n+2$ variables, respectively, is called *recursion*:

$f(x_1, x_2, \ldots, x_n, 0) = g(x_1, x_2, \ldots, x_n)$,

$f(x_1, x_2, \ldots, x_n, y+1) = h(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y))$.

Here, y is taken as the inductive variable in the sense that the value of f at $y+1$ is defined by the value of f at y. The variables $x_1, x_2, \ldots, x_n$ are treated as parameters and are assumed to remain fixed and both the functions g and h known.

Regular function

Let $g(x_1, x_2, \ldots, x_n, y)$ be a total function of $n+1$ variables where y is the inductive variable and $x_1, x_2, \ldots, x_n$ are assumed to be fixed parameters. If there exists at least one value of y, say $\overline{y} \in \mathbb{N}$, such that the function $g(x_1, x_2, \ldots, x_n, \overline{y}) = 0$ for all $n-$tuples $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$, then g is called a *regular* function.

For example, the function $g(x, y) = y \dot{-} x$, where $\dot{-}$ is a *proper subtraction function* (also called *limited subtraction*, denoted sub(x,y), and read as *x is subtracted from y* is a regular function since $g(x, y) = 0$ for all x if $y = 0$. However, not all total functions are regular, for example, $g(x, y) = |2y - x|$ is total (since defined for all x, y) but $|2y - x| = 0$ holds only for those values of x which are twice the value of y and not for all values of x i.e., only for even values of x.

μ ***operation***

A function $f(x_1, x_2, \ldots, x_n)$ is said to be defined from a total function $g(x_1, x_2, \ldots, x_n, y)$ by minimalization (or minimization) or μ operation if

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} \mu_y(g(x_1, x_2, \ldots, x_n, y)) = 0 & \text{if there exists such a } y \\ \text{indeterminate} & \text{otherwise} \end{cases}$$

where μ_y means the least $y \geq 0$.

By the above definition, it follows that f is total if g is regular. In case, if g is not regular, then the operation of minimization may give rise to a partial function.

Primitive recursive function

A function is called *primitive recursive* if and only if it can be obtained from the initial functions by a finite number of applications of composition and recursion operations. For example, the predecessor function P defined as $P(0) = 0,\ P(y+1) = y = \Pi_1^2(y, P(y))$ is a primitive recursive function.

Recursive function

A function is called *recursive* if and only if it can be defined from the initial functions by a finite number of applications of composition, recursion and minimization operations over a regular function. For example, the Fibonacci function given earlier is a recursive function.

It follows from the definition of regular and primitive recursive functions that the class of primitive recursive functions is properly contained in the class of recursive functions.

Partial recursive function

A function is called *partial recursive* if and only if it can be obtained from the initial functions by a finite number of applications of composition, recursion, and minimization. Note that by waiving the condition of minimization to be necessarily over regular functions required for defining recursion functions, a larger class of partial recursive functions is obtained.

The function $f(x) = x/2$ is a partial recursive function, since $f(x) = \mu_y(|2y - x| = 0)$ and $g(x, y) = |2y - x|$ is not regular.

3.3 Some Algebraic Structures of Languages

It is known (Ahmadi et al., 2014) that certain algebraic structures have useful applications in formal languages. Moreover, a number of algebraic structures of partitions of a set and that of an integer have been developed which have useful applications in computer arithmetic, formal languages, and sequential machines (see (Knuth, 1981), (Tremblay and Manohar, 1997) for details). In view of the above, we define suitable operations on the set of partitions of a language and develop certain monoids and semigroup. Also, certain algebraic structures of a language defined over a string are

described.

Definition 3.3.1 Cardinality bounded languages

Let X^{n*}, henceforth called a *cardinality bounded language* over an alphabet X, denote the set of all strings of length $\leq n$ over X, that is, $X^{n*} = X^n \cup X^{n-1} \cup \cdots \cup X^0$. In other words, $\{X^{n*}\}$ is a strictly monotonic increasing nested sequence i.e., $X^{0*} \subset X^{1*} \subset \cdots$, and obviously $X^* = X^{0*} \cup X^{1*} \cup \cdots \cup X^{n*} \cup \ldots$. However, it gives an alternative form of representation in line the usual one viz., $X^* = X^0 \cup X^1 \cup \cdots \cup X^n \cup \ldots$, where X^n is the set of all strings of length n over X.

Moreover, $\bigcup_{n=0}^{\infty} X^n = \bigcup_{n=0}^{\infty} X^{n*}$, but $\bigcap_{n=0}^{\infty} X^n = \varnothing$, whereas $\bigcap_{n=0}^{\infty} X^{n*} = \{\varepsilon\}$.

Also, each of X^{n*} is a well-ordered set with $\subset$ (inclusion), and hence a finite *ordinal*, say $\alpha, \beta, \gamma, \ldots$, satisfying the following properties:

(i) $\beta \in \alpha \Longrightarrow \beta \subset \alpha$ (ii) each α is well-ordered by $\subset$ and (iii) neither α nor its element is an element of itself.

For example, let $X = \{0, 1\}$, then $X^{0*} = \{\varepsilon\}$, $X^{1*} = \{\varepsilon, 0, 1\}$, $X^{2*} = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$, $X^{3*} = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$, and so on.

Let us recapitulate that the cardinality of a language L, denoted $|L|$, is the number of strings in L. Thus X^* is countably infinite over any X.

Theorem 3.3.1 The Cardinality of the language X^{n*} over an alphabet X is $\Sigma_{i=0}^{n} k^n$, where k is the cardinality of the alphabet.

Proof

Let $X = \{0\}$. Then,

$|X^{n*}| = |X^0| + |X^1| + |X^2| + \cdots + |X^n| = |\{\varepsilon\}| + |\{0\}| + |\{00\}| + |\{000\}| + \cdots + |\{0\}^n| = 1^0 + 1^1 + 1^2 + \cdots + 1^n$.

Let $X = \{0, 1\}$. Then,

$$|X^{n*}| = \left|X^0\right| + \left|X^1\right| + \left|X^2\right| + \cdots + |X^n| = |\{\varepsilon\}| + |\{0, 1\}| + |\{00, 01, 10, 11\}| +$$

$$+ |\{000, 001, 010, 011, 100, 101, 110, 111\}| + \cdots + |\{0, 1\}^n| = 2^0 + 2^1 + 2^2 + \cdots + 2^n.$$

Let $X = \{0, 1, 2\}$. Then,

$$|X^{n*}| = \left|X^0\right|+\left|X^1\right|+\left|X^2\right|+\cdots+|X^n| = |\{\varepsilon\}|+|\{0, 1, 2\}|+|\{00, 01, 02, 10, 11, 12, 20, 21, 22\}|+$$

$$|\{000, 001, 002, 010, 011, 012, 020, 021, 022, 100, 101, 102, 110, 111, 112, 120, 121, 122, 200, 201,$$

$$202, 210, 211, 212, 220, 221, 222\}| + \cdots + |\{0, 1, 2\}^n| = 3^0 + 3^1 + 3^2 + \cdots + 3^n.$$

Let $X = \{0, 1, 2, 3\}$. Then,

$$|X^{n*}| = \left|X^0\right|+\left|X^1\right|+\left|X^2\right|+\cdots+|X^n| = |\{\varepsilon\}|+|\{0, 1, 2, 3\}|+|\{00, 01, 02, 03, 10, 11, 12, 13,$$

$$20, 21, 21, 22, 23, 30, 31, 32, 33\}|+|\{000, 001, 002, 003, 010, 011, 012, 013, 020, 021, 022, 023,$$

$$030, 031, 032, 033, 100, 101, 102, 103, 110, 111, 112, 113, 120, 121, 122, 123, 130, 131, 132, 133,$$

$$200, 201, 202, 203, 210, 211, 212, 213, 220, 221, 222, 223, 230, 231, 232, 233, 300, 301, 302, 303,$$

$$310, 311, 312, 313, 320, 321, 322, 323, 330, 331, 332, 333\}| = 4^0 + 4^1 + 4^2 + \cdots + 4^n.$$

By induction, if X be a $k-$element set, we have

$$|X^{n*}| = k^0 + k^1 + k^2 + \cdots + k^n.$$

3.3.1 Monoids of equivalence classes of a partition of a language

Definition 3.3.2 Let R_a be a relation on X^* such that for $s, t \in X^*, sR_at$ if and only if s and t are of equal length. It is easy to see that R_a is an equivalence relation on X^* and hence, it partitions X^* into its equivalence classes. In other words, a partition of X^* can be viewed as a collection of disjoint languages of X^*, whose union is X^*.

Theorem 3.3.2 $(X^*/R_a, \star)$ is a monoid and $(L/R_a, \star)$ is a commutative monoid under catenation of equivalence classes, where $L = u^*, u \in X^*$.

Proof

Let the equivalence class generated by $S \in X^*$ be denoted $[S]_{R_a}$ or simply $[S]$, and the quotient set X^*/R_a denote the family of all equivalence classes of X^*.

Let us define an operation $\star$ on X^*/R_a such that $[s] \star [t] = [st]$ where st is the catenation of s and t. Then, $(X^*/R_a, \star, [\varepsilon])$ is a monoid of the partition of X^* induced by R_a, where $[\varepsilon]$ is the identity of catenation. The operation $\star$ on X^*/R_a is neither commutative nor idempotent, in general. However, the identity element $[\varepsilon]$ is the only idempotent element.

Also, the operation $\star$ on X^*/R_a is commutative if X is a singleton.

Let u^* be defined as the set of all strings over $u \in X^*$. Then, $(L/R_a, \star, [\varepsilon])$ is a commutative monoid where $L = \mathrm{u}^*, \mathrm{u} \in \mathrm{X}^*$.

Similarly, for each of the relations R_b, R_c, and R_d defined on X^* as

(i) sR_bt iff both s and t have the same number of occurrences of each symbol,

(ii) sR_ct iff s and t agree in their first symbols, and

(iii) sR_dt iff s and t agree in their last symbols;

the respective quotient set is a non-commutative and non-idempotent monoid. Moreover, each of R_b, R_c, and R_d, similar to R_a, defined on a language u^*, partitions it, and the respective quotient set is a commutative monoid of the partitions of u^*.

3.3.2 Monoids of partitions of a language

Definition 3.3.3 Let $\mathcal{F}(X^*)$ denote the collection of all partitions of X^* and $S = \{S_1, S_2, \cdots\}$ and $T = \{T_1, T_2, \cdots\}$ be two partitions of X^*. That is, $S_i's$ and $T_i's$ are the blocks of S and T, respectively, and each block is a subset of X^*.

Theorem 3.3.3 $(\mathcal{F}(X^*), \odot)$ is a commutative, idempotent monoid, where $\odot$ is defined as follows: For any $S, T \in \mathcal{F}(X^*), S \odot T$ consists of the set of nonempty intersections of every block of S with every block of T.

Proof

As intersection on languages is associative and commutative, the operation $\odot$ on $\mathcal{F}(X^*)$ is both associative and commutative. Moreover, the partition consisting of a unique single block is the identity of $\odot$. Also, $P \odot P = P$ for all $P \in \mathcal{F}(X^*)$ i.e., $\odot$ on $\mathcal{F}(X^*)$ is idempotent. Thus, $(\mathcal{F}(X^*), \odot)$ or $(\mathcal{F}(X^*), \odot, \{\overline{X^*}\})$ is a commutative, idempotent monoid.

Theorem 3.3.4 $(\mathcal{F}(X^*), \oplus)$ is a commutative, idempotent monoid, where $\oplus$ is defined as follows: For $S, T \in \mathcal{F}(X^*)$. A subset P of X^* belongs to $S \oplus T$ if

(i) P is the union of one or more elements of S;

(ii) P is the union of one or more elements of T; and

(iii) No element of P satisfies (i) and (ii) except P itself.

Proof

Clearly, $\oplus$ on $\mathcal{F}(X^*)$ is associative and commutative as union on languages is associative and commutative. The partition consisting of singleton blocks is the identity

of the operation $\oplus$ on $\mathcal{F}(X^*)$. Thus, $(\mathcal{F}(X^*), \oplus)$ or $(\mathcal{F}(X^*), \oplus, \{\overline{x_0}, \overline{x_1}, \overline{x_2}, \cdots\})$ where $x_i's$ are the elements of X^*, is a commutative, idempotent monoid.

Theorem 3.3.5 $(\mathcal{F}(X^*), \oslash)$ is a semigroup, where $\oslash$ is defined as follows: For any $S, T \in \mathcal{F}(X^*), S \oslash T$ is the union of every block of S with every block of T if no element of the block of S and/or T appears previously. In the case, a block has an element that appeared previously, it is not included in the union.

Proof

It is immediate to see that $\oslash$ is associative but non-commutative and every element $P \in \mathcal{F}(X^*)$ is idempotent as $P \oslash P = P$ holds. Thus, $(\mathcal{F}(X^*), \oslash)$ is only a semigroup as there is no identity element.

It is immediate to see that all the foregoing theorems hold good for X^{n*} as well.

Example 3.10

Let $X = \{0, 1\}$ be an alphabet and $n = 2$. Then, $X^{2*} = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$. Let $S, T \in \mathcal{F}(X^*)$ where $S = \{\overline{\varepsilon, 0, 1, 00, 01, 10, 11}\}$ and $T = \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10}, \overline{11}}\}$. Then, the following hold:

(i) $S \odot T = \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10}, \overline{11}}\} \in \mathcal{F}(X^*), S \odot S = \{\overline{\varepsilon, 0, 1, 00, 01, 10, 11}\} = S$, and $S \odot T = T \odot S$. Similarly, results could be computed to show associativity. Thus, $(\mathcal{F}(X^{2*}), \odot)$ is a commutative, idempotent monoid with $\{\overline{\varepsilon, 0, 1, 00, 01, 10, 11}\}$ as the identity.

(ii) $S \oplus T = \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10, 11}}\} \in \mathcal{F}(X^{2*}), S \oplus S = \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10, 11}}\} = S$, and $S \oplus T = T \oplus S$. Moreover, $I = \{\overline{\varepsilon}, \overline{0}, \overline{1}, \overline{00}, \overline{01}, \overline{10}, \overline{11}\}$ is the identity element since $I \oplus T = \{\overline{\varepsilon}, \overline{0}, \overline{1}, \overline{00}, \overline{01}, \overline{10}, \overline{11}\} \oplus \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10}, \overline{11}}\} = \{\overline{\varepsilon, 0, 1, \overline{00, 01, 10}, \overline{11}}\} = T$, for

any T. Results could be computed to show that $\oplus$ is associative. Thus, $(\mathcal{F}(X^{2*}), \oplus)$ is a commutative, idempotent monoid with $\{\overline{\varepsilon}, \overline{0}, \overline{1}, \overline{00}, \overline{01}, \overline{10}, \overline{11}\}$ as the identity.

(iii) $S \oslash T = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} \in \mathcal{F}(X^{2*}), T \oslash S = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10, 11}\} \in \mathcal{F}(X^{2*})$, and $T \oslash S \neq S \oslash T$. Moreover, $T \oslash T = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} \oslash \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} = T$. In order to show associativity, let $R = \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\}$. Then, $(S \oslash T) \oslash R = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} \oslash \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\} = \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\}$, and $S \oslash (T \oslash R) = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10, 11}\} \oslash (\{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10}, \overline{11}\} \oslash \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\}) = \{\overline{\varepsilon, 0, 1}, \overline{00, 01, 10, 11}\} \oslash \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\} = \{\overline{\varepsilon, 0, 1, 00}, \overline{01, 10}, \overline{11}\}$ i.e., $(S \oslash T) \oslash R = S \oslash (T \oslash R)$. Thus, $(\mathcal{F}(X^{2*}), \oslash)$ is an idempotent semigroup.

3.3.3 Some algebraic structures of a language defined over a string

$(u^*, \star)$ is a commutative monoid under catenation where u^* denotes the set of all strings over $u \in X^*$. Also, the monoid $C = (u^*, \star)$ is isomorphic to the monoid $N = (\mathbb{N}, .)$, where $\star$ and . denote catenation and multiplication, respectively.

Proof

Let $f : C \longrightarrow N$ be a function defined as

$$f(u) = \begin{cases} 1, & \text{if } u = \varepsilon, \\ n, & \text{if } u = u^n, \forall u \in C, \end{cases}$$

where u^n is the $n-$times catenation of u itself.

Then, $\forall u, v \in C$, since $f(uv) = f(u_1u_2\cdots u_nv_1v_2\cdots v_n)$
$= f(u_1)f(u_2)\cdots f(u_n)f(v_1)f(v_2)\cdots f(v_n) = f(u_1u_2\cdots u_n)f(v_1v_2\cdots v_n) = f(u)f(v)$, the function f is a monoid homomorphism. Let $u, v \in C$ such that $f(u) = f(v)$ i.e.,

$f(u_1)f(u_2)\cdots f(u_n) = f(v_1)f(v_2)\cdots f(v_n)$. Then, as strings are ordered, we have $u_1 = v_1, u_2 = v_2, \cdots, u_n = v_n$ i.e., $u = v$, which imply that f is injective. Moreover, by the definition of $f, \forall n \in N, \exists u \in C$ such that $f(u) = n$ i.e., f is surjective. Hence f is an isomorphism.

Theorem 3.3.6 A finite $C = (u^*, \star)$ is a cyclic group of order n.

Proof

Let u^* be represented as $\{u^0, u^1, \cdots, u^{n-1}, \cdots\}$. A finite C can be represented as $(u_n^*, \star)$ where u_n^* is the set of n elements of u^*. Let C be finite viz.,
$C = \{C^i, \star\}, i = 0, 1, \cdots, n-1$, where

$$C^i = \begin{cases} C^{i+1}, \ 0 \leq i < n-1 \\ C^0, \quad i = n-1. \end{cases}$$

Let $C^iC^j = C^{i+j}, i + j < n$ and $C^iC^j = C^{i+j-n}, i + j \geq n$. Then, C is a cyclic group of order n.

Example 3.11

Let $u = bba$, $u^* = \{\varepsilon, bba, bbabba, \dots\}$ and $n = 7$. Then, $C = \{C^0, C^1, C^2, C^3, C^4, C^5, C^6\} = \{\varepsilon, bba, bbabba, bbabbabba, bbabbabbabba, bbabbabbabbabba, bbabbabbabbabbabba\}$.

From C, $C^1C^2 = C^3$, $C^4C^5 = C^2$, $C^6C^1 = C^0$, etc. Thus, C is a cyclic group of order 7.

Theorem 3.3.7 Languages of a finite u^* form a bounded distributive lattice.

Proof

Let a finite u^* be represented as $u^{n\circledast} = \mathrm{u}^0 \cup \mathrm{u}^1 \cup \ldots \cup \mathrm{u}^{n-1}$, and G be the set of all possible languages of $\mathrm{u}^{n\circledast}$. Let H be a structure consisting of G with union and intersection representing the (join) $\vee$ and (meet) $\wedge$ operations, respectively. Let $L_1, L_2, L_3 \in G$. Then, $L_1 \vee L_2 = L_2 \vee L_1$, $L_1 \vee (L_2 \vee L_3) = (L_1 \vee L_2) \vee L_3$ and $L_1 \vee L_1 = L_1$ hold, as the union of languages is associative, commutative and

idempotent. Thus, $(G, \vee)$ is a commutative, idempotent semigroup. Also, as the intersection of languages is commutative, associative and idempotent, $(G, \wedge)$ is a commutative, idempotent semigroup.

Moreover, as the absorption properties hold i.e., $L_1 \vee (L_1 \wedge L_2) = L_1$ and $L_1 \wedge (L_1 \vee L_2) = L_1$. $H = (G, \vee, \wedge)$ is a lattice.

Also, $\forall L \in G$, as $L \vee \varnothing = L$, $\varnothing$ is the identity element of the join operation and, as $L \wedge G = L$, G is the identity of the meet operation. Thus, H is a bounded lattice.

In addition, as $L_1 \vee (L_2 \wedge L_3) = (L_1 \vee L_2) \wedge (L_1 \vee L_3)$ and $L_1 \wedge (L_2 \vee L_3) = (L_1 \wedge L_2) \vee (L_1 \wedge L_3)$ hold, H is a bounded distributive lattice.

Example 3.12

Let $u = 01 \in X$ over an alphabet $X = \{0, 1\}$, and $u^{3\circledast} = \{\varepsilon, 01, 0101\}$.

The set G of all possible languages of $u^{3\circledast}$ is $\{\varnothing, \{\varepsilon\}, \{01\}, \{0101\}, \{\varepsilon, 01\}, \{\varepsilon, 0101\}, \{01, 0101\}, \{\varepsilon, 01, 0101\}\}$.

Then, $\{01\} \vee (\{01\} \wedge \{01, 0101\}) = \{01\}$, $\{01\} \wedge (\{01\} \vee \{01, 0101\}) = \{01\}$ i.e., absorption properties hold. Also, $\varnothing \vee \{01, 0101\} = \{01, 0101\}$ and $\{\varepsilon, 0101\} \wedge G = \{\varepsilon, 0101\}$ i.e., $\varnothing$ is the identity for $\vee$, and G is the identity for $\wedge$. Similarly, results for various other combinations could be computed.

Thus, $(G, \vee, \wedge,)$ is a bounded distributive lattice.

CHAPTER FOUR

GRAMMARS AND AUTOMATA

As described earlier, a formal language is a set of strings over an alphabet and it is devoid of any meaning to strings. It can be finite or infinite. A finite language can be described by listing its strings which is not the case for infinite languages; even for some finite languages the listing could be quite cumbersome. Moreover, in general, for a language to be interesting mathematically, it needs to have an infinite set of strings.

Formal language theory (FLT) develops mathematical and computational techniques for specifying languages. How simple or complex a language could be depends on the level of its description. There are two fundamental machineries for defining or specifying languages: *grammars* (which generate the strings) and *automata* (which recognize the strings) of a language. Regular expressions could also be used to define regular languages. The significance of a finite device, such as a grammar, lies in the fact that it provides means to investigate the structure of an infinite language as well.

In this chapter, machineries for defining languages: grammars and automata are described and exemplified.

4.1 Grammars: Chomsky's Hiererchy

A grammar is a finite generating device which provides structure on the strings or sentences of an infinite language. A language is called *computably enumerable* if it can be defined by some formal grammar. Equivalently a computably enumerable

language belongs to the set of provable statements of the first-order logic.

Essentially, a grammar of a language L is a finite set of *production* or *grammatical rules* that describe how to form valid strings from the alphabet of L. In other words, a grammatical structure of a language provides means to determine whether or not a string belongs to the set of valid strings. A computer programmer is mainly concerned with applying the production rules (or rules of syntax) to produce syntactically correct programs.
In this section, mainly following ((Jiang et al., 2010), (Tremblay and Manohar, 1997)), formulations of various grammars are presented.

Schematically, a grammar is a quadruple (or 4–tuple) $G = (\Sigma_N, \Sigma_T, S, P)$ where Σ_N is a (finite) set of non-terminals, Σ_T is a (finite) set of terminals where $\Sigma_T \cap \Sigma_N = \varnothing$, $S \in \Sigma_N$ is a distinguished element called the *start symbol*, and P is a finite set of the relations from $(\Sigma_T \cup \Sigma_N)^* \Sigma_N (\Sigma_T \cup \Sigma_N)^*$ to $(\Sigma_T \cup \Sigma_N)^*$ and an element (α, β) of P is usually represented as $\alpha \longrightarrow \beta$, called a *production rule* (or simply *production*) or *rewriting rule*. Note that α, β are strings over $\Sigma_N \cup \Sigma_T$ (called the *vocabulary* of the language), α has at least one symbol from Σ_N, terminals are used to constitute the sentences, non-terminals are the intermediate symbols used to constitute the structure of the sentences, and the production rules specify how sentences are formed (see (Ruohonen, 2009) and (Jiang et al., 2010) for details). For example, let $\Sigma_T = \{0, 1\}$, $\Sigma_N = \{S\}$, and $P = \{S \longrightarrow 0S1, S \longrightarrow \varepsilon\}$. Then $G = (\Sigma_T, \Sigma_N, S, P)$ is a grammar.

The set of all distinct strings obtained by invoking the production rules of the grammar G is the *language* generated by G, denoted $L(G)$ i.e., $L(G)$ is a subset of the set of all terminal strings over Σ_T. The objective of applying production rules in succession is to obtain from a given string containing terminals and/or non-terminals,

the end string containing only terminals. Any specific sequence of productions on the start symbol yields a distinct string in the language. If there is more than one way to generate the same string, the grammar is said to be ambiguous. In other words, as long as a string contains either S or any designated non-terminal, a new string can be obtained from it.

The following are some typical forms of derivations that often occur in course of applying production rules.

Let G be a grammar, $\alpha \longrightarrow \beta$ a production rule, and $u, v \in (\Sigma_T \cup \Sigma_N)^* - \{\varepsilon\}$. Then, v is called a *one-step* or *direct* derivation of u, denoted $u \Longrightarrow v$, if there exist strings w_1 and w_2 (possibly empty) such that $u = w_1\alpha\, w_2$ and $v = w_1\ \beta w_2$. This proof theoretic concept extends to cases in which derivation may be completed in a sequence of steps which can be described as follows:

A sequence of strings $\{v_0, v_1, \ldots, v_k\}$, $k \geq 0$ is called a *derivation* of the string v from the string u, written as $u \Longrightarrow^* v$, if $u = v_0 \Longrightarrow v_1 \Longrightarrow v_2 \Longrightarrow \ldots \Longrightarrow v_{k-1} \Longrightarrow v_k = v$. In the case $k = 0$, $u = v$. The relation $\Longrightarrow^*$ is the reflexive transitive closure of $\Longrightarrow$.

Moreover, a derivation of v from the unique non-terminal symbol S viz., $S \Longrightarrow v_1 \Longrightarrow v_2 \Longrightarrow v_3 \Longrightarrow \ldots \Longrightarrow v_n \Longrightarrow v$, written as $S \Longrightarrow^* v$, is called a *sentential form*, and $L(G) = \{v \in \Sigma^* : S \Longrightarrow^* v\}$.

A grammar G is said to generate a string w comprising of only terminals if and only if there exists a derivation of w which starts with S and produces w through a finite sequence of productions applied in any order. Moreover, if such a derivation sequence exists, none of the intermediate string can be longer than w.

Example 4.1

Let a grammar $G = (\{a, b\}, \{S, A, B, C\}, S, P)$ where the production rule P is given as:

$$S \longrightarrow BA/BC, A \longrightarrow SC, B \longrightarrow a, C \longrightarrow b$$

The application of the production rule of G yields

$$S \Longrightarrow BC \Longrightarrow ab,$$

$$S \Longrightarrow BA \Longrightarrow aA \Longrightarrow aSC \Longrightarrow aSb \Longrightarrow aBCb \Longrightarrow aabb,$$

$$S \Longrightarrow BA \Longrightarrow aA \Longrightarrow aSC \Longrightarrow aSb \Longrightarrow aBAb \Longrightarrow aaAb \Longrightarrow aaSCb \Longrightarrow aaSbb$$

$$\Longrightarrow aaBCbb \Longrightarrow aaabbb,$$

Continued application of P yields the set $\{ab, aabb, aaabbb, \dots\}$, thus $L(G) = \{a^n b^n | n \geq 1\}$.

Chomsky (1956, 1959) classified grammars into the following four classes by imposing restrictions on the production rules, known as the *Chomsky hierarchy*: unrestricted grammars (Type 0), context-sensitive grammars (Type 1), context-free grammars (Type 2), and regular grammars (Type 3). The languages generated by Type 0, Type 1, Type 2 and Type 3 grammars are, respectively, called *type 0* (or *computably enumerable*), *Type 1* (or *context-sensitive*), *Type 2* (or *context-free*) and *Type 3* (or *regular*) languages (see (Jiang et al., 2010) for details).

4.1.1 Regular (or Type 3) grammars

A grammar that contains production rules of the form $\alpha \longrightarrow \beta$ where $|\alpha| \leq |\beta|$, $\alpha \in \Sigma_N$, and β is of the form αB or α where $\alpha \in \Sigma_T$ and $B \in \Sigma_N$, is called *regular*.

In other words, a grammar having all production rules of the form $A \longrightarrow a$, or $A \longrightarrow aB$, where $A, B \in \Sigma_N$ and $a \in \Sigma_T$, is said to be *regular*. That is, in any production rule, the left hand side string is a single non-terminal and the right hand side is either a terminal or a terminal followed by a non-terminal. For example, the grammar $G = (\{S, A\}, \{a, b\}, S, S \longrightarrow aA, A \longrightarrow bA/b)$ is regular and generates the regular language $L = \{ab^m \mid m \geq 1\}$.

4.1.2 Context-free (or Type 2) grammars

A grammar that contains production rules of the form $\alpha \longrightarrow \beta$ where $|\alpha| < |\beta|$, $\alpha \in \Sigma_N$, satisfying $|\alpha| = 1$ i.e., the left hand side consists of a single class symbol, is called *context-free*. In other words, in a context-free grammar, production rules are of the form $A \longrightarrow \beta$ where $A \in \Sigma_N$ and $\beta \in (\Sigma_T \cup \Sigma_N)^*$. Thus, a Type 3 grammar is also a Type 2 grammar. It is called *context-free* in the sense that it rewrites the rewriting variable in a sentential form irrespective of the other symbols in its context or neighborhood. It is also sometimes called a *phrase structure grammar* since the non-terminal symbols can be interpreted as names of syntactic categories, and the $\longrightarrow$ can be interpreted as *consists of* (see (Jager and Rogers, 2012) for details).

Although, such a grammar, owing to its simple structure, can specify large parts of the syntax for computer or artificial languages, it is not capable of representing even significant aspects of the English language (natural language) since the context dependency is usually needed in order to adequately arrange the structure of the sentences.

For example, the grammar $G = (\{S\}, \{a, b\}, S, S \longrightarrow aSb/\varepsilon)$ is of Type-2 and generates the context-free language $L = \{a^n b^n \mid n \geq 1\}$.

4.1.3 Context-Sensitive (or Type 1) grammars

A grammar that contains production rules of the form $\alpha \longrightarrow \beta$, where $|\alpha| \leq |\beta|$ is called *context-sensitive*. The restriction imposed on a production rule entails that the left-hand side is never longer than the right hand side. The term *context-sensitive* is used in the following sense:
In the production rule $\alpha \longrightarrow \beta$ of a context-sensitive grammar, α and β can be expressed as $\alpha = uAv$ and $\beta = u\Phi v$ where strings u and/or v may be possibly empty but Φ must be nonempty and $A \in \Sigma_N$. The production $\alpha \longrightarrow \beta$ i.e., $uAv \longrightarrow u\Phi v$ entails that A is rewritten as Φ in the context of u and v (see (Tremblay and Manohar, 1997) and (Xavier, 2005) for details).

In other words, a grammar G is said to be of Type 1 if each production $\alpha \longrightarrow \beta$ in P satisfies $|\alpha| \leq |\beta|$ with $S \longrightarrow \varepsilon$ allowed provided S does not appear on the right hand side of any production (see (Jiang et al., 2010) for details).

For example, the grammar $G = (\Sigma_N, \Sigma_T, S, P)$, where $\Sigma_T = \{a, b, c\}$, $\Sigma_N = \{S, A, B\}$, S the start symbol, and P consists of the following production rules: $S \longrightarrow aAbc/abc$, $Ab \longrightarrow bA$, $Ac \longrightarrow Bbcc$, $bB \longrightarrow Bb$ and $aB \longrightarrow aa/aaA$, is context-sensitive and defines the language $L(G) = \{a^n b^n c^n \mid n \geq 1\}$ as follows:

$S \Longrightarrow abc$,
$S \Longrightarrow aAbc \Longrightarrow abAc \Longrightarrow abBbcc \Longrightarrow aBbbcc \Longrightarrow aabbcc$,
$S \Longrightarrow aAbc \Longrightarrow abAc \Longrightarrow abBbcc \Longrightarrow aBbbcc \Longrightarrow aaAbbcc \Longrightarrow aabAbcc \Longrightarrow aabbAcc \Longrightarrow aabbBbccc \Longrightarrow aabBbbccc \Longrightarrow aaBbbbccc \Longrightarrow aaabbbccc$ etc., (see (Linz, 2001) for details).

4.1.4 Unrestricted (or Type 0) grammar

An unrestricted (or Type 0) grammar is a grammar having no restrictions on the left and right hand sides of the production rules. For example,

$G = (\{S, A, B, C, D, E\}, S, \{a\}, \{S \longrightarrow ACaB \mid a \mid \varepsilon, Ca \longrightarrow aaC, AD \longrightarrow AC,$

$CB \longrightarrow DB \mid E, aD \longrightarrow Da, aE \longrightarrow Ea, AE \longrightarrow \varepsilon \}$ is a Type 0 grammar and generates the language $L = \{a^n | n$ is a power of $2\}$ through the following derivations: $S \Longrightarrow ACaB \Longrightarrow AaaC\mathrm{B} \Longrightarrow AaaE \Longrightarrow AaEa \Longrightarrow AEaa \Longrightarrow aa$, $S \Longrightarrow ACaB \Longrightarrow AaaC\mathrm{B} \Longrightarrow AaaD\mathrm{B} \Longrightarrow AaDa\mathrm{B} \Longrightarrow ADaa\mathrm{B} \Longrightarrow ACaa\mathrm{B} \Longrightarrow AaaCa\mathrm{B} \Longrightarrow AaaaaC\mathrm{B} \Longrightarrow AaaaaE \Longrightarrow AaaaEa \Longrightarrow AaaEaa \Longrightarrow AaEaaa \Longrightarrow AEaaaa \Longrightarrow aaaa$, etc.

Remark 4.1

For brevity, let the class symbols T_0, T_1, T_2 and T_3 denote, respectively, the Type 0, Type 1, Type 2, and Type 3 grammars, and relatedly, let the respective classes of languages generated by these grammars be denoted $L(T_0)$, $L(T_1)$, $L(T_2)$ and $L(T_3)$. Chomsky showed that $L(T_3) \subset L(T_2) \subset L(T_1) \subset L(T_0)$ and hence the *hierarchy*. Accordingly, a language is said to be of Type k, $k = \overline{0,3}$, if it can be defined by a grammar of Type k but it cannot be defined by a grammar of Type $k+1$. In Grune and Jacobs (1990), it is shown that both $L(T_2)$ and $L(T_3)$ are found of the same power as far as the issues related to *expressibility* and *ease of parsing* alone are taken into consideration; although they are intended to deal with problems of different nature.

In course of time, both extensions and modifications on Chomsky's original formulation of hierarchy of formal grammars, such as the notions of mildly context-sensitive languages and sub-regular hierarchies have appeared (see (Joshi et al., 1990), (Rogers and Pullum, 2007), (Jäger and Rogers, 2012) for details).

Chomsky (1957) pointed out that as English has centre embedding constructions: involving two dependent elements a and b that are not adjacent, and that may

contain another instance of the same construction between the two parts, such as *neither-nor* constructions, it is not regular. Huybregts (1984) and Shieber (1985) argued that Swiss-German is not context-free as the dependencies between verbs and their objects were unbounded in length. In fact, it has crossing dependencies between objects and verbs, and the number of these interlocked dependencies is potentially unbounded. Since context-free grammars can handle an unbounded number of interlocked dependencies only if they are nested, Swiss-German cannot be context-free. On the same lines, (Postal and Langendoen, 1984) observed that English was not context-free. In view of the fact that the aforesaid constructions were seen to extend to almost all other natural languages, it became imperative to design grammars formalisms that could be suitable for doing linguistics and also be in the vicinity of context-free grammars along with preserving its computational tractability.

Joshi (1985) described a list of properties that an extension of the context-free languages should have if it is to be of practical use for linguistics: It should contain all context-free languages, it can describe a limited number of types of cross-serial dependencies, its membership problem has polynomial complexity, and all languages in it have constant growth property. He called classes of languages having these properties mildly context-sensitive as they extend the context-free languages and slightly absorbing some context-sensitive languages. In the literature, the classes of mutually equivalent formalisms satisfying the said properties are the TAG-languages (TAG abbreviating Tree Adjoining Grammar) and the MG-languages (MG abbreviating Minimalist Grammar). TAG-languages consist of Tree Adjoining Grammars (TAGs), Combinatory Categorical Grammars (CCGs), Linear Indexed Grammars (LIGs), and Head Grammars (HGs). Joshi et al. (1990) observed that they were equivalent as they described the same class of languages. MG-languages include Linear Context-free Rewrite Systems and Set-Local Multi-Component TAGs, and the formalisation of Noam Chomsky's Minimalist Grammars. The membership prob-

lem for TAG-languages is $O(n^6)$. Non-context free languages that belong to the TAG-languages are $a^n b^m c^n d^m$, the copy language, $a^n b^n c^n, a^n b^n c^n d^n$, etc. Examples of MG-languages include $a_1^n \cdots a_k^n$ for arbitrary k, the k-copy language for any k(w^k for arbitrary k). Thus, from the examples above, TAG-languages may contain only up to four different types of interlocked unlimited (crossing or nesting) dependencies, but there is no such upper bound for MG-languages. This leads to a higher computational complexity of the membership problem for MG-languages. Becker et al. (1991) argued that this added complexity was actually needed to capture all aspects of string order variation in German which could be applicable to other natural languages. Currently, though investigations are still on, it is assumed that all natural languages were MG-languages.

4.2 Automata

Automata theory studies mathematical models of computing devices (or machines) and the class of problems that can be solved by such devices. An automaton is an abstract computing device that recognizes strings of a language.

It has a mechanism to read input, which is a string over a given alphabet, *process* it and *decide* whether the string is in the language or not. As such, akin to defining four classes of grammars for generating languages, four classes of machines can be defined for recognizing languages.
In this section, mainly following (Hopcroft and Ullman, 1969), (Ruohonen, 2009) and (Kari, 2013), details of automata theory are presented. We then introduced a new operation on regular languages and prove some related results.

4.2.1 Finite Automata (or Finite-State Accepters)

An automata is finite if it has a finite memory. The class of finite automaton is broadly divided into deterministic and non-deterministic finite types. In deterministic finite automata, if the *internal state*, *input* and *contents* of the storage are known, it is possible to predict the future behavior of the automaton, otherwise it is non-deterministic. At the onset, it may be noted that the regular languages are precisely those languages that can be recognized by finite automata (see (Kleen, 1952) for details).

Definition 4.2.1 Deterministic finite automaton (DFA)
A DFA is a quintuple (or 5–tuple) $M = (Q, \Sigma, q_0, \delta, A)$ where

$*$ Q is a finite set of states (at all times, the internal memory is in some state $q \in Q$);

$*$ Σ is the set of input symbols (the machine only operates on strings over the alphabet Σ);

$*$ q_0 is the initial state $(q_0 \in Q)$ of the machine;

$*$ δ is the (state) transition function which maps each pair $(q_i,\ a)$, where q_i is a state and a is an input symbol, to a unique next state q_j : $(q_i,\ a) = q_j$; i.e., $\delta : Q \times \Sigma \longrightarrow Q$; and

$*$ A is a *set of terminal (final or accepting) states* $(A \subseteq Q)$.

The string is accepted if the internal state of the machine, after reading the whole input, is some state of A, rejected otherwise.

As its input, the automaton M receives a string $u = a_1 \cdots a_n$ which it starts to read from the left. In the beginning, M is in its initial state q_0 reading the first symbol a_1 of u. The next state q_k is determined by the transition function: $\delta(q_0, a_1) = q_k$. If M is in state q_k reading the symbol a_m, its next state is $\delta(q_k, a_m)$. If the final state of M, after the last input symbol a_n is read, is a state of A, then M is said to accept u, otherwise u is rejected.

M accepts the empty input ε if the initial state q_0 is also a terminal state.

The language recognized by an automaton M, denoted $L(M)$, is the set of all strings accepted by M (Kari, 2013).

Example 4.2

Let $M = (Q, \Sigma, q_0, \delta,\ A)$ be a DFA where

$Q = \{q_0,\ q_1\}$, $\Sigma = \{a,\ b\}$, $q_0 = q_0$, $A = \{q_1\}$, and δ as given below:

	a	b
q_0	q_1	q_0
q_1	q_0	q_1

M accepts an input string $u = bababab$ as follows:

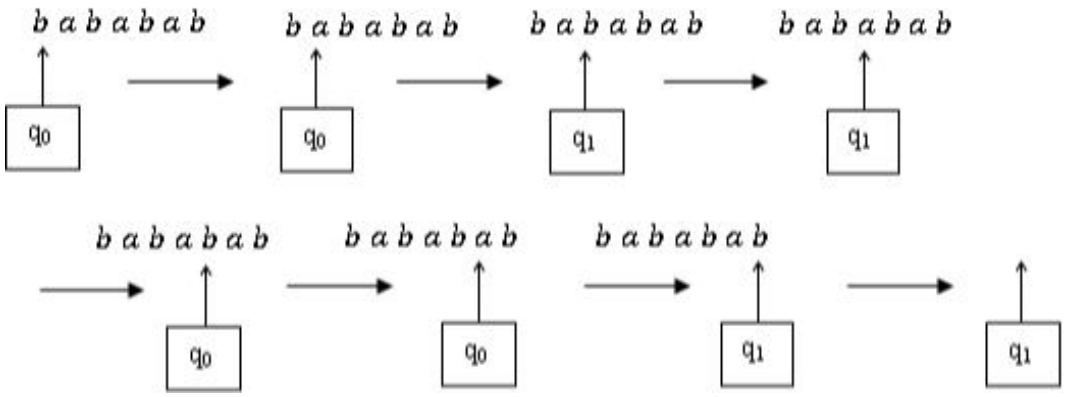


Figure 4.2.1: Example of a finite automaton accepting string

Alternatively, a finite automaton can be represented graphically called *transition diagram*: the vertices are the states; and the edges are the input symbols; final states are indicated by double circles, and the initial state is indicated by a short incoming arrow. For instance, the transition diagram for the DFA M considered above is as follows:

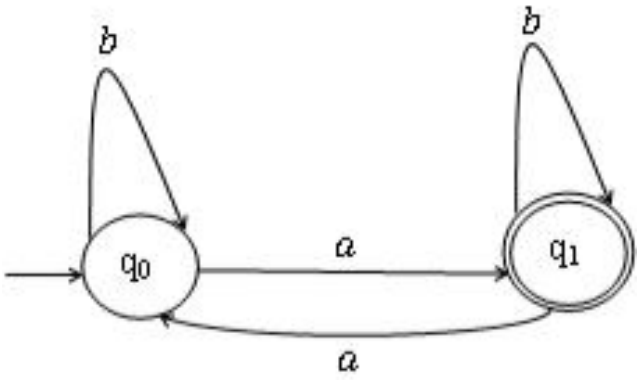


Figure 4.2.2: Transition diagram for an example of a DFA

In order to determine whether or not a string is accepted, one needs to start from the initial state, follow the path labeled with the input letters, and if the state where the path ends is a final state, the string is said to be *accepted*, otherwise *rejected*. Strings such as *abaa*, *bba*, *aaaaa*, *aaabb* and *baaa* are accepted while input strings *baba*, *baa* and *aba* are rejected (see (Kari, 2013) for details).

If the languages L_1 and L_2 are recognized by finite automata M_1 and M_2, respectively, then also the languages $L_1 \cup L_2$, $L_1 \cap L_2$ $L_1 - L_2, \overline{L_1}, L_1 L_2$, etc., are recognized by finite automata.

The proofs of the aforesaid results are outlined as follows:
Let L_1 and L_2 be chosen to share the same alphabet. If this is not the case the union of the original alphabets is taken as alphabet. It may further be assumed that the alphabet of the automata M_1 and M_2 is common.

A *product automaton* of M_1 and M_2 is constructed as follows:

If $M_1 = (Q, \Sigma, q_0, \delta, A)$ and $M_2 = (S, \Sigma\ , s_0, \gamma, B)$, then the product automaton is

$M_1 \times M_2 = (Q \times S, \Sigma, (q_0, s_0), \sigma, C)$ where the set C of terminal states is chosen accordingly. The set of states $Q \times S$ consists of all ordered pairs of states (q_i, s_j) where q_i is in Q and s_j is in S.
If $\delta(q_i, a) = q_k$ and $\gamma(s_j, a) = s_l$, then we define $\sigma((q_i, s_j), a) = (q_k, s_l)$.

Now, for recognizing $L_1 \cup L_2$, C is chosen to consist of exactly those pairs (q_i, s_j) where q_i is in A or/and s_j is in B i.e., at least one of the automata is in a terminal state after reading the input string.

On the other hand, for recognizing $L_1 \cap L_2$, C is chosen to consist of all pairs (q_i, s_j) where q_i is in A and s_j is in B, that is, both automata finish their reading in a terminal state. For recognizing $L_1 - L_2$, C is chosen to consist of those pairs (q_i, s_j) where q_i is in A and s_j is not in B, that is, M_1 finishes in a terminal state after reading the input string but M_2 does not.

Further, let M be a finite automaton for a language L, then an automaton recognizing the complement $\overline{L}$ is obtained from M simply by changing the set of terminal states to its complement.
For catenation the proof is straightforward.

Definition 4.2.2 Length Product
Let $L_1 = \{u_1, u_2, \dots\}$ and $L_2 = \{v_1, v_2, \dots\}$ be languages. Then, the length product of L_1 and L_2, denoted $L_1 \otimes L_2$, is defined as $L_1 \otimes L_2 = \{(u_i, v_j) | |u_i| = |v_j|,\ u_i \in L_1,\ v_j \in L_2\}$ where $|u_i|$ denotes the length of the string u_i and (u_i, v_j) represents an ordered pair.
Unlike Cartesian product, in order that the operation $\otimes$ to be defined, the strings needs to be of equal length.

Theorem 4.2.1 If the languages L_1 and L_2 are recognized by finite automata M_1 and M_2, respectively, then their length product $L_1 \otimes L_2$ is recognized by a finite

automaton, say M.

Proof

Let $M_1 = (Q, \Sigma_1, q_0, \delta, F)$ and $M_2 = (P, \Sigma_2, p_0, \beta, G)$ be two automata such that M_1 recognizes L_1 and M_2 recognizes L_2. Then, an automaton M, recognizing $L_1 \otimes L_2$, is constructed as follows:

$M = (R, \Sigma, \mathrm{r}_0, \mu, \mathrm{D})$ where $R = Q \times P$, $\Sigma = \Sigma_1 \times \Sigma_2$, $r_0 = (q_0, p_0)$, $D = F \times G$, and if $\delta(q_k, a) = q_r$ and $\beta(p_m, b) = p_s$, then μ is defined as $\mu((q_k, p_m), (a, b)) = (q_r, p_s)$ where $a \in \Sigma_1$, $b \in \Sigma_2, q'_k s \in Q, p'_m s \in P$.

Let us consider a miniature example to illustrate the process.

Example 4.3

Let $M_1 = (Q, \Sigma_1, q_0, \delta, F)$ with $Q = \{q_0, q_1, q_2\}$, $\Sigma_1 = \{a, b\}$, $q_0 = q_0$, $F = \{q_2\}$, and δ is given as below:

	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_0

The transition diagram for M_1 is the following:

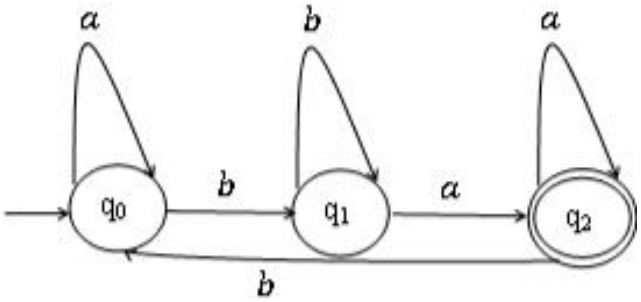


Figure 4.2.3: Transition diagram for an automaton M_1 used to construct an automaton for length product

Also, $L(M_1) = \{ba, aba, bba, \dots\}$.

Let $M_2 = (P, \Sigma_2, p_0, \beta, G)$ with $P = \{p_0, p_1\}$, $\Sigma_2 = \{0, 1\}$, $p_0 = p_0$, $G = \{p_1\}$, and β is given as below:

	0	1
p_0	p_0	p_1
p_1	p_1	p_1.

The transition diagram for M_2 is the following:

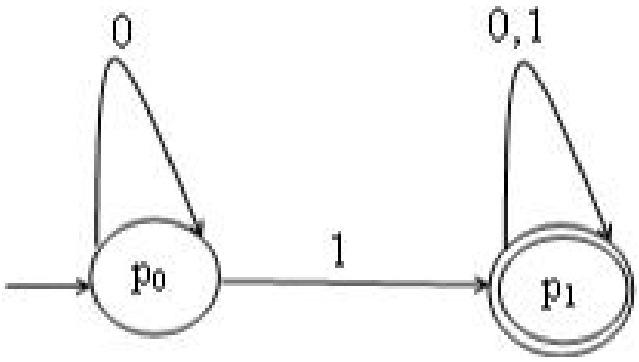


Figure 4.2.4: Transition diagram for an automaton M_2 used to construct an automaton for length product

Also, $L(M_2) = \{01, 011, 001, \dots\}$.

Now the automaton M recognizing $L_1 \otimes L_2$ is defined as

$M = (R, \Sigma, r_0, \mu, D)$ where $R = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1), (q_2, p_0), (q_2, p_1)\}$, $\Sigma = \{(a, 0), (a, 1), (b, 0), (b, 1)\}$, $r_0 = (q_0, p_0)$, $D = \{(q_2, p_1)\}$, and μ is constructed as

	$(a,0)$	$(a,1)$	$(b,0)$	$(b,1)$
(q_0,p_0)	(q_0,p_0)	(q_0,p_1)	(q_1,p_0)	(q_1,p_1)
(q_0,p_1)	(q_0,p_1)	(q_0,p_1)	(q_1,p_1)	(q_1,p_1)
(q_1,p_0)	(q_2,p_0)	(q_2,p_1)	(q_1,p_0)	(q_1,p_1)
(q_1,p_1)	(q_2,p_1)	(q_2,p_1)	(q_1,p_1)	(q_1,p_1)
(q_2,p_0)	(q_2,p_0)	(q_2,p_1)	(q_0,p_0)	(q_0,p_1)
(q_2,p_1)	(q_2,p_1)	(q_2,p_1)	(q_0,p_1)	(q_0,p_1).

The transition diagram for M is as follows:

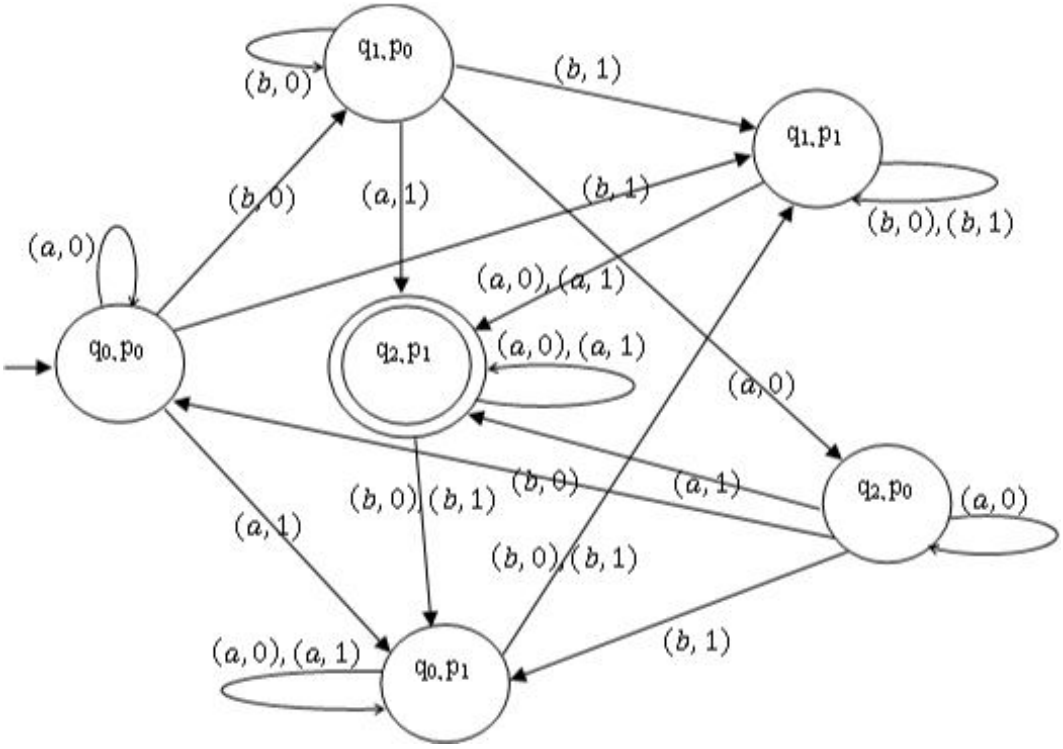


Figure 4.2.5: Transition diagram for an automaton M recognizing the length product of the languages L_1 and L_2

Also, $L(M) = \{(ba,01),(aba,011),(aba,001),(bba,011),(bba,001),\dots\}$.

Theorem 4.2.2 The length product is closed under union and intersection.

Proof

Let L_1, L_2, L_3 and L_4 be regular languages. Then, from above, $\overline{L_1}, \overline{L_2}, \overline{L_3}, \overline{L_4}$ are regular, and hence $\overline{L_1} \cup \overline{L_2}$ and $\overline{L_1} \cap \overline{L_2}$ are regular.

Moreover, applying De Morgan's laws, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, and thus $L_1 \cap L_2$ is regular, similarly for $L_3 \cap L_4$. Thus, $L_1 \cap L_2 \otimes L_3 \cap L_4$ is regular from theorem 4.2.1

Analogously, $L_1 \cup L_2 \otimes L_3 \cup L_4$ is also regular.

Definition 4.2.3 Nondeterministic finite automata (NFA)

The term *non-determinism* refers to availability of choices in state transitions. An NFA, unlike a DFA, allows all possible outgoing transitions that the input string could necessiate, however, some of them may lead to a non-final state. A string is accepted if there is at least one choice of transitions that takes the machine to a

final state.

An NFA is a quintuple (or 5–tuple) $M = (Q, \Sigma, q_0, \delta, A)$ where

Q, Σ, q_0 and A are as for DFA; and the transition function δ is defined as $\delta : Q \times \Sigma \longrightarrow 2^Q$, the power set of Q. That is, for each state q_i and an input letter a, the transition function maps each pair $(q_i,\ a)$ to exactly one subset (possible next states) of Q.

It is distinctive of an NFA that the function δ yields n set of states rather than a single state as is the case with a DFA.

Example 4.4

Let $M = (Q, \Sigma, q_0, \delta, A)$ be an NFA where $Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, A = \{q_2\}$ and δ given as below:

	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_1\}$	$\{q_1, q_2\}$
q_2	$\varnothing$	$\{q_2\}$.

The transition diagram for M is as follows:

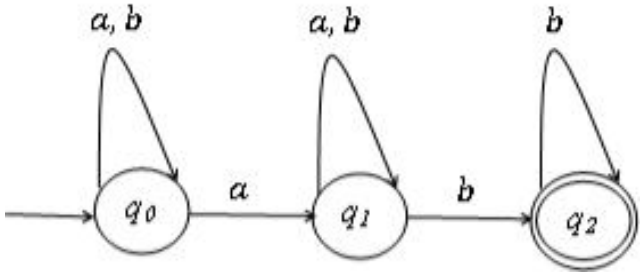
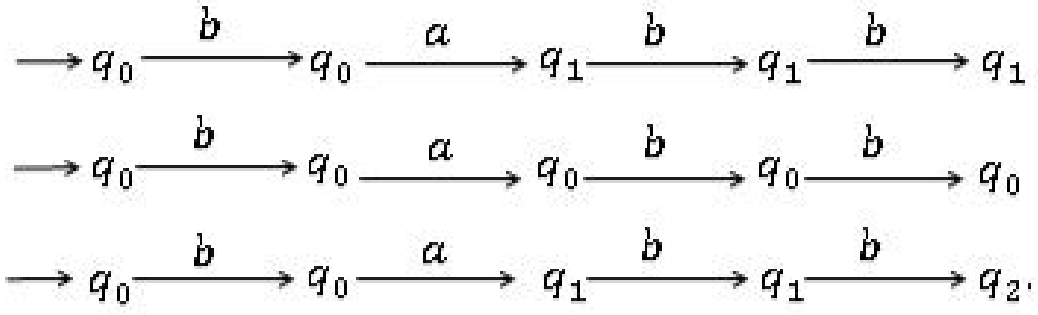


Figure 4.2.6: Transition diagram for an example of an NFA

From the diagram, there are two transitions from state q_0 with input letter a to states q_0 and q_1, and two transitions from q_1 with input letter b to q_1 and q_2. However, there is no transition from q_2 with input letter a.

Moreover, the input string $u = babb$ may follow the following choices of transitions:



Though the first two computation paths end in a non-final state, the string is still accepted as the last computation path takes the machine to an accepting state (see (Ruohonen, 2009) and (Kari, 2013) for details).

Definition 4.2.4 NFA with ε – moves

ε – *moves* are spontaneous transitions that result in a change of states without reaching any input letter and there is no restriction on its number i.e., any number of ε - moves are allowed.

An NFA M with ε–moves (ε–NFA, for short) is defined as $M = (Q, \Sigma, \delta, q_0, A)$ where Q, Σ, q_0, and A are the same as for an arbitrary NFA, and δ is a function:

$$Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^Q$$

that specifies for each q the transitions with all input letters a and ε.

1. $\delta(q, a)$ is the set of states p such that there is a move from q to p with input a, and

2. $\delta(q, \varepsilon)$ is the set of states p such that there is a *spontaneous move* from q to p (see (Ruohonen, 2009) and (Kari, 2013) for details).

Example 4.5

let $M = (Q, \Sigma, \delta, q_0, A)$ be an ε–NFA with $Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, A = \{q_1\}$, and δ as given below:

	a	b	ε
q_0	q_0	q_2	q_1
q_1	q_2	q_1	q_2
q_2	q_2	q_2	q_2.

The transition diagram for M is as follows:

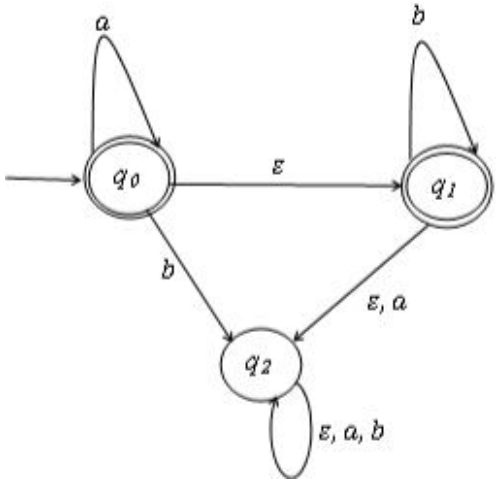


Figure 4.2.7: Transition diagram for an example of an ε–NFA

Also, $L(M) = \{a^n\ b^m \mid n,\ m \geq 0\}$.

As mentioned earlier, the languages defined by Finite automata (DFA, NFA, or ε–NFA) are regular (see (Kari, 2013) for details). Thus, in order to prove a language L regular, it is sufficient to show that one of the aforesaid discussed finite acceptors, regular grammar or regular expression defines it. Dually, if a language fails to be defined by a finite acceptor, regular grammar or regular expression, it is non-regular. As characteristic property of a regular language is that it must satisfy *pumping lemma*; however, the converse need not hold.

Theorem 4.2.3 (pumping lemma for regular languages) ((Kari, 2013) and (Maciel, 2014))
Let L be a regular language. Then there exists some positive constant n such that every string $x \in L$ of length at least n can be divided into three segments $x = uvw$

satisfying

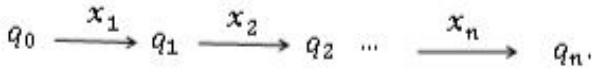
$$\begin{cases} |uv| \leq n, \text{and} \\ v \neq \varepsilon, \qquad (*) \end{cases}$$

and, for all $i \geq 0$, the string $uv^i w$ is in L.

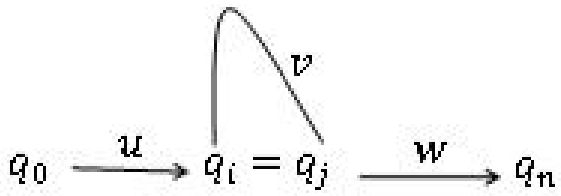
(i.e., $uw \in L, uvw \in L, uvvw \in L, uvvvw \in L$ etc.)

Proof

Let M be an NFA recognizing the language L, and n be the number of states of M. Consider a string $x = x_1 \dots x_n$. If $x \in L(M)$ then there exist a transition:



States $q_0, \dots, q_n$ can't be distinct as there are $n+1$ states. Therefore $q_i = q_j$ for some $0 \leq i < j \leq n$. Thus, the above transition can be written as



where $u = x_1 \ldots x_i,\ v = x_{i+1} \ldots x_j, w = x_{j+1} \ldots x_n$. Clearly, $|v| = j - i \geq 1$ and $|uv| = j \leq n$. Moreover, for all i, $uv^i w$ takes us from q_0 to q_i, then i–times round the loop from q_i to q_i, and then q_i to q_n. That is, for all $i \geq 0, uv^i w \in L$.

For example, let $L = \{a^m\ b^n \mid m \leq n\}$ be a regular language over an alphabet $\Sigma = \{a, b\}$. Let k be the positive constant of the pumping lemma. Consider $x = a^k b^k$. Clearly, $x \in L$ and $|x| \geq k$. Then, by the Pumping Lemma, x can be written as $x = uvw$ with $|uv| \leq k,\ v \neq \varepsilon$ and $uv^i w \in L$ for all i. Note that $|uv| \leq k$ implies v contains only $a's$, and $v \neq \varepsilon$ implies v contains at least one a. Therefore, for $i = 2$, $uv^2 w$ is not in L since it contains more $a's$ than $b's$, a contradiction and hence L is not regular.

It is an important aspect of pumping lemma that a language which does not satisfy it cannot be regular, and hence, none of regular grammars or finite automata needs to be applied.

4.2.2 Pushdown automaton

A *pushdown automaton* (PDA) is a computing device that employs a *stack*. Given an input signal, current state, and stack symbol, the automaton can follow a transition to another state and manipulates (push or pop) the stack.

In other words, a pushdown automaton (PDA) is a non-deterministic finite state automaton that has access to an infinite memory, organized as a stack.

Definition 4.2.5 Pushdown Automaton (PDA)

A PDA M is a septuple (or 7–tuple) $M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, A)$ where

$*$ Q is a finite set of states;

$*$ Σ is finite set of input alphabet;

$*$ Γ is the finite set of stack alphabet;

$*$ δ is the transition function mapping $Q \times(\Sigma \cup \{\varepsilon\}) \times\Gamma$ into finite subsets of $Q \times\Gamma^*$;

$*$ $q_o \in Q$ is the initial state;

$*$ $Z_o \in \Gamma$ is the start symbol of the stack; and

$*$ $A \subseteq Q$ is the set of final (accepting) states.

Definition 4.2.6 Instantaneous description (ID)

Essentially, an ID consists of all relevant information that is needed in subsequent steps of the computation. In other words, it is a device to record the configuration of a PDA at any point of time. It is a triple $(q,\ u,\ \alpha)$ where q is the state of the PDA, u the remaining input i.e., the suffix of the original input that has not been used yet, and α the content of the stack. Some standard configurations are the following:

$(q_1,\ u_1,\ \alpha_1) \vdash (q_2,\ u_2,\ \alpha_2)$ if there exists a move that takes the first ID into the second ID, $(q_1,\ u_1,\ \alpha_1) \vdash^* (q_2,\ u_2,\ \alpha_2)$ if there is a sequence of moves (possibly empty) from the first ID to the second ID, and

$(q_1,\ u_1,\ \alpha_1) \vdash^n (q_2,\ u_2,\ \alpha_2)$ if the first ID becomes the second ID in exactly n moves.

There are two modes of acceptance in PDA: *acceptance by empty stack* and *acceptance by final state.*

In acceptance by empty stack, a string is accepted, if and only if, all input letters are consumed and the stack becomes empty i.e., a PDA M accepts a string $u \in \Sigma^*$ by empty stack, if and only if, $(q_0,\ u,\ Z_0) \vdash^* (q, \varepsilon, \varepsilon)$ for some $q \in Q$, where q can be any state, including final. The language $L(M)$, recognized in this case, is given by $L(M)$:

$$L(M) = \{u \in \Sigma^* \mid \exists\ q \in Q :\ (q_0,\ u,\ Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}.$$

In the case of acceptance by final state, a string is accepted, if and only if, after reading all input letters the PDA is in an accepting state i.e., a string $u \in \Sigma^*$ is accepted by PDA M by final state, if and only if, $(q_0,\ u,\ Z_0)\ \vdash^*\ (q, \varepsilon, \alpha)$ for some $q \in A$ and $\alpha \in \Gamma^*$, q has to be a final state, but the stack does not need to be empty. The language $L(M)$, recognized in this case, is given by
$L(M)\ =\ \{u \in \Sigma^*\ |\exists\ q \in A, \alpha \in \Gamma^* :\ (q_0,\ u,\ Z_0)\ \vdash^*\ (q, \varepsilon, \alpha)\}$.

The two possible modes of acceptance described above are equivalent (see (Kari, 2013) for details).

Example 4.6

Consider the PDA $M\ =\ (\{q_0,\ q_1\},\ \{a,\ b\},\ \{Z_o,\ A\}, \delta,\ q_0, Z_0,\ \varnothing)$ where δ is given as

$\delta(q_0,\ a,\ Z_0)\ =\ \{(q_0,\ AZ_0)\}$, $\delta(q_0,\ a,\ A)\ =\ \{(q_0,\ AA)\}$, $\delta(q_0,\ b,\ A)\ =\ \{(q_1, \varepsilon)\}$, $\delta(q_1,\ b,\ A)\ =\ \{(q_1, \varepsilon)\}$, $\delta(q_1, \varepsilon,\ Z_0)\ =\ \{(q_1, \varepsilon)\}$, and all other sets $\delta(q,\ a,\ Z)$ are empty.

The string *aaaabbbb* is accepted as follows:
$(q_0,\ aaaabbbb, Z_o)\ \vdash (q_0,\ aaabbbb,\ AZ_o)\ \vdash (q_0,\ aabbbb,\ AAZ_o)\ \vdash (q_0,\ abbbb,\ AAAZ_o) \vdash$
$(q_0,\ bbbb,\ AAAAZ_o) \vdash (q_1,\ bbb,\ AAAZ_o)\ \vdash (q_1,\ bb,\ AAZ_o)\ \vdash (q_1,\ b,\ AZ_o)\ \vdash$
$(q_1, \varepsilon,\ Z_o)\ \vdash (q_1, \varepsilon, \varepsilon)$.
Note that it does not matter which set is chosen as the set of final states, since acceptance by empty stack is applied.

Push down automata define context-free languages (CFL). Similar to regular languages, a context-free language must satisfy the pumping lemma, however, the latter is not a sufficient condition to prove that a language is context-free.

Theorem 4.2.4 (Pumping lemma for CFL) ((Power, 2002) and (Kari, 2013))
Let L be a context-free language. Then there exists a positive number n such that every string $z \in L$ of length at least n can be divided into five segments $z = uvwxy$ such that

$$\begin{cases} |uwx| \leq n, \text{and} \\ v \neq \varepsilon, \text{ or } x \neq \varepsilon, \end{cases} \qquad (**)$$

and for all $i \geq 0$ the string $uv^iwx^iy \in L$.

Proof

Let z be a context-free language i.e., for a positive number n, $z \in L$ can be written as $z = uvwxy$ satisfying $(**)$. Observe that the string z can be generated by the grammar $G = (\{u, v, w, x, y\}, \{A\}, S, P)$ where P has the following derivations

$$S \Longrightarrow uAy, \quad A \Longrightarrow vAx, \quad A \Longrightarrow w.$$

Clearly, the derivation $A \Longrightarrow vAx$ may be repeated i times:

$$S \Longrightarrow uAy \Longrightarrow uvAxy \Longrightarrow uvvAxxy \Longrightarrow \ ... \Longrightarrow uv^iAx^iy \Longrightarrow uv^iwx^iy \ .$$

Thus $uv^iwx^iy \in L$ for every $i \geq 0$.

Example 4.7

We wish to show that the language $L = \{a^mb^mc^m \mid m \geq 1\}$ is not context-free.

Let L be context-free, and let k be a positive number associated with L by the pumping lemma. Let us choose the string $z = a^kb^kc^k$ from the language L. Let $z = uvwxy$ such that $|vwx| \leq k$ and $vx \neq \varepsilon$. Then for all $i \geq 0$, z can be represented as uv^iwx^iy.

Let $i = 2$, then v or x cannot contain a mixture of letters from $\{a, b, c\}$, otherwise

they would be in the wrong order for uv^2wx^2y; and if v or x contain just $a's$, $b's$ or $c's$ then uv^2wx^2y cannot maintain the balance between the three letters.

Hence, the choice $i = 2$ leads to $uv^iwx^iy \notin L$. Therefore, the language $L = \{a^mb^mc^m \mid m \geq 1\}$ is not context-free.

Mirror language (i.e., the set of strings wu over a given alphabet such that u is the mirror image of w), *Palindrome* language (i.e., the set of strings u that are identical to their mirror image), $\{a^nb^n \mid n \geq 1\}$, well-formed programs of *python* (or any other high level programming language), and well-formed arithmetic expressions are some examples of (non-regular) context-free languages (see (Jäger and Rogers, 2012) for details).

4.2.3 Linear bounded automata

In pushdown automaton, reading and writing (i.e., replacing a symbol by another one) take place on top of the stack, there is no direct access to other parts of the stack. If reading and writing throughout the stack, and reading the input everywhere all the time, are allowed, the resulting automation is called a *linear bounded automaton* (LBA).

Definition 4.2.7 Linear Bounded Automata (LBA)

A LBA is an octuple (or 8–tuple) $M = (Q, \Sigma, \Gamma, q_0, X_L, X_R, \delta, A)$ where

* Q is a finite set of states;

* Σ is the input alphabet (alphabet of the language);

* Γ is the tape alphabet $(\Sigma \subset \Gamma)$;

* $q_0 \in Q$ is the initial state;

* $X_L \in \Gamma - \Sigma$ is the left endmarker;

* $X_R \in \Gamma - \Sigma$ is the right endmarker;

* $(A \in Q)$ is set of terminal states; and

* δ is the (state) transition function which maps each pair $Q\backslash\{A\} \times \Gamma$ to a set $Q \times \Gamma \times \{L, R\}$. The transition $(q_j, Y, D) \in \delta(q_i, X)$ implies that LBA M is in state q_i reading a tape cell containing X, M changes its state to q_j, replaces X by Y in that cell and move in the direction of D ($D = L$ for left, and $D = R$ for right).

The memory contents are given using configurations (or Instantaneous Descriptions). A *configuration* is a quadruple (q_i, α, X, β) where q_i is a state, $\alpha X \beta$ is the tape, and X is the tape symbol under scan. A configuration $(q_j, \alpha', Y, \beta')$ is the direct successor of the configuration (q_i, α, X, β) if it is obtained from (q_i, α, X, β) by one transition denoted $(q_i, \alpha, X, \beta) \vdash (q_j, \alpha', Y, \beta')$.

In the beginning, the tape contains the input string u plus the endmarkers $(X_L u X_R)$, and the corresponding initial configuration is $(q_i, \varepsilon, X_L, uX_R)$ where q_i is an initial state. The LBA accepts the input u if $(q_i, \varepsilon, X_L, uX_R) \vdash^* (q_j, \alpha, Y, \beta)$ where q_i is an initial state and q_j is a terminal state. Note that an LBA halts after it finishes reading the input string and, the string is accepted if it halts in a terminal state, rejected if it halts in a non-terminal state (see (Hopcroft and Ullman, 1969) and (Ruohonen, 2009) for details).

Linear bounded automata define context-sensitive languages. Some examples of context-sensitive languages (that are not context-free) are as follows: the set of all prime numbers (where each number n is represented by a string of length n), the set of all square numbers, the copy language (the set of all strings over Σ that consist of two identical half's i.e., the set of strings ww over a given Σ such that w is an

arbitrary string of symbols over Σ, $a^n b^m c^n d^m$, $a^n b^n c^n$ and $a^n b^n c^n d^n e^n f^n$ (see (Jiang et al., 2010) and (Jäger and Rogers, 2012) for details).

4.2.4 Turing machines

In the mid-1930s and early 1940s, Alan Turing suggested certain automata as models for computing devices which today are known as Turing Machines. A Turing machine (TM) is like an LBA except that there are no endmarkers and the tape potentially extends infinitely in both directions. In the tape alphabet, there is a special symbol B, the blank. At each time, only finitely many tape symbols are different from the blank. In other words, despite the fact that a TM has an infinite memory, since its operation, similar to a finite-state Automaton (FSA), is controlled by a finite device, the size of the memory for any terminating computation must be finite. Note however, that there is a sharp divide between a FSA and a TM. The tape of a FSA is of fixed length, it can be processed in only one direction, tape cannot be read again, and no writing on the tape is allowed. On the other hand, a TM can operate either left or right on the tape, write intermediate and final outcomes on it, reinterpret any part of the tape, and most significantly, the existing tape can always be extended to an unlimited length simply by the addition of new cells to either end. It is known that any language that can be defined by a TM is computably enumerable and vice versa. The significance of a Turing model of computation lies in its capability of computing any partial recursive function and hence it includes all computations that any machine can perform and this is why it has come to be recognized as a general model for performing computations (see (Linz, 2001), (Hopcroft et al., 2001), (Jäger and Rogers, 2012) and (Maciel, 2014) for details).

Definition 4.2.8 Turing Machine

A Turing Machine is a septuple (or 7–tuple) $M = (Q, \Sigma, \Gamma, q_0, B, \delta, A)$ where

$*$ Q is a finite set of states;

$*$ Σ is the input alphabet (alphabet of the language);

$*$ Γ is the tape alphabet $(\Sigma \subset \Gamma)$;

$*$ $q_0 \in Q$ is the set of initial states;

$*$ $B \in \Gamma \backslash \Sigma$ is the blank symbol;

$*$ $(A \in Q)$ is set of terminal (accepting) states;

$*$ δ is the transition function which maps each pair $Q \backslash \{A\} \times \Gamma$ to a set $Q \times \Gamma \times \{L, R\}$.

The transition $(q_j, Y, L) \in \delta(q_i, X)$ imply that M is in state q_i reading tape symbol X, M changes its state to q_j, replaces X by Y on the tape, and move the R/W head one cell to the left. Similarly, $(q_j, Y, R) \in \delta(q_i, X)$ can be defined, but R/W head moves to the right, in this case. There are no restrictions for the movement of the read-write head since there are no endmarkers.

Initially the input string is written on the tape, flanked by blanks, and the read-write head reads the first symbol of the input. In case the input is empty, the read-write head reads one of the blanks. A Turing machine M halts if there is no possible move, if M halts in a final state, the input string is accepted, otherwise rejected. The string is also rejected if M never halts or halts in a state other than the terminal state. In other words, there are three possible outcomes of executing a Turing machine over a given input: it may halt and accept the input, or halts and rejects the input, or never halts i.e., enters an infinite loop.

The memory contents of the tape are given using configurations (or Instantaneous Descriptions). A configuration is a quadruple (q_i, α, X, β) where q_i is a state, $\alpha X \beta$ is the tape, and X is the tape symbol under scan. If the read-write head is

not reading a blank, then the corresponding configuration is $(q_i, \alpha, X, \ \beta)$ where q_i is a state, $\alpha X \ \beta$ is the longest substring in the tape not containing any blanks, and X is the tape symbol read by M in this substring. On the other hand, if the read-write head is reading a blank, then the corresponding configuration is $(q_i, \varepsilon, B, \ \beta)$ (resp. $(q_i, \alpha, B, \varepsilon)$) where β (resp. α) is the longest substring in the tape not containing blanks. In particular, if the tape contains only blanks i.e., the tape is empty, then the corresponding configuration is $(q_i, \varepsilon, B, \varepsilon)$.

Similar to an LBA, a configuration $(q_j, \alpha', Y, \ \beta')$ is the direct successor of the configuration $(q_i, \alpha, X, \ \beta)$ if it is obtained from $(q_i, \alpha, X, \ \beta)$ by one transition, denoted $(q_i, \alpha, X, \ \beta) \vdash (q_j, \alpha', Y, \ \beta')$. A Turing Machine accepts the input string if $(q_i, \alpha, X, \ \beta) \vdash^* (q_j, \alpha', Y, \ \beta')$ where q_i is an initial state and q_j the terminal state (see (Hopcroft et al., 2001), (Ruohonen, 2009) and (Kari, 2013) for details).

Example 4.8

A Turing Machine $M = (\{q_0, q_F, q_L, q_a, q_a', q_b, q_b'\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, q_F)$, where δ is as defined below, recognizes palindrome languages over $\{a, b\}$.

The transition function δ is defined as follows:

$\delta(q_0, B) = (q_F, B, R)$: Remaining input is empty

$\delta(q_0, a) = (q_a, B, R)$: Erase first input letter a

$\delta(q_a, a) = (q_a, a, R)$

$\delta(q_a, b) = (q_a, b, R)$

$\delta(q_a, B) = (q_a', B, L)$: End of input found

$\delta(q_a', B) = (q_F, B, R)$: Remaining input was empty

$\delta(q_a', a) = (q_L, B, L)$: Erase last input letter a

$\delta(q_0, b) = (q_b, B, R)$: Erase first input letter b

$\delta(q_b, a) = (q_b, a, R)$

$\delta(q_b, b) = (q_b, b, R)$

$\delta(q_b, B) = (q_b^{'}, B, L)$: End of input found

$\delta(q_b^{'}, B) = (q_F, B, R)$: Remaining input was empty

$\delta(q_b^{'}, b) = (q_L, B, L)$: Erase last input letter b

$\delta(q_L, a) = (q_L, a, L)$: Move back to beginning

$\delta(q_L, b) = (q_L, b, L)$

$\delta(q_L, B) = (q_0, B, R)$: Beginning of input found.

The string $aabbaa$ is accepted as follows:

$q_0aabbaa \vdash q_a\ abbaa \vdash a\ q_a\ bbaa \vdash ab\ q_a\ baa \vdash abb\ q_a\ aa \vdash abba\ q_a\ a \vdash abbaa\ q_a\ B \vdash abba\ q_a^{'}\ a \vdash abb\ q_L\ a \vdash q_L\ Babba \vdash q_0abba \vdash q_a\ bba \vdash b\ q_a\ ba \vdash bb\ q_a\ a \vdash bba\ q_a\ B \vdash bb\ q_a^{'}\ a \vdash b\ q_L\ b \vdash q_L\ bb \vdash q_L\ Bbb \vdash q_0\ bb \vdash q_b\ b \vdash b\ q_b\ B \vdash q_b^{'}\ b \vdash q_L\ B \vdash q_0\ B \vdash q_F\ B$ (see (Kari, 2013) for details).

Similar to unrestricted grammars, Turing machines can define all computably enumerable languages i.e., a language defined by some formal grammar. Equivalently, Type 0 grammars can express any language that can be accepted by a Turing Machine. Examples of computably enumerable languages are a set of computer programs that halt after a finite number of steps. A programmer simply compiles the program into a Turing machine and let it run, and then the output is *yes* if the program terminates. It is important, however, to note that computably enumerable languages are *semi-decidable* i.e., a TM that takes a string w as input and outputs the answer *yes* if and only if w is generated by a grammar G, and if w is not generated by G, the machine either outputs a different answer or it runs for ever. A Turing machine can systematically list all proofs of theorems one after the other, if

the last line of the proof equals the string in question, the output is *yes*; otherwise it moves on to the next proof (see (Jäger and Rogers, 2012) for details).

CHAPTER FIVE

FORMAL MULTILANGUAGES AND THEIR ALGEBRA

In this section, the two approaches to multisets viz., functional approach and equivalence relation approach are studied. The mathematics of *sorts* is developed and applied to modify and extend the extant theory of multisets. Finally, the concept of multilanguages and some related algebra are formulated.

5.1 Multisets

Intuitively, a *multiset* (*mset*, for short) is an unordered collection of objects in which, unlike an ordinary set, duplicates or multiples of objects are admitted. Each individual occurrence of an object in an mset is called its *elements*. All copies of an object in an mset are indistinguishable. The objects of an mset are the distinguishable elements.

In most of the known application areas, objects are allowed to repeat in an mset finitely, albeit in a theoretical development, infinite multiplicities of objects are also dealt with (see (Blizard, 1989), (Monro, 1987), (Singh et al., 2007) for details).

As emphasized in (Monro, 1987) and (Syropoulos, 2003), collections admitting objects with finite multiplicities could be viewed in two different ways: Sets with distinguishable repeated elements (e.g., people or vehicles sharing a common property), called *multisets*; and sets with indistinguishable repeated elements (e.g., soup of elementary particles), called *multinumbers*. However, in the literature of multisets, Monro's terminology is usually reversed. Syropoulos (2003), in particular, calls the

former *real multiset* and the latter *multiset*. In the literature, the two notions viz., *real multisets* and *multisets* are, respectively, formalized as a *set equipped with an equivalence relation* or *with a function*, which we referred as *multisets of equivalence relation approach* and *multisets of functional approach*, respectively.

5.1.1 Multisets of Functional Approach

Formally, an mset A over D is a *cardinal-valued* function i.e., $A : D \longrightarrow \aleph = \{0, 1, 2, \cdots\}$, also called a *generalized characteristic function*, where D is a set. Let $A(x), M_A(x)$ or C_x denote the number of times an object x occurs in A. The set D is called the *ground* or *generic* set of the class of all msets containing objects from D. The cardinality of an mset A is the sum of the multiplicities of all its objects, denoted $C(A)$ or $|A|$. The set of all objects of an mset is called its *root, support* or *carrier* set, annoted A^*. The cardinality of the root set of an mset is called its *dimension*. In general, if x_1 appears k_1 times, x_2 appears k_2 times,$\cdots, x_n$ appears k_n times in an mset A, then A is expressed as $A = [k_1/x_1, k_2/x_2, \cdots, k_n/x_n]$ or $A = [x_1^{k_1}, x_2^{k_2}, \cdots, x_n^{k_n}]$ or $A = [x_1^{k_1} x_2^{k_2} \cdots x_n^{k_n}]$. Moreover, for convenience, the curly brackets are also used in place of the square brackets. In fact the last form of representation as string, even without using any bracket, turns out to be the most compact one, especially in computational parlance.

Moreover, the *cardinality bounded multiset space* X^n, grounded on a finite set X, contains all msets built up on X such that no element of X appears in an mset $A \in X^n$ with multiplicity $> n$ and, for the maximum mset $\mathbb{Z} \in X^n$, which always exists, $m_{\mathbb{Z}}(x) = max\ m_A(x), \forall A \in X^n, \forall x \in A$ (see (Singh et al., 2011) for details).

An mset is called *regular* or *constant* if all its objects occur with the same multiplicity and the common multiplicity is called its *height*. For example, $[a, b]_{3,3}$ is a regular mset of height 3. An mset is called *simple* if all its elements are the same. For example, $[a]_3$ is a simple mset. It follows that the root set of a simple mset is

a singleton. The mset for any ground set D is called *empty*, denoted $\varnothing$ or [ ], if $m_{\varnothing}(x) = 0$ for all $x \in D$.

An mset A is called a *submset* of another mset B, denoted $A \subseteq B$, if $m_A(x) \leq m_B(x)$ for all $x \in D$.

The *union* of two msets A and B, denoted $A \cup B$, is the mset C defined by $m_C(x) = max\{m_A(x), m_B(x)\}$. That is, C contains both A and B.

The *intersection* of the msets A and B, denoted $A \cap B$, is the mset C defined by $m_C(x) = min\{m_A(x), m_B(x)\}$. That is, C is contained in both A and B.

The *addition* of msets A and B, denoted $A + B$ or $A \uplus B$, is the mset C defined by $m_C(x) = m_A(x) + m_B(x)$. However, in a cardinality bounded mset space X^n, this definition needs to be modified as follows: $m_{A+B}(x) = min\{n, m_A(x) + m_B(x)\}$.

The *arithmetic difference* or *subtraction* of B from A where $B \subseteq A$, denoted $A - B$, is the mset $C \subseteq A$ such that $m_C(x) = m_A(x) - m_B(x)$, for all objects x. In general, $m_C(x) = m_A(x) - m_{A \cap B}(x) = max\{m_A(x) - m_B(x), 0\}$, for all objects x. It is also called the *relative complement* of B in A. Moreover, even if B is not contained in A, this definition holds good. In multiset context, an element may occur both in an mset and in its complement. Accordingly, some of the consequences of the aforesaid definition may not satisfy the laws of classical set theory. For example, if $A = [a, b]_{4,5}$ and $B = [a, b]_{2,3}$ then $A - B = [a, b]_{2,2} \subset B$, contradicting the classical law: $(A - B) \cap B = \varnothing$ (see (Singh et al., 2011) for further details).

The functional approach has received wider attention than the equivalence relation approach (see (Singh et al., 2007), (Singh et al., 2008),(Blizard, 1989), (Girish and John, 2009), (Yager, 1986), (Dershowitz and Manna, 1979), (Wildberger, 2003) for example). Having this in mind, we shall not dwell upon describing further details of functional approach.

5.1.2 Multisets of Equivalence Relation Approach

Weyl (1959) was the first to formulate equivalence relation approach to describe multisets and applied it to a variety of problems in Physics, Chemistry and Genetics. An aggregate (multiset) is defined as a set of elements with an equivalence relation.

Monro (1987) is a seminal exposition on expounding mathematics of multiset exploiting equivalence relation approach and viewed that some operations of functional approach has no counter parts in equivalence relation approach.

In the following, by explicating the concept of *sorts*, a modified and extended version of the theory of multisets developed in (Monro, 1987) is presented.

Definition 5.1.1 Multisets

Monro (1987) observes that an mset $[a, a, b]$ can be viewed as being of the form $[a, a', b]$, where a and a' are different objects of the same *sort* and b is of a different sort. Notationally, elements of distinct sorts are denoted by distinct letters and elements of the same sort by the same letter, with a sequence of dashes distinguishing different elements of that sort.

Formally, an mset A is defined as a pair (A_0, ρ), where A_0 is a set and ρ an equiva-

lence relation defined on A_0. The set A_0 is called the *field* of the multiset. In other words, an mset is a set of equivalence classes produced by the given equivalence relation. Elements of A_0 in the same equivalence class will be said to be of the same sort. The pair (A_0, ρ), where A_0 is a set and ρ the empty equivalence relation on A_0, is an ordinary set. Every multiset in equivalence relation approach has an associated multiset in functional approach, for example, the mset $[a, b, b', b'', c, c', e]$ has an associated mset $[a, b^3, c^2, e]$ in functional approach.

Definition 5.1.2 Sorts and Subsorts

Let the sort of an mset (A_0, ρ) be denoted by $s_1, s_2, \cdots, s_n$. Thus, an mset (A_0, ρ) can be equivalently represented as a set of all its sorts i.e., $(A_0, \rho) = \{s_1, s_2, \cdots s_n\}$. We shall call a sort S', a *subsort* of S, denoted $S' \subseteq S$, if and only if every element of S' is an element of S and, $|S'| \leq |S|$.

Equivalently, in this case, S will be called a *supersort* of S'. Further, two sorts S and S' will be called *equal* if and only if $S \subseteq S'$ and $S' \subseteq S$.

Let $|s_i|$ denote the cardinality of the sort s_i of an mset A. Then the cardinality of A, denoted $|A|$, can be given by $|A| = \sum |s_i|, \forall s_i \in A$. The maximum cardinality of a sort in an mset A is said to be the *height* of A, denoted $hgtA$.

For example, if $A = [a, a', a'', b, b', c, d, d', d'', d''', e]$, then $A = \{s_1, s_2, s_3, s_4, s_5\}$. Moreover, $|A| = 11$ and $hgtA = 4$.

An mset A is said to be *regular* or *constant* if $\forall s_1, \cdots, s_n \in A, |s_1| = |s_2| = \cdots = |s_n|$. For example, $[a, a', b, b']$ is a regular mset of height 2. The empty multiset is a constant mset with height 0. An mset A is said to be *simple* if it sort set is a singleton. For example, $[a, a', a'', a''']$ is a simple mset.

Definition 5.1.3 Multisubset (msubset or submset, for short)

In (Monro, 1987) a *submset* is defined as follows:

Let $A = (A_0, \rho)$ be an mset. A submultiset B of A, written $B \subseteq A$ or $A \supseteq B$, is a

multiset (B_0, σ) where B_0 is a subset of A_0 and σ is ρ-*restricted* to B_0. An mset is called the *parent in relation to its msubsets*. For example, $[a, b, b']$ is a submset of $[a, b, b', b'', c]$ and the latter is a parent mset of the former.

Equivalently, in terms of sorts, msubset of an mset can be more suitably defined as follows: B is a submset of A if and only if every sort of B is a subsort of some sort in A. By definition, $\varnothing$ is a submset of every mset.

A submset B of A is said to be *full* if for all $s_i \in A$, $\exists\, t_k \in B$ such that $|t_k| \leq |s_i|$. For example, $[a, b, b', c, c', c'']$ is a full msubset of $[a, a', b, b', b'', c, c', c'', c''']$.

Remark 5.1

It is easy to see that, similar to functional approach, $\subseteq$ is antisymmetiric i.e., $A \subseteq B$ and $B \subseteq A \Longrightarrow A = B$, and it is a partial ordering on the class of msets defined on a given domain.

Definition 5.1.4 Equal msets and Cognate msets

Similar to defining equality of msets in functional approach, we say that two msets A and B are *equal* or *the same*, written as $A = B$, if and only if $A \subseteq B$ and $B \subseteq A$.

Two msets A and B are said to be *cognate* or *similar* if and only if every sort of $A(B)$ is either a subsort or a supersort of $B(A)$.

Definition 5.1.5 Replete or whole submultiset

Let A be a multiset. A replete submultiset B of A is a submultiset of A such that if $a \in B$ and $a' \in A$ and a, a' are of the same sort, then $a' \in B$; that is, every sort of B is a sort of A. For example, let $A = [a, b, b', c, c', c'']$ then $B = [a, b, b']$ is a replete submultiset of A.

Definition 5.1.6 Disjoint Union

The disjoint union of msets $A = (A_0, \rho)$ and $B = (B_0, \sigma)$ is an mset $D = (D_0, \tau)$ with field $D_0 = A_0 \uplus B_0$ (disjoint union of A_0 and B_0) and the equivalence relation τ satisfying that for $a, b \in A_0 \uplus B_0, a\tau b$ if and only if either $a, b \in A_0$ and $a\rho b$ or $a, b \in B_0$ and $a\sigma b$. For example, let $A = [a, a', b, c, d, d'], B = [x, y, y', z, z']$, then $D = [a, a', b, c, d, d', x, y, y', z, z']$.

Remark 5.2

This definition, which is similar to the definition of *additive union*, denoted $\uplus$ in functional approach, is exactly the operation of *coproduct* defined in ((Monro, 1987), p. 173).

Definition 5.1.7 Union and Intersection

In (Monro, 1987) *Union and Intersection* are defined as follows:
Let $G = (G_0, \gamma), A = (A_0, \rho)$, and $B = (B_0, \sigma)$ be msets such that $A_0, B_0 \subseteq G_0$, and ρ, σ are γ-*restricted* to A_0 and B_0, respectively. The union of A and B, denoted $A \cup B$, is an mset $D = (D_0, \tau)$ with field $A_0 \cup B_0$ and the equivalence relation τ as γ-*restricted* to D_0. That is, $A \cup B$ is the smallest mset D which contains both A and B.

The intersection of A and B, denoted $A \cap B$, is an mset $D = (D_0, \tau)$ with field $A_0 \cap B_0$ and the equivalence relation τ as γ-*restricted* to D_0. That is, $A \cap B$ is the largest mset D contained in both A and B.

Equivalently, in terms of sorts, $A \cup B$ is the smallest set of sorts which contains both A and B, and $A \cap B$ is the largest set of sorts contained in both A and B. For example, let $G = [a, b, b', b'', c, c', c'', c''', d, d', e, e', e'', f, f'], A = [a, b, b', c, d], B = [b, b', c, c', d, d']$, then $A \cup B = [a, b, b', c, c', d, d'], A \cap B = [b, b', c, d]$.

Remark 5.3

As observed in (Monro, 1987), p. 175), in order that the union of two msets be *meaningful*, they need to be submsets of the same mset, in our observation, same applied to intersection.

Monro (1987), p. 176, stressed that *max , min* and *subtraction* operations, have no counter parts in equivalence relation approach. However, with our explication of sorts, the following are presented.

Definition 5.1.8 Maximum and Minimum

Let $G = (G_0, \gamma), A = (A_0, \rho)$, and $B = (B_0, \sigma)$ be msets such that $A_0, B_0 \subseteq G_0$, and ρ, σ are γ-restricted to A_0 and B_0, respectively. Let $r_1, r_2, \cdots, r_n$; $s_1, s_2, \cdots, s_m$; and $t_1, t_2, \cdots, t_k$ be the sorts of G, A, and B, respectively. Clearly, each s_m or t_k is either an r_n or a subsort of r_n, and $s_m \subseteq t_k$ or $t_k \subseteq s_m$ or none of these, $\forall n, m, k$. We define

(i) $max[A, B] = max[|s_m|, |t_k|]$, and

(ii) $min[A, B] = min[|s_m|, |t_k|]$, for $s_m \in A, t_k \in B, \forall m, k$.

Definition 5.1.9 Difference

$G - A = [r_n - s_m]$, for $s_m \in A, r_n \in G, \forall n, m$.

For example, let $G = [a, a', a'', b, b', c, c', c'', d, d', d'', d''', e, e', f, f']$, $A = [a, b, c, c', c'']$, $B = [a, a', c, c', f, f']$, then $max[A, B] = [a, a', b, c, c', c'', f, f']$, $min[A, B] = [a, c, c']$, and

$G - A = [a', a'', b', d, d', d'', d''', e, e', f, f']$, $(G - A) \cup A = [a', a'', b', d, d', d'', d''', e, e', f, f'] \cup [a, b, c, c', c''] = [a, a', a'', b, b', c, c', c'', d, d', d'', d''', e, e', f, f'] = G$,

$(G - A) \cap A = [a', a'', b', d, d', d'', d''', e, e', f, f'] \cap [a, b, c, c', c''] = \varnothing$

It is pleasant to see that $(G - A) \cup A = G, (G - A) \cap A = \varnothing$, which do not hold in functional approach. That is, Demorgan's laws, unlike in functional approach, hold in equivalence relation approach.

Remark 5.4

In view of the aforesaid explication of sorts, it follows that Monro's observation ((Monro, 1987), p. 176) that *max , min* and *subtraction* operations, as defined in Definition 3.2, have no counter parts in equivalence relation approach, needs to be revamped.

Definition 5.1.10 Multiset function

In (Monro, 1987) *Multiset function* is defined as follows:

Let $A = (A_0, \rho)$ and $B = (B_0, \sigma)$ be msets. An mset function from A to B, written $f : A \longrightarrow B$, is a function $f : A_0 \longrightarrow B_0$ which respects sorts i.e., if $a, a' \in A_0$ and $a\rho a'$, then $f(a)\sigma f(a')$.

With applications of sorts the following are defined:

Let A and B be msets. A function $f : A \longrightarrow B$ is called a *constant* mset function if $\forall a \in r_n, f(a) = b \in q_k, \forall n, k$ i.e., $|ranf(a)| = 1$, where r_n, q_k are the sorts of A and B respectively. The function $I : A \longrightarrow A$ is called an *identity* mset function on A, denoted I_A, if $\forall a \in r_n, \forall n, I_A(a) = a$.

Let $A = (A_0, \rho)$ and $B = (B_0, \sigma)$ be two msets. Then

(i) The mset function $f : A \longrightarrow B$ is an *injection* (one-one) if

$f : A_0 \longrightarrow B_0$ is one-one, sort-preserving, and $|s_i| \leq |t_k|$ where s_i is a sort in A, t_k is the corresponding sort in B.

For example, $f : [a, a', b] \longrightarrow [c, c', e, e']$, defined by $f(a) = c', f(a') = c, f(b) = e'$ is an injection.

(ii) The mset function $f : A \longrightarrow B$ is a *surjection* (onto) if

$f : A_0 \longrightarrow B_0$ is onto, sort-preserving, and $|s_i| \geq |t_k|$.

For example, $f : [a, a', b] \longrightarrow [c, d]$, defined by $f(a) = f(a') = d, f(b) = c$ is a surjection.

(iii) The mset function $f : A \longrightarrow B$ is a *bijection* (one-one and onto) if

$f : A_0 \longrightarrow B_0$ is a bijection, sort-preserving, and $|s_i| = |t_k|$.

For example, $f : [a, a', b, b', b'', b'''] \longrightarrow [c, c', c'', c''', d, d']$, defined by

$f(a) = d', f(a') = d, f(b) = c'', f(b') = c, f(b'') = c''', f(b''') = c'$ is a bijection.

We choose to call the aforesaid conditions required to hold between the cardinalities of the sorts as the *consistency* criteria.

Similar to functional approach, certain results that hold for ordinary sets do not hold for multisets of equivalence relation approach as well e.g., multisets of equal cardinality need not have a bijection between them, and the Schröder-Bernstein theorem fails for multisets (see (Blizard, 1989), (Monro, 1987), (Singh et al., 2007) for details). For example, the msets $A = [a, a', b, b', b'', b'''], B = [c, c', c'', c''', c'^v, d]$, though both are of equal cardinality, but no bijection between them can exist, since the consistency criteria required to hold between the cardinalities of the sorts fails.

Definition 5.1.11 Multiset ordering

Let M(X) denote the class of finite msets over a totally ordered set X. Let $A, B \in M(X)$ and $r_1, r_2, \cdots, r_n$; $q_1, q_2, \cdots, q_m$ be the sorts of A and B, respectively. We define the List of A, denoted $List(A)$, as the list of r_i's such that $|r_1| \geq |r_2| \geq \cdots \geq |r_n|$ i.e., arranging the sorts of A in descending order of their cardinalities, and similarly for B, etc.

Then an ordering on M(X) can be defined as

A dominates B, written $A >> B$, if and only if $List(A) \stackrel{lex}{>} List(B)$.

That is, if $List(A)$ is lexicographically greater than $List(B)$.

For example, let $A = [a, b, b', b'', c, c', c'', c''', h, h', h'']$ and

$B = [a, a', a'', d, d', d'', d''', e, e', g, g', p, r, r']$. Then

$List(A) = [c, c', c'', c''', b, b', b'', h, h', h'', a]$ and

$List(B) = [d, d', d'', d''', a, a', a'', e, e', g, g', r, r', p]$.

Thus, $A >> B$ since $List(A)$ is lexicographically greater than $List(B)$.

It is important that this formulation of mset ordering is in consonance with Dershowitz-Manna ordering (see (Dershowitz and Manna, 1979) for details).

For, the above example can be represented as follows:

$ListA = [4, 3, 3, 1]$ and $ListB = [4, 3, 2, 2, 2, 1]$. Now, B can be obtained from A by removing 3, and replacing it by $[2, 2, 2]$.

Definition 5.1.12 Metrics and Metric spaces

Let $E = M(X)$ denote the class of all finite msets built on a set X. Let $A, B, C, \cdots \in E$. As described above, all msets $\in E$ are sets of sorts represented by their respective leading elements. One can define some suitable *measures* on E to enrich mathematics of multisets. For instance, $m(A) = |A|$ or $m(A) = hgtA$, etc. In the following, some metrics and metric spaces are formulated.

Let (distance) $d : E \times E \rightarrow \Re$, the set of real numbers, be defined as

$d(A, B) = \Sigma_{m,k} \, ||s_m| - |t_k||, \forall A, B \in E, s_m \in A, t_k \in B, \forall m, k.$

Theorem 5.1.1 *d is a pseudo-metric on E and (E, d) is a pseudo-metric space.*

Proof

Let us assume that the notions of metric and pseudo-metric are defined in the usual way. Clearly, for all $A, B, C \in E$, d satisfies the following properties:

$(M1)$ $d(A, B) \geq 0$

$(M2)$ $d(A, B) = d(B, A)$

$(M3)$ $d(A, B) \leq d(A, C) + d(C, B)$

$(M4)$ $d(A, A) = 0$.

$d(A, B) = 0$ if and only if $A = B$ does not hold. That is, $d(A, B) = 0$ does not necessarily imply that $A = B$; however, if $A = B$ implies $d(A, B) = 0$.

Thus d is a pseudo-metric on E and (E, d) is a pseudo-metric space.

Similarly, Let $d^{\circ} : E \times E \rightarrow \Re$ be defined as

$d^{\circ}(A, B) = max||s_m| - |t_k||, \forall A, B \in E, s_m \in A, t_k \in B, \forall m, k.$

Then d° satifies $(M1)$ through $(M4)$ and hence, it is a pseudo-metric and (E, d°) is a pseudo-metric space.

Similar results hold if $|A|$ is replaced by $hgtA$, etc.

Definition 5.1.13 Diameter of an mset and Distance between two msets

Let (E, d) be a metric space and let $A \in E$. Then the *diameter* of A, denoted $\delta(A)$, can be defined by

$\delta(A) = sup\{d(s_i, s_j) | s_i, s_j \in A, \forall i, j\}$, where $d(s_i, s_j) = ||s_i| - |s_j||$.

The *distance* between two msets $A, B \in E$, denoted $\rho(A, B)$, can be defined by $\rho(A, B) = inf\{d(s_m, t_k) | s_m \in A, t_k \in B, \forall m, k\}$ where $d(s_m, t_k) = ||s_m| - |t_k||$.

If A and B has at least a sort in common, implies $\rho(A, B) = 0$, but the converse need not hold. Also $|A|$ can be viewed as the distance $\rho(A, \varnothing)$ of A from the empty mset.

Remark 5.5

It is viewed that the equivalence relation approach, due to its set-like foundation, is found suitable to categorical models while the functional approach, owing to its numeric foundation, is found amenable to *hands-on applications*. In order to develop multilanguages, the functional approach is adopted.

5.2 Formal Multilanguages

Knuth, in the summer of 1964, while writing the book entitled *The Art of Computer Programming*, drafted a chapter on *The theory of languages* where he emphasized the need to allow strings to repeat in a language. In fact, (Knuth, 1992) is the first place

where an elaborate description of *formal multilanguages* or simply *multilanguages* appeared. Knuth defines a multilanguage simply as a multiset of strings.

5.3 Basic Definitions

The definitions of Alphabets and strings given for standard formal language hold good for multilanguages as well.

Over an alphabet $\sum$, let $\sum^{\odot}$ denote the multiset of all strings, including the empty string which is unique, and $\sum^{\oplus}$, the multiset of all non-empty strings. A multilanguage, denoted M_L, is an msubset of $\sum^{\odot}$.

Theorem 5.3.1 The multilanguages over any alphabet cannot be given in an infinite list as there are non-denumerable infinite amount of them.

Proof

Let $\sum$ be an alphabet and let M_L be a multilanguage such that $M_L = u_1, u_2, u_3, \cdots,$ ($u_1 = u_2$ allowed), where $u_1, u_2, u_3, \cdots,$ is the list of all strings over $\sum$.

Suppose $M_{L1}, M_{L2}, M_{L3}, \cdots,$ is the list of all multilanguages over $\sum$. Observe that M_L is not any of the $M'_{Li}s$, and the string u_i is in M_L if and only if it is not in M_{Li}. Contradicting the fact that both contained all the strings over $\sum$. Therefore the multilanguages over $\sum$ cannot be given in an infinite list.

5.4 Operations on Multilanguages

Operations on strings in standard formal language hold good for multilanguages as well.

5.4.1 Concatenation of Multilanguages

The *catenation* of the multilanguages M_{L1} and M_{L2} is $M_{L1}M_{L2} = [\ u_1u_2|\ \ u_1 \in M_{L1}$ and $\ u_2 \in M_{L2}]$.

Similar to standard formal languages, catenation of multilanguages is associative as catenation of strings is associative.
The n^{th} catenation of the multilanguage M_L is $M_L^n = [\ u_1 \dots\ u_n|\ u_1, \dots,\ u_n \in M_L]$. Also, $M_L^1 = M_L, \quad M_L^0 = \{\varepsilon\}$.

For any multilanguage $M_L, M_L\varnothing = \varnothing M_L = \varnothing$ and $M_L\{\varepsilon\} = \{\varepsilon\}M_L\ =\ M_L$ i.e., the multilanguages $\varnothing$ and $\{\varepsilon\}$ are zero and unit elements with respect to catenation.

For example, let $M_{L1}\ = [ab, b, b], \quad M_{L2}\ = [aa,\ aa, ba], M_{L3} = [u \in \sum^{\odot} |u$ has even number of $b's$ and strings with a sequence of two $b's$ repeat twice $]$, and $M_{L4} = [u|u$ starts with a b and the rest of the symbols are $a's]$ be multilanguages over an alphabet $\Sigma = \{a, b\}$, then $M_{L1}M_{L2} = [abaa, abaa, abba, baa, baa, bba, baa, baa, bba]$ and $M_{L3}M_{L4} = [u|\ u$ has an odd number of $b's]$.

5.4.2 Concatenation Closure of a Multilanguage

The catenation closure of a multilanguage M_L, denoted $M_L^{\odot}$, is defined as

$$M_L^{\odot} = \uplus_{i=0}^{\infty} M_L^i.$$

The positive closure $M_L^{\oplus}$ of M_L is

$$M_L^{\oplus} = \uplus_{i=1}^{\infty} M_L^i.$$

Remark 5.6

A string in $M_L^{\odot}$ and $M_L^{\oplus}$ repeat even if it does not repeat in M_L provided $\varepsilon \in M_L$. Moreover, if $\varepsilon \notin M_L$ the multiplicity of every element of $M_L^{\odot}$ and $M_L^{\oplus}$ will be finite, and infinite otherwise.

Similar to standard formal language, $M_L^{\odot} = M_L^{\oplus} \cup \{\varepsilon\}$, $M_L^{\odot} = M_L^{\oplus}$ iff $\varepsilon \in M_L$.

5.4.3 Union, Intersection, Difference and Complement

Let $|M_L|_u$ denote the number of occurrences of the string u in M_L. Let M_{L1} and M_{L2} be multilanguages over an alphabet Σ. Then,
(i) The union of M_{L1} and M_{L2}, denoted $M_{L1} \cup M_{L2}$, is the multilanguage $M_{L1} \cup M_{L2} = [u \in \Sigma^{\odot} | u = max(|M_{L1}|_u, |M_{L2}|_u)]$.

(ii) The intersection of M_{L1} and M_{L2}, denoted $M_{L1} \cap M_{L2}$, is the multilanguage $M_{L1} \cap M_{L2} = [u \in \Sigma^{\odot} | u = min(|M_{L1}|_u, |M_{L2}|_u)]$.

(iii) The difference (or *relative complement*) of M_{L1} and M_{L2}, denoted $M_{L1} - M_{L2}$, is the multilanguage $M_{L1} - M_{L2} = [u \in \Sigma^{\odot} | u = (|M_{L1}|_u - |M_{L2}|_u)]$.

(iv) The complement $\overline{M_L}$ of a multilanguage M_L is $\overline{M_L} = [u \in \Sigma^{\odot} | u \notin M_L]$.

5.5 Algebra of Multilanguages

5.5.1 Prefix, Suffix and Substring in a Multilanguage

Let $\sum$ be an alphabet, $\sum^{\odot}$ be the mset of all strings over $\sum$, and $u, v \in \sum^{\odot}$.

u is a *substring* of v iff there exist $x, y \in \sum^{\odot}$, such that $v = xuy$. In other words, a substring (also, called *segment*, *subword* or *factor*) of a string is any sequence of consecutive symbols that appear in the string. The term substrings of a string represents the mset of all substrings of a string.

u is a *prefix* of v iff there is some $y \in \sum^{\odot}$ such that $v = uy$. That is, a prefix of a string u is a substring of u that occurs at the beginning of u. The term prefixes of a string represents the mset of all prefixes of a string.

u is a *suffix* of v iff there is some $x \in \sum^{\odot}$ such that $v = xu$. That is, a suffix of a string u is a substring that occurs at the end of u. The term suffixes of a string represents the mset of all suffixes of a string.

Thus, Prefixes, suffixes and substrings are multilanguages.

A prefix is any sequence of leading symbols of the string, and a suffix is any sequence of trailing symbols of the string. Moreover, a substring of a string is a prefix of a suffix of the string, and also, a suffix of a prefix.

u is a proper prefix (suffix or substring) of v iff u is a prefix (respectively, suffix or substring) of v and $u \neq v$. Prefixes and suffixes both are substrings, but the converse need not hold.

For example, let $w = abaa$ be a string over $\sum = \{a, b\}$. Its prefixes, suffixes and substrings are as follows:

Prefixes: $[\varepsilon, a, ab, aba, abaa]$.

Suffixes: $[\varepsilon, a, aa, baa, abaa]$.

Substrings: $[\varepsilon, a, b, ab, a, ba, aba, a, aa, baa, abaa]$.

The border of a string is both its prefix and suffix, e.g., *bab* is a border of *babab*.

Remark 5.7

The prefixes and suffixes are defined in the same way both in standard formal language and multilanguage. For example, prefixes and suffixes of the string w considered above are the same in both. However, for substrings, this is not the case. The substrings of w in a multilanguage are as given above, where as in a standard formal language, substrings of w are $\{\varepsilon, a, b, ab, ba, aba, aa, baa, abaa\}$.

5.5.2 Lattices of Prefixes, Suffixes and Substrings of a String in a Multilanguage

Theorem 5.5.1 Multilanguages of prefixes, suffixes and substrings of a string form a bounded distributive lattice.

Proof

Let X be an mset of prefixes, suffixes or substrings of a string, and Y be the mset of all possible multilanguages of X. Let Z be a structure consisting of Y with union and intersection representing the (join) $\vee$ and (meet) $\wedge$ operations, respectively. Let $M_{L1}, M_{L2}, M_{L3} \in Y$. Since union of multilanguages is associative, commutative and idempotent, $M_{L1} \vee M_{L2} = M_{L2} \vee M_{L1}$, $M_{L1} \vee (M_{L2} \vee M_{L3}) = (M_{L1} \vee M_{L2}) \vee M_{L3}$ and $M_{L1} \vee M_{L1} = M_{L1}$ hold, Thus, $(Y, \vee)$ is a commutative, idempotent semigroup. Moreover, as the intersection of multilanguages is commutative, associative and idempotent, $(Y, \wedge)$ is a commutative, idempotent semigroup.

Also, $M_{L1} \vee (M_{L1} \wedge M_{L2}) = M_{L1}$ and $M_{L1} \wedge (M_{L1} \vee M_{L2}) = M_{L1}$, $\forall M_{L1}, M_{L2}, M_{L3} \in Y$ i.e., absorption laws hold. Thus, $Z = (Y, \vee, \wedge)$ is a lattice.

Moreover, as $\forall M_L \in Y, M_L \vee \varnothing = M_L$, $\varnothing$ serve as the identity element of the join

operation and, as $M_L \wedge Y = M_L$, Y serve as the identity of the meet operation. Then, Z is a bounded lattice.

Furthermore, as union of multilanguages distributes over intersection and intersection distributes over union, imply $\forall M_{L1}, M_{L2}, M_{L3} \in Y, M_{L1} \vee (M_{L2} \wedge M_{L3}) = (M_{L1} \vee M_{L2}) \wedge (M_{L1} \vee M_{L3})$ and $M_{L1} \wedge (M_{L2} \vee M_{L3}) = (M_{L1} \wedge M_{L2}) \vee (M_{L1} \wedge M_{L3})$ hold. Thus, Z is a bounded distributive lattice.

As an example, let $u = aba, v = aa$, where $u, v \in \Sigma^{\odot}$ over an alphabet $\Sigma = \{a, b\}$. Then the prefixes X of u are $[\varepsilon, a, ab, aba]$ and the mset Y of all possible multilanguages of X is $[\varnothing, [\varepsilon], [a], [ab], [aba], [\varepsilon, a], [\varepsilon, ab], [\varepsilon, aba], [a, ab], [a, aba], [ab, aba], [\varepsilon, a, ab], [\varepsilon, a, aba], [\varepsilon, ab, aba], [a, ab, aba], [\varepsilon, a, ab, aba]]$.

Let $M_{L1} = [ab], M_{L2} = [aba]$ then $[ab] \vee ([ab] \wedge [aba]) = [ab] \vee \varnothing = [ab]$ and $[ab] \wedge ([ab] \vee [aba]) = [ab] \wedge [[ab], [aba]] = [ab]$. Also, let $M_{L3} = \varnothing, M_{L4} = Y$, then $\varnothing \vee [ab] = [ab]$ and $[ab] \wedge Y = [ab]$. Similarly, results for distributivity could be computed. Thus, $(Z, \vee, \wedge,)$ is a bounded distributive lattice.

Another example, consider $v = aa$, the substrings X of v are $[\varepsilon, a, aa, a]$ and the mset Y of all possible multilanguages of X is $[\varnothing, [\varepsilon], [a], [aa], [a], [\varepsilon, a], [\varepsilon, aa], [\varepsilon, a], [a, aa], [a, a], [aa, a], [\varepsilon, a, aa], [\varepsilon, a, a], [\varepsilon, aa, a], [a, aa, a], [\varepsilon, a, aa, a]]$.

Let $M_{L1} = [a], M_{L2} = [a, a]$, then $[a] \vee ([a] \wedge [a, a]) = [a] \vee [a] = [a]$ and $[a] \wedge ([a] \vee [a, a]) = [a] \wedge [a, a] = [a]$. Also, let $M_{L3} = \varnothing, M_{L4} = Y$, then $\varnothing \vee [a, a] = [a, a]$ and $[a, a] \wedge Y = [a, a]$. Similarly, results for distributivity could be computed.
Thus, $(Z, \vee, \wedge,)$ is a bounded distributive lattice.

5.5.3 Prefix, Suffix and Substring Closures in a Multilanguage

Let M_L be a multilanguage. In line with the standard formal language, we define the the following for M_L:

Prefix closure of M_L :

$pref_{M_L}(v) = [u | v = uy; v \in M_L, u, y \in \sum^{\odot}]$, and

$pref(M_L) = \uplus_{v \in M_L} pref_{M_L}(v) = [u | v = uy; v \in M_L, u, y \in \sum^{\odot}]$.

Suffix closure of M_L :

$suff_{M_L}(v) = [u | v = xu; v \in M_L, x, u \in \sum^{\odot}]$, and

$suff(M_L) = \uplus_{v \in M_L} suff_{M_L}(v) = [u | v = xu; v \in M_L, x, u \in \sum^{\odot}]$.

Substring closure of M_L :

$subs_{M_L}(v) = [u | v = xuy; v \in M_L, x, u, y \in \sum^{\odot}]$, and

$subs(M_L) = \uplus_{v \in M_L} subs_{M_L}(v) = [u | v = xuy; v \in M_L, x, u, y \in \sum^{\odot}]$.

For example, let $M_L = [ab, ab, aba]$ over $\sum = \{a, b\}$. Then

$pref(M_L) = [\varepsilon, a, ab, a, ab, a, ab, aba]$,

$suff(M_L) = [\varepsilon, b, ab, b, ab, a, ba, aba]$, and

$subs(M_L) = [\varepsilon, a, b, ab, a, b, ab, a, b, ab, a, ba, aba]$.

As in a standard formal language L, the following results hold in M_L as well:

A multilanguage is said to be *prefix*, *suffix* or *substring closed* if $pref(M_L) = M_L, suff(M_L) = M_L$ or $subs(M_L) = M_L$, respectively. For example, $M_L = [\varepsilon, a, a, b]$ is both prefix, suffix and substring closed. The prefix, suffix and substring closure operators in a multilanguage M_L are idempotent:

$pref(pref(M_L)) = pref(M_L), suff(suff(M_L)) = suff(M_L), subs(subs(M_L)) = subs(M_L)$.

Remark 5.8

The prefix, suffix and substring closures of a multilanguage and that of a standard formal language differ. For example, in the case of a multilanguage $M_L = [ba, aba], suff(M_L) = [\varepsilon, a, ba, a, ba, aba]$, and in a standard formal language $L = \{ba, aba\}, suff(L) = \{\varepsilon, a, ba, aba\}$.

Definition 5.5.1 Superstring in a multilanguage
Given a multilanguage M_L of n strings over an alphabet Σ, a *superstring* of M_L is a single string that contains every string of M_L as a substring. Thus, catenation of the distinct strings of M_L in any order gives a trivial superstring of M_L.

For example, let $M_L = [0011, 0100, 01110, 0100, 01110]$ over $\Sigma = \{0, 1\}$.
Then 0011010001110, 01001110 and 001110100 are some of the superstrings of M_L.

5.5.4 Prefix, Suffix and Substring Relations in a Multilanguage

Let X consist of prefixes (suffixes or substrings) of a string $v \in \sum^{\odot}$ and R, be the respective binary relation on X. Hence forth, R shall stand for R (prefix, suffix or substring), unless otherwise stated.

i. R is *reflexive*: $\forall m/x \in X, m/xRm/x$. Since every prefix (suffix or substring) u is a prefix (respectively, suffix or substring) of itself.

ii. R (prefix or suffix) is *complete*: $\forall m/x \neq n/y \in X$, either $m/xRn/y$ or $n/yRm/x$. Since for every pair of prefixes (or suffixes) of a string v, one is a prefix (respectively, suffix) of the other. However, for substrings, this is not licit. For example, let $v = abbc$ and $m/x = 1/a, n/y = 1/c$ then both $m/x, n/y \in X$, but neither $m/xRn/y$ nor $n/yRm/x$.

iii. R is *antisymmetric*: $\forall m/x, n/y \in X$, if $m/xRn/y$ and $n/yRm/x$, then $m/x =$

n/y. Since whenever u is a prefix (suffix or substring) of v and v is a prefix (respectively, suffix or substring) of u, then u must be the same as v as duplicates are indistinguishable.

iv. R is *transitive*: $\forall m/x, n/y, k/z \in X$, if $m/xRn/y$ and $n/yRk/z$, then $m/xRk/z$. Since whenever u is a prefix (suffix or substring) of v and v is a prefix (respectively, suffix or substring) of w, clearly u is a prefix (respectively, suffix or substring) of w.

Clearly, R (prefix or suffix) is a linear mset order, where as R (substring) is only a partial mset order.

Remark 5.9

Prefixes (suffixes or substrings) of a string, denoted X and their respective binary relations, denoted R i.e., R on X, gives rise to a *multiset relational structure* (X, R), hence forth notated $\mathbb{X} = (X, R)$. In general, X is an mset and R is an mset relation. In particular, X may be a list of *attributes* defined on a domain and R, the relation name (see (Grefen and de By, 1994) for related descriptions).

Maximal, Minimal and Isolated Elements of $\mathbb{X} = (X, R)$

Let $\mathbb{X} = (X, R)$, where X consists of all nonempty prefixes (suffixes or substrings) of a string $v \in \sum^{\oplus}$ and R, its respective relation on X. Then,

- m/x is a *maximal* element of $\mathbb{X}$ iff $\forall n/y \in X, n/yRm/x$;
- For R (prefix or suffix), m/x is a *minimal* element of $\mathbb{X}$ iff $\forall n/y \in X, m/xRn/y$;

- For R (substring), m/x is a *minimal* element of $\mathbb{X}$ iff $\forall n/y \in X, m/xRn/y$; or else they are unrelated; and

- m/x is an isolated element of $\mathbb{X}$ iff m/x is both a maximal and minimal element of $\mathbb{X}$.

  Note that, an isolated element of a structure is obtained when a string of length 1 is considered.

For example, let $\sum = \{a, b\}, v = abba \in \sum^{\oplus}$, then the non-empty prefixes and substrings of v are $a, ab, abb, abba$ and $a, b, ab, b, bb, abb, a, ba, bba, abba$, respectively. Now for R (prefix), $1/a$ and $1/abba$ are the minimal and maximal elements, respectively, and for$R$ (substring), $2/a$ and $2/b$ are the minimal elements and $1/abba$ is the maximal element.
Also, let $u = a \in \sum^{\oplus}$, then a is the nonempty prefixes, suffixes and substrings of u and $1/a$ is the isolated element for R.

Theorem 5.5.2 $\mathbb{X} = (X, R)$ always has a unique maximal element and, only under R (prefix or suffix), it has also a unique minimal element.

Proof

Let $p/z \neq t/s \in X$, and let $p/z, t/s$ both be maximal elements of $\mathbb{X}$. Since p/z is a maximal element, $t/sRp/z$. Similarly, since t/s is a maximal element, $p/zRt/s$. Now, by antisymmetry of these relations, $p/z = t/s$. This contradicts the assumption that $p/z \neq t/s$. Hence $\mathbb{X}$ has a unique maximal element. Similarly, the other parts can be proved.

Chains of $\mathbb{X} = (X, R)$

A submset structure $\mathbb{C} = (C \subseteq X, R)$ of the prefix (suffix or substring) mset relational structure $\mathbb{X}$ is a chain in $\mathbb{X}$, if distinct points of C are pair wise comparable in X i.e., $\forall m/x \neq n/y \in X$, either $m/xRn/y$ or $n/yRm/x$ in X. Also, $\mathbb{X}$ itself is said

to be a chain if the distinct points on X are pair wise comparable. A chain is said to be maximal if no other chain strictly contains it.

It is easy to see that every (C, R) with R (prefix or suffix) is a chain, and the mset of all chains in a prefix (suffix or substring) relational structure is partially ordered by mset inclusion relation.

For example, let $v = aba \in \sum^{\oplus}$, and $X = [2/a, 1/b, 1/ab, 1/ba, 1/aba]$ be the mset of all nonempty substrings of v. Then,

submsets $C_1 = [2/a, 1/ab], C_2 = [2/a, 1/ba], C_3 = [2/a, 1/aba], C_4 = [2/a, 1/ab, 1/aba], C_5 = [2/a, 1/ba, 1/aba], C_6 = [1/ab, 1/aba], C_7 = [1/ba, 1/aba], C_8 = [1/b, 1/ab], C_9 = [1/b, 1/ba], C_{10} = [1/b, 1/aba], C_{11} = [1/b, 1/ab, 1/aba], C_{12} = [1/b, 1/ba, 1/aba]$ are chains in $\mathbb{X}$.

Moreover, C_4, C_5, C_{11} and C_{12} are the maximal chains.

Graphical Representation of R on X

Let X consist of all prefixes (suffixes or substrings) of a string $v \in \sum^{\oplus}$ and R, its respective relation on X. This relation can be represented by a directed graph $G = (V, E)$, where V is defined by $V = X$, and E by $(m/x, n/y) \in E$ iff $m/xRn/y$.

For example, Let $\sum = \{a, b\}, v = abaa \in \sum^{\oplus}$, and let the non-empty prefixes and substrings of v be given as:

Prefixes: $s = a, u = ab, w = aba, x = abaa$, and

Substrings: $r = a, s = b, t = ab, u = ba, w = aa, x = aba, y = baa, z = abaa$.

Then the directed graph for the prefix relation R on $X = [1/s, 1/u, 1/w, 1/x]$ is presented in fig. A, while that of the substring relation R on $X = [3/r, 1/s, 1/t, 1/u, 1/w, 1/x, 1/y, 1/z]$ is presented in fig. B below.

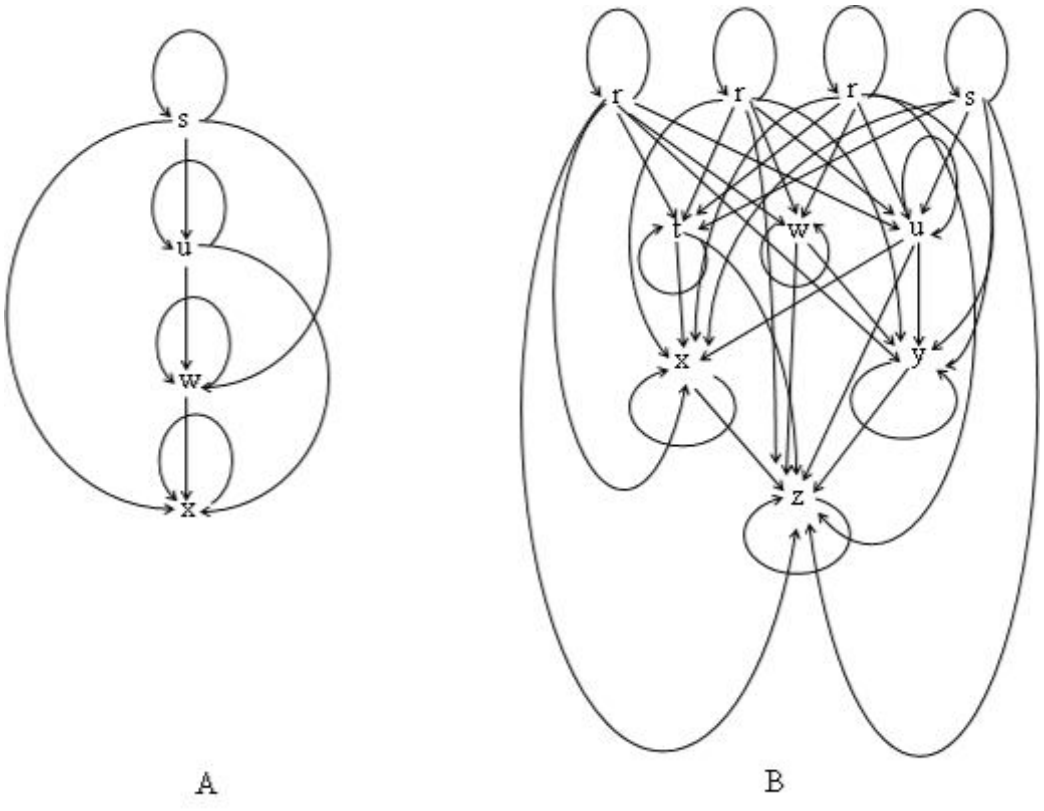


Figure 5.5.1: Graph for prefix relation (A) and Graph for substring relation (B)

Theorem 5.5.3 A prefix (suffix or substring) relation graph is a directed acyclic graph for $n \geq 2$.

Proof

Let $G = (V, E)$ denote the graph of a prefix (suffix or substring) relation and let $(m/x, n/y) \in E$ be represented as $m/xEn/y$.

Let $n \geq 2$, and the graph is cyclic. That is, there exist distinct elements $m_1/x_1, \cdots, m_k/x_k \in V$ such that $m_1/x_1E\cdots Em_k/x_kEm_1/x_1$.

Since these relations are transitive, $m_1/x_1Em_2/x_2$ and $m_2/x_2Em_3/x_3$ imply $m_1/x_1Em_3/x_3$ and, in turn, by repeated application of transitivity, we have $m_1/x_1Em_k/x_k$. Hence, $m_1/x_1 = m_k/x_k$, by antisymmetry of these relations. This contradicts the assumption that $m_1/x_1, \cdots, m_k/x_k$ are distinct and $n \geq 2$.

Hasse diagram of R on X

Let $\mathbb{X} = (X, R)$, where X consists of all nonempty prefixes (suffixes or substrings) of a string $v \in \sum^{\oplus}$ and R, its respective relation on X. In order to represent the relations by Hasse diagram, we assume that $\forall m/x, n/y \in X, m/xRn/y$ only if $\nexists p/z \in X$ such that $m/xRp/z$ and $p/zRn/y$. For each ordered pair for which this relation holds, draw m/x in a vertical plane below n/y (or n/y below m/x) and connect them with a straight line, i.e., a Hasse diagram for prefix (suffix or substring) relation is a directed graph with vertex mset V and edges set E minus self-loops and edges implied by transitivity. As an example, the Hasse diagrams for the prefix relation in fig. A and, that of the substring relation in fig. B, are presented in Fig. C and D, respectively.

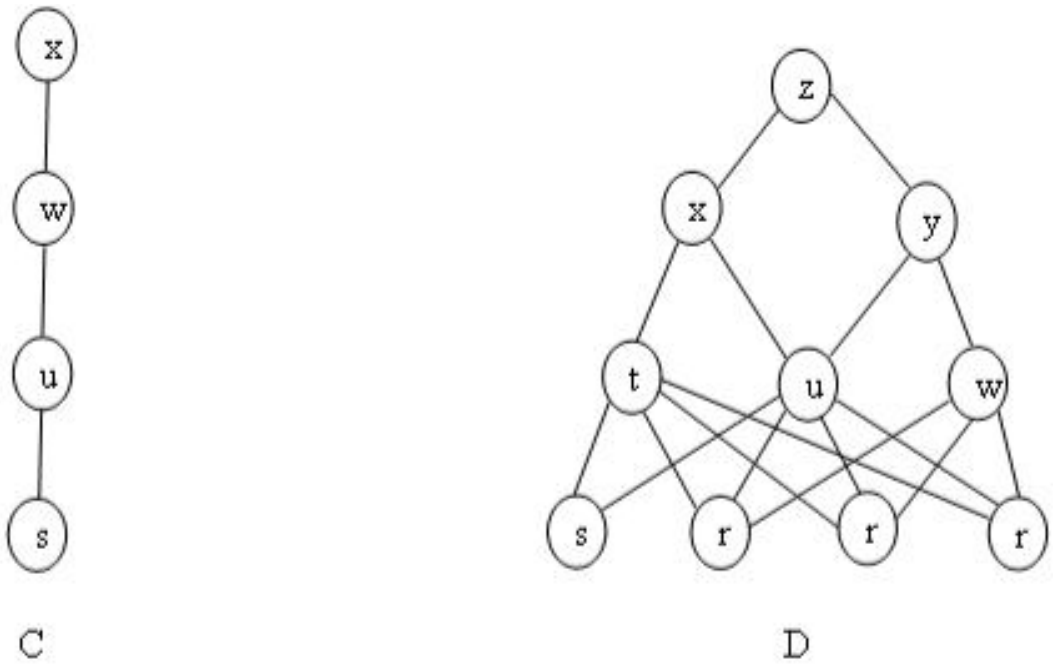


Figure 5.5.2: Hasse diagram for prefix relation (A) and Hasse diagram for the substring relation (B)

Remark 5.10

Akin to set relations, two or more different prefix (suffix or substring) mset relations may have similar Hasse diagrams. For example, the Hasse diagram of the suffix relation R on $X = [k, l, m, n]$, for the nonempty suffixes of the string $abaa$ with $k = a, l = aa, m = baa, n = abaa$, is similar to that of the prefix relation presented in fig. C.

5.5.5 Monoids of Prefixes, Suffixes and Substrings of a nonempty String in a Multilanguage

Let $\sum$ be an alphabet and $\sum^{\oplus}$, the mset of all nonempty strings over $\sum$. For $v \in \sum^{\oplus}$, let $v^{\odot}$ denote the multiset of all prefixes (suffixes or substrings) of v, and R its respective relation. We define a binary operation $*$ on $v^{\odot}$ as follows:
$\forall m/x, n/y \in v^{\odot}$, if $m/xRn/y$, $m/x * n/y = n/y$, otherwise $m/x * n/y = n/y * n/y$.

Thus, $*$ is associative and ε is the identity element. Hence, $(v^{\odot}, *, \varepsilon)$, usually written as $(v^{\odot}, *)$, is a monoid.
Also, every element $m/x \in v^{\odot}$, is idempotent with respect to the operation $*$, since $m/x * m/x = m/x, \forall m/x \in v^{\odot}$. However, $*$ on $v^{\odot}$ is not commutative as $m/x * n/y \neq n/y * m/x, \forall m/x, n/y \in v^{\odot}$.

For example, let $\sum = \{a, b\}$, and let $v = abaa \in \sum^{\odot}$. Consider the substrings $\varepsilon, a, b, ab, a, ba, aba, a, aa, baa, abaa$ of v and let $p = \varepsilon, r = a, s = b, t = ab, u = ba, w = aa, m = aba, n = baa$, and $q = abaa$. Then, $v^{\odot} = [1/p, 3/r, 1/s, 1/t, 1/u, 1/m, 1/w, 1/n, 1/q]$. It is straightforward to see that $*$ is associative, since $\forall m/x, n/y, k/z \in v^{\odot}, (m/x * n/y) * k/z = m/x * (n/y * k/z)$, and $1/p$ is the identity element. Therefore, $(v^{\odot}, *)$ is a monoid.

Homomorphism between Monoids of Prefixes, Suffixes and Substrings of a string in a multilanguage

Let $M = (v^{\odot}, *)$ and $N = (u^{\odot}, *)$ be two monoids of prefixes (suffixes or substrings) of $v, u \in \sum^{\odot}$. A homomorphism between M and N is a function $f : v^{\odot} \longrightarrow u^{\odot}$ such that for every element $m/x \in v^{\odot}$ there exists exactly one element $n/y \in u^{\odot}$ such that $(m/x, n/y) \in f$, the pair $(m/x, n/y)$ occurs only $C_1(m/x, n/y)$ times and, for every pair of elements $m/x, p/z \in v^{\odot}$, if $m/x * p/z$, then

i. $f(m/x * p/z) = f(m/x) * f(p/z)$; and

ii. $f(\varepsilon_{(v^{\odot})}) = \varepsilon_{(u^{\odot})}$, where $\varepsilon_{(v^{\odot})}$ and $\varepsilon_{(u^{\odot})}$ are the same identity on $v^{\odot}$ and $u^{\odot}$ both, as mentioned earlier.

For example, let $\sum = \{a, b\}$ and $v = aba, u = abaa \in \sum^{\odot}$; then their substrings are $\varepsilon, a, b, ab, a, ba, aba$ and $\varepsilon, a, b, ab, a, ba, aba, a, aa, baa, abaa$, respectively. For the substrings of v, let $d = \varepsilon, k = a, m = b, n = ab, p = ba, q = aba, v^{\odot} = [1/d, 2/k, 1/m, 1/n, 1/p, 1/q]$ and, for that of u, let $e = \varepsilon, r = a, s = b, t = ab, h = ba, w = aa, x = aba, l = baa, z = abaa, u^{\odot} = [1/e, 3/r, 1/s, 1/t, 1/h, 1/w, 1/x, 1/l, 1/z]$. Then, $f : (v^{\odot}, *) \longrightarrow (u^{\odot}, *)$, defined by $f\{(1/d, 1/e)/1, (2/k, 2/r)/2, (1/m, 1/w)/1, (1/n, 1/l)/1, (1/p, 1/l)/1, (1/q, 1/z)/1\}$, is a monoid homomorphism of substrings, while $g : (v^{\odot}, *) \longrightarrow (u^{\odot}, *)$, defined by $g\{(1/d, 1/h)/1, (2/k, 2/r)/2, (1/m, 1/e)/1, (1/n, 1/w)/1, (1/p, 1/x)/1, (1/q, 1/z)/1\}$ is not a monoid homomorphism, since $g(1/d) \neq 1/e$.

CHAPTER SIX

SUMMARY, CONCLUSION AND RECOMMENDATIONS

6.1 Summary

In this dissertation, in addition to presenting an overview of the theory of formal languages, suitable operations are defined on the class of partitions of a language which give rise to certain monoids and semigroups. Also, certain algebraic structures of a language defined over a string are described. Moreover, a new operation is defined over a regular language and its characteristic properties are described.

The distinctive features of the two approaches to multisets viz., functional approach and equivalence relation approach are outlined, and mathematics of *sorts* is developed and applied to modify and extend the extant theory of multisets. It is viewed that the equivalence relation approach, due to its set-like foundation, is found suitable to categorical models while the functional approach, owing to its numeric foundation, is found amenable to *hands-on applications*.

In order to develop the concept of a multilanguage, the functional approach is adopted. Multilanguages and their operations are formulated, and their distinctive features vis-a-vis standard formal languages are highlighted. In particular, prefix, suffix and substring relations and their algebraic properties are formulated in a Multilanguage. It is further shown that prefix and suffix relations give rise to a relational structure of a linear multiset order, and substring relations determine only a partial multiset order. The distinctive features of these relations in multilanguage vs. standard formal language are identified by using their graphical and Hasse diagram

representations. Also, certain monoids of these relations and their homomorphisms were introduced.

6.2 Conclusion

In this dissertation, besides presenting a comprehensive and critical survey of existing literature, formal languages and their operations were described and strengthened, and some algebra of languages were formulated. Also, grammars and automata were studied, some new concepts were introduced and exemplified. Moreover, functional and equivalence relation approaches to multisets and their distinctive features were presented and the concept of a multilanguage was developed.
More explicitly, the following are the main contributions of this dissertation:

1. Four distinct equivalence relations, were defined on X^* and u^*, where $u \in X^*$, and X an alphabet, and it was shown that $(X^*/\backsim, \star)$ formed a monoid and $(u^*/\backsim, \star)$ a commutative monoid under catenation (Definition 3.3.2 and Theorem 3.3.2).

2. Three distinct operations $\odot, \oplus$ and $\oslash$ were defined on $\mathcal{F}(X^*)$, the class of all partitions of X^* on an alphabet X, and it was shown that $\mathcal{F}(X^*)$ formed a commutative, idempotent monoid under the first two, but only a semigroup under the last one (Definition 3.3.3 and Theorems 3.3.3, 3.3.4, 3.3.5). It was also shown that a finite $C = (u^*, \star)$ is a cyclic group of order n (Theorem 3.3.6), and languages over a finite u^* form a bounded distributive lattice (Theorem 3.3.7).

3. The concept of length product, denoted $\bigotimes$, was introduced on languages (Definition 4.2.2) and the following theorems were proved:
   Theorem 4.2.1 If the languages L_1 and L_2 are recognized by finite automata M_1 and M_2, respectively, then their length product $L_1 \bigotimes L_2$ is recognized by

a finite automaton, say M.

Theorem 4.2.2 The length product is closed under union and intersection.

4. By introducing the notions of *sorts* and *subsorts*, operations of maximum, minimum and difference of two multisets were reformulated, and a metric was defined on the class of all finite multisets built on a set X (Definitions 5.1.2, 5.1.8, 5.1.9, 5.1.12 and Theorem 5.1.1).

5. Various operations defined on standard formal languages (section 3.2) are extended to multilanguages and, besides having a number of observations, the following main results were obtained.

   Theorem 5.5.1 Multilanguages of prefixes, suffixes and substrings of a string form a bounded distributive lattice.

   Theorem 5.5.2 A prefix (suffix or substring) relational structure always has a unique maximal element and, only (prefix or suffix) relational structure has also a unique minimal element.

   Theorem 5.5.3 A prefix (suffix or substring) relation graph is a directed acyclic graph for $n \geq 2$.

   In section 5.5.5, Monoids of prefixes, suffixes and substrings of a nonempty string in a multilanguage and homomorphism between them were formulated.

6.3 Recommendations

The constructions provided in section 3.3.2, specially defined on X^{n*}, may be found useful to Network segmentation, analysis of large databases, finite-state machines, etc. In particular, an alternative representation of a language, developed in Definition 3.3.1, may be exploited for further research.

List of Journal Publications

- **Journal Papers**

1. Singh, D. and Isah, A. I., (2014). An Outline of the Development of the Theory of Formal languages, *Int. J. Latest Trends in Computing*, Vol. 5, No. 2, 172-181.

2. Singh, D. and Isah, A. I., (2014). Prefixes, Suffixes and Substrings in a Multilanguage: A Perspective, *Applied Mathematical Sciences*, Vol. 8, 6325 - 6339.

3. Singh, D. and Isah, A. I., (2015). Some Algebraic Structures of Languages, *Journal of mathematics and computer science*, vol. 14, 250-257.

4. Singh, D. and Isah, A. I., (2015). A Note on Automata, *Journal of mathematics and computer science*, vol. 15, 88-96.

- **Communicated Journal Paper**

1. Singh, D. and Isah, A. I., Mathematics of Multisets: A Unified Approach, under review.

References

Ahmadi, B., Campbell, C. M., and Doostie, H. (2014). Non-commutative finite monoids of a given order n ≥ 4. *VERSITA*, 22(2):29–35.

Baulch, M. (2010). Metric properties of automata and formal languages. *Department of Mathematics and Statistics, University of Melbourne, pp. 1-46.*

Becker, T., Joshi, A., and Rambow, O. (1991). Long-distance scrambling and tree adjoining grammars. In *: Proceedings of the fifth conference on European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 21-26.*

Bilska, A. O., Linder, K. H., Procopiuc, M., Procopiuc, O., Rodger, S. H., Salemme, J. R., and Tsang, E. (1997). A collection of tools for making automata theory and formal languages come alive. *Duke University, Durham, NC, pp. 1-5.*

Blizard, W. D. (1989). Multiset theory. *Notre Dame Journal of formal logic*, 30:36 – 66.

Cai, J. and Paige, R. (1995). Using multiset discrimination to solve language processing problems without hashing. *New York University, New York,U S A, pp. 1-38.*

Cavalcante, R., Finley, T., and Rodger, S. H. (2004). A visual and interactive automata theory course with jflap 4.0. *Dept of Computer Science, Duke University, Durham, NC, pp. 1-5.*

Cavaliere, M., Freund, R., Oswald, M., and Sburlan, D. (2007). Multiset random context grammars, checkers, and transducers. *Theoretical Computer Science, Elsevier*, 372(2-3):136–151.

Chakraborty, S. (2003). Formal languages and automata theory- regular expressions

and finite automata- computer engineering and networks laboratory. *Swiss Federal Institute of Technology (ETH), Zurich, pp. 1-17.*

Chesnevar, C. I., Cobo, M. L., and Yarcik, W. (2003). Using theoretical computer simulators for formal languages and automata theory. *Department of Computer Science and Engineering, Univ. Nacional del Sur-Alem 1253, Bohia Blanca, Argentina, pp. 1-5.*

Chomsky, N. (1956). Three models for the description of languages. *Transactions on Information Theory*, 2(3):113–124.

Chomsky, N. (1957). Syntactic structures. *Mouton and Co., The Hague, Netherlands.*

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):137–167.

Dershowitz, N. and Manna, Z. (1979). Proving termination with multiset orderings. *Comm. ACM*, 22:465 – 476.

Gallier, J. (2010). Introduction to the theory of computation, formal languages and automata models of computation. *Lecture Notes, University of Pennsylvania, pp. 1-60.*

Girish, K. P. and John, S. J. (2009). Relations and functions in multiset context. *Information Sciences*, 179(6):758–768.

Grefen, P. W. P. J. and de By, R. A. (1994). A multiset extended relational algebra: A formal approach to a practical issue. *In: 10th International Conference on Data Engineering (ICDE), Houston, TX, USA, pp. 80-88.*

Grune, D. and Jacobs, C. H. (1990). Parsing techniques – a practical guide. *Ellis Horwood, England, pp. 1-320.*

Harrow, K. (1978). How to show something is not: Proofs in formal language and computability theory. *Computer and Information Science Department, Brookln College, Brookln, Newyork, pp. 27-30.*

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata, Languages, and Computation.* 2^{nd} edn., Addison-Wesley.

Hopcroft, J. E. and Ullman, J. D. (1969). *Formal Languages and Their Relations to Automata.* Addison-Wesley Publishing.

Hughes, C. (2007). Formal languages and automata theory. *Lecture Notes, University of Central Florida, pp. 27-28.*

Hung, T. and Rodger, S. H. (2000). Increasing visualization and interaction in the automata theory course. *Computer science Department, Duke University, Durham, pp. 1-6.*

Huybregts, R. (1984). The weak inadequacy of context-free phrase structure grammars. *In: Ger Jan de Haan, Mieke Trommelen, and Wim Zonneveld, editors, Van periferie naar kern, Foris,Dordrecht, pp. 81-99.*

Ilie, L. (1997). Some recent results on contextual languages. *Turku Centre for Computer Science TUCS, Lemminaisenkatu 14 Finland*, (Technical Report No. 96):1–17.

Jager, G. and Rogers, J. (2012). Formal language theory: Refining the chomsky hierarchy. *Phil. Trans. R. Soc. B*, 367:1956–1970.

Jantzen, M. (2003). Intersecting multisets and applications to multiset languages. *Universitat Hamburg, FB Informatik, Hamburg, pp. 1-8.*

Jiang, T., Ravikumar, B., Li, M., and Regan, K. W. (2010). Formal grammars and languages. *Lecture Notes, pp. 1 – 40.*

Joshi, A. (1985). How much context-sensitivity is necessary for characterizing structural descriptions - tree adjoining grammars. *In: David Dowty, Lauri Karttunen, and Arnold Zwicky, editors, Natural Language Processing, Theoretical, Computational and Psychological Perspectives, Cambridge University Press, Cambridge, UK.*

Joshi, A. K., Shanker, K. V., and Weir, D. (1990). The convergence of mildly context-sensitive grammar formalisms. *Technical Reports (CIS)*, Paper 539:1–65.

Kari, J. (2013). Automata and formal languages. *University of Turku, Finland, pp. 1– 150.*

Kleen, S. C. (1952). *Introduction to Metamathematics.* D. Van Nostrand, Wolters-Noordhoff.

Knuth, D. (1981). *The Art of Computer Programming, Semi-numerical Algorithms, Vol. II,* 2^{nd} *edn.* Addison-Wesley.

Knuth, D. E. (1992). Context-free multilanguages. *Computer Science Department, Stanford University, Academic Press, pp. 1-13.*

Linz, P. (2001). *An Introduction to Formal Languages and Automata.* Jones and Bartlett Publishers, Inc., 3^{rd} edn.

Lloyd, J. W. (1998). Programming with multisets. *Department of Computer Science, University of Bristol, U K, pp. 1-16.*

Maciel, A. (2014). Automata and computability, lecture notes, department of computer science. *Clarkson University, pp. 1-82.*

Meduna, A. and Vitex, M. (2004). New language operations in formal language theory. *Department of Information Technology, Brno University of Technology, Brno, Czech Republic, pp. 1-24.*

Monro, G. P. (1987). The concept of multiset. *Zeitschriftfür mathematische Logik und Grundlagen der mathematic [mathematical logic quarterly]*, 33(2):171– 178.

Nederhof, M. J., Satta, G., and Shieber, S. (2003). Partially ordered multiset, context-free grammars and free-word-order parsing. *In Proceedings of the Eighth International Workshop on Parsing Technologies,Nancy, France, pp. 171-182.*

Nederpelt, R. P. (2002). Weak type theory: A formal language for mathematics. *Eindhoven University of Technology, pp. 1-76.*

Perrin, D. (2003). Automata and formal languages. *Universite de Marne-la-vallee, pp. 1-22.*

Postal, P. M. and Langendoen, D. T. (1984). English and the class of context-free languages. *Computational Linguistics*, 10(3-4):177–181.

Power, J. (2002). Notes on formal language theory and parsing. *Department of Computer Science National University of Ireland, pp. 1-40.*

Rodger, S. H., Lim, J., and Reading, S. (2007). Increasing interaction and support in the formal languages and automata theory course. *Computer Science Department, Duke University Durham, pp. 1-5.*

Rodger, S. H., Wiebe, E., Lee, K. M., Morgan, C., Umar, K., and Su, J. (2009). Increasing engagement in automata theory with jflap. *SIGCE' 09, march 3-7, 2009, Chattanooga, Tennessee, USA, pp. 1-5.*

Rogers, J. and Pullum, G. (2007). Aural pattern recognition experiments and the subregular hierarchy:. In *Marcus Kracht, editor, Proceedings of 10th Mathematics of Language Conference, University of California, Los Angeles, pp. 1-7.*

Rosen, K. H. (1999). Discrete mathematics and its applications. *Mc Graw-Hill International Editions, 4th edn.*

Ruohonen, K. (2009). Formal languages. *Lecture Notes, Tampere University of Technology, pp. 1-94.*

Sassaman, L., Patterson, M. L., Bratus, S., Locasto, M. E., and Shubina, A. (2011). Security applications of formal language theory. *Dartmouth, Computer Science Technical Report, pp. 1-31.*

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Singh, D., Ibrahim, A. M., Tella, Y., and Singh, J. N. (2007). An overview of the applications of multisets. *Novi Sad J. Math.*, 37(2):73–92.

Singh, D., Ibrahim, A. M., Tella, Y., and Singh, J. N. (2011). Complementation in multiset theory. *International Mathematical Forum*, 6:37 – 40.

Singh, D., Ibrahim, A. M., Yohanna, T., and Singh, J. N. (2008). A systematization of fundamentals of multisets. *Lecturas Mathematicas*, 29:33–48.

Syropoulos, A. (2003). Categorical models of multisets. *Romanian Journal of Information Science and Technology*, 6(3-4):393–400.

Tiplea, F. L. (2005). Formal language and automata theory. *Lecture Notes, School of Computer Science, University of Central Florida, Orlando, FL 32816, pp. 1-164.*

Tremblay, J. and Manohar, R. (1997). *Discrete Mathematical Structures with Applications to Computer Science.* Tata McGraw-Hill.

Wang, Y. (2009). A formal syntax of natural languages and the deductive grammar. *Foundamenta Informaticae*, 90(4):353–368.

Weyl, H. (1959). *Philosophy of Mathematics and Natural Science (Appendix B).* Revised and Augmented English edn., fourth printing.

Wildberger, N. J. (2003). A new look at multisets. *School of Mathematics, UNSW, Sydney, Australia, pp. 1- 21.*

Worthington, J. (2009). A bialgebraic approach to automata and formal language theory. *Mathematics Department, Malott Hall, Cornell University, Ithaca, USA, pp. 1-29.*

Xavier, S. P. E. (2005). *Theory of Automata, Formal Languages and Computation.* New Age International Publishers.

Yager, R. R. (1986). On the theory of bags. *International Journal of General Systems*, 13:23–37.