Chapter 1

Introduction

1.1 Formal Languages and Operations: Introduction and Motivation

Formal language theory, the study of abstract sets of words over a fixed alphabet of symbols, is one of the oldest research areas in the theory of computing. Despite its age, formal language theory also continues to attract new attention, especially its application to various fields, including the theory of codes, bio-informatics and many others.

Arguably, at the core of formal language theory are two central concepts: that generative devices, such as automata and grammars, can be used to define a class of languages, and that languages can be combined to form new languages using language operations. Mateescu and Salomaa state that "a major part of formal language theory can be viewed as the study of finitary devices for generating infinite languages" [148, p. 2]. The importance of language operations is a slightly more subtle point. However, we cannot underestimate their power: for instance, language operations themselves are at the heart of several fundamental language generation systems–for instance, regular expressions, L-systems and context-free grammars. Further, we can mention the study of concepts such as cones and abstract families of languages (AFLs), for which closure properties under language operations are the defining characteristics.

The two concepts of generative devices and language operations form the starting point for any meaningful study of formal language theory. In particular, closure properties of classes of languages under various language operations give us insight into the both the power of the classes and the power of the operations. Closure properties also help us to meaningfully compare two classes of languages.

However, operations on languages also have crucial importance as emerging areas of research gain importance. In particular, the interpretation of strands of DNA as words in a formal language has been the subject of much research recently, in theoretical areas as well as areas fundamentally linked to the use of DNA as a natural computing method. The language operations under investigation are models of the manner in which DNA strands interact under various settings.

A fundamental development in the area of language equations was the introduction of the notion of shuffle on trajectories, defined by Mateescu *et al.* [147]. Shuffle on trajectories is a framework for defining word operations based on a set of *trajectories*, a language which specifies the way in which the corresponding operation behaves. Thus, shuffle on trajectories is a parameterized class of language operations: each choice of a set of trajectories yields a distinct language operation. The idea of replacing the study of word operations by the study of languages is a major innovation, and leads to very clear, unified results on the applicable language operations.

Operations modeled by shuffle on trajectories include concatenation, the most fundamental language-theoretic operation, the standard shuffle operation, which has a long history in the study of formal languages, as well as many variants of the shuffle operation, including insertion, literal shuffle, perfect shuffle and many others. Each of these operations has been the subject of study in the literature, both for specific applications – including the theory of codes, concurrency theory and other areas – as well as for research into the formal properties of these operations, and their effect on classes of languages.

This thesis examines the concept of trajectories in greater detail. In particular, we seek to unify several different areas of research in theoretical computer science by investigating each of them in the framework of shuffle on trajectories. By formalizing each of these areas, we provide new insight into their fundamental results. Results such as decidability problems and closure properties can be examined in a uniform way, often leading to much simpler proofs.

One result of this research is a demonstration that the shuffle on trajectories formalism, introduced as a unification for word operations, is also important as a unification for more complex constructs. For instance, we define classes of languages related to codes using the shuffle on trajectories model. By doing so, we can model an entire class of languages by a single set of trajectories. This further re-enforces the value of the trajectory model.

In the following four sections, we present an informal description of the main areas of research in this thesis: descriptional complexity, the theory of codes, language equations and iterated language operations.

1.2 Descriptional Complexity of Languages

Descriptional complexity is the study of measures of complexity of languages and operations. It is a very broad area, and includes much work on varied classes of languages. We focus our work on descriptional complexity on the regular languages, a fundamental class of languages. For regular languages, the main focus of work on descriptional complexity is on *state complexity*. The (deterministic, nondeterministic) state complexity of a regular language is the minimal number of states in any (deterministic, nondeterministic) finite automata accepting that language. Given a binary language operation α under which the regular languages are closed, the (worst case) state complexity of α is a function f such that for all regular languages L_1, L_2 accepted by automata of size n_1, n_2, there exists an automaton of size $f(n_1, n_2)$ accepting $\alpha(L_1, L_2)$. For instance, if we are interested in the union operation, it is known that an automaton of size $n_1 n_2$ can be found accepting the union of two languages which are accepted by automata of size n_1 and n_2.

Recently, research into the state complexity of regular languages has seen a great deal of activity. This work is motivated by the desire to have reliable estimates of the amount of memory required for automata when certain language operations are applied. This is crucial in applied areas where finite automata are used in practice, for instance, in pattern matching, natural language processing, and other areas.

We examine the state complexity of shuffle on trajectories in Chapter 4. Being an infinite class of operations, shuffle on trajectories presents some unique challenges; previous results on the state complexity of operations have focused on single operations, rather than an entire family of operations. However, the fact that shuffle on trajectories is a class of language operations parameterized by a set of trajectories–which is itself a language–allows us to make interesting comparisons between the descriptional complexity of a set of trajectories and the state complexity of the resulting shuffle on trajectories operation. We find that another descriptional complexity measure of languages, namely the *density* of a language, gives us interesting insight into the relationship between the complexity of a set of trajectories and the complexity of the resulting language operation. In particular, we find that less dense sets of trajectories correspond to less complex operations, in terms of state complexity.

1.3 Codes and Trajectories

A code is a language which has strong decodability properties: given a sequence of words from a code which are concatenated together, there is only one way to recover the original code words from the concatenated sequence. Codes have many applications, including compression, error detection and security [97, pp. 511–512].

Subclasses of the class of codes, such as prefix codes and hypercodes, are often studied for interesting combinatorial and mathematical properties. This is sometimes know as the *theory of variable-length codes*. In Chapter 6, we present our contribution to this area, which we call T-codes. Intuitively, T-codes represent the natural extension of certain subclasses of codes, including prefix codes and hypercodes, which are defined via shuffle on trajectories. With the definition of T-codes, we can present results about classes of T-codes by arguing instead about the associated sets of trajectories T. This yields general results about properties of interest for T-codes.

We note that the idea of a general method for defining several code-like classes of languages has received some attention in the literature. We briefly note some of this work in Section 6.1. However, we feel that the notion of a T-code, in using shuffle on trajectories, has the advantage of being general enough to capture several classes of codes studied in the literature on variable length codes, but at the same time, is specific enough to allow us to obtain interesting results. Specifically, decidability properties are often trivial in the framework of T-codes.

1.4 Language Equations

The study of language equations, that is, equations or systems of equations consisting of constant languages, language operations, and unknowns, and which are solved in terms of languages, is one of the oldest areas in the theory of computation. Many fundamental areas of computer science are intricately linked to the study of language equations.

As an example, we note the context-free languages, which are crucial in the design of programming languages and compilers. The theory of context-free grammars as a generative device is well developed. However, each context-free grammar is equivalent to a system of language equations, and many deep results in this area have been obtained (see, e.g., Autebert *et al.* [10]).

Our study of language equations will focus on equations whose operations are taken from the class of operations defined by shuffle on trajectories. In studying the decidability of the existence of solutions to such an equation, it will be useful to define the notion of an *inverse* to shuffle on trajectories. This inverse is known as *deletion along trajectories*. We first study the properties of deletion along trajectories in Chapter 5, and apply it to language equations in Chapter 7.

The inverse of a language operation, first defined by Kari [106], allows us to solve language equations much in the same way we solve equations such as $a + x = b$, where a, b are integers and x is unknown, and, as noted by Kari, our inverse works in a similar manner to how subtraction works as an inverse to addition. In particular, given our equation, we can recover the unknown value by applying the inverse operation to the known constants, much in the same way as in solving the equation $a + x = b$.

We further study language equations with two unknowns. In this case, the constructions involved are more complicated. However, we succeed in characterizing a large class of trajectories, including many studied operations, with which we are able to positively solve the decidability problem for language equations in two unknowns. For all of the equation forms we consider, we also obtain complementary undecidability results.

1.5 Iteration

In Chapter 8, we investigate iterated versions of trajectory-based operations. Our main motivation is the examination of languages which are closed under a fixed shuffle on trajectories or deletion along trajectories operation. There is a simple relationship between languages which are closed under shuffle on (resp., deletion along) trajectories and the iterated shuffle on (resp., deletion along) trajectories operation.

We also examine other applications of iterated trajectory-based operations. In particular, we examine the concept of primitivity, the property of a word not being able to be expressed as the power of another word, as it is related to shuffle on trajectories. The concept of primitivity, in relation to the concatenation operation, is a natural concept in formal language theory, an interesting intersection between the theory of formal languages and combinatorics, and also the source of one of the most well-known open problems in formal language theory–namely, whether or not the set of all primitive words is a context-free language. Primitivity was extended to both shuffle and insertion by Kari and Thierrin [118] and to a very general class of operations by Hsiao *et al.* [69]. In Chapter 8, we find that with a slightly more natural definition of iterated shuffle on trajectories, we find that the notion of primitivity can be examined without some of the assumptions made in the more general case of Hsiao *et al.* [69]. We also consider extensions to the very well-known results of Lyndon and Schützenberger via shuffle on trajectories. This allows us to further examine conditions of uniqueness and existence of primitive roots of words.

We also revisit language equations in Chapter 8 and characterize the solution to certain explicit language equations involving shuffle on trajectories using its iterated counterpart. This is in contrast to the implicit language equations examined in Chapter 7, where we examined the question of the decidability of the existence of solutions. Our characterizations of the solutions of explicit language equations is a parallel of classic language equations which have been studied in the literature.

1.6 Organization

This thesis is organized as follows. Chapter 2 is devoted to preliminary definitions, and may be referred to as necessary by the reader. Chapter 3 examines related work on shuffle, iteration and shuffle on trajectories.

In Chapter 4, we discuss the complexity of the shuffle on trajectories operation in terms of state complexity, a much studied measure of the complexity of an operation which acts on regular languages.

Chapter 5 develops the concept of *deletion along trajectories*. We show that it serves as an inverse to shuffle on trajectories, in the sense defined by Kari [106]. We also examine the closure properties of deletion on trajectories.

In Chapter 6, we apply the framework of traditional code classes to shuffle on trajectories. This gives a generalization of many classical code classes. We examine many previously studied aspects of codes in this general setting.

Chapter 7 examines the solutions to equations involving shuffle and deletion along trajectories. Several questions for several forms of equations are examined, as well as certain forms of systems of equations involving shuffle on trajectories.

Finally, in Chapter 8, we consider the iteration of shuffle and deletion on trajectories. We are particularly interested in the relationship between iteration and the smallest language closed under shuffle on trajectories.

We conclude in Chapter 9 by examining some open problems raised in this thesis. We also discuss possible future research areas related to the trajectories model.