2020

Maximality and Applications of Subword-Closed Languages

Rhys Davis Jones
University of North Florida, Jonerd19@gmail.com

Maximality and Applications of Subword-Closed Languages

by

Rhys Jones

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science in Mathematical Sciences
Department of Mathematics and Statistics
College of Arts and Sciences
University of North Florida

Chairperson: Daniela Genova, Ph.D.
Michelle Dedeo, Ph.D.
Jae-Ho Lee, Ph.D.

Date of Approval:
7/23/2020

Keywords: Finite State Automata, Regular Languages, Subword Closure,
Involution Mappings, Strictly Locally Testable Languages, DNA Code Words

Table of Contents

Abstract

Characterizing languages D that are maximal with the property that $D^* \subseteq S^{\otimes}$ is an important problem in formal language theory with applications to coding theory and DNA codewords. Given a finite set of words of fixed length S, the constraint, we consider its subword closure, $S^{\otimes}$, the set of words whose subwords of that fixed length are all in the constraint. We investigate these maximal languages and present characterizations for them. These characterizations use strongly connected components of deterministic finite automata and lead to polynomial time algorithms for generating such languages. We prove that the subword closure $S^{\otimes}$ is strictly locally testable. Finally, we discuss applications to coding theory and encoding arbitrary blocks of information on DNA strands. This leads to very important applications in DNA codewords designed to obtain bond-free languages, which have been experimentally confirmed.

Chapter 1

Introduction

Formal Language Theory, a branch of Theoretical Computer Science, is concerned with the study of sets of finite strings of symbols, languages, where each symbol is chosen from a given finite set, the alphabet. Our investigation is mainly based on work from Konstantinidis and Santean, [18], on subword-closed languages. We investigate languages that are maximal with the property that every word obtained by a finite concatenations of words in the languages conforms to a given subword constraint. We also investigate the subword closure itself. As it turns out, both the maximal languages mentioned earlier and the subword closure are regular languages. In addition, we prove that the subword closure belongs to a subclass of regular languages called strictly locally testable languages.

Given a subword constraint, which is a set S of words of some fixed length k, over a given alphabet Σ, the subword closure $S^{\otimes}$ of S is the set of words for which all subwords of length k are in S. In 2008, Cui and Konstantinidis [4] considered the problem of encoding arbitrary sequences of data blocks into the words of $S^{\otimes}$. That is to say, if Σ^k is the set of all words of length k over some alphabet Σ, for some $k \geq 1$, Cui and Konstantinidis introduced a method for encoding Σ^k onto some set of words D such that $D^* \subseteq S^{\otimes}$. This method, however, does not give a *complete* structural characterization of these languages D.

In [18], Konstantinidis and Santean consider the problem of characterizing both *nonempty* and then *nontrivial* languages D whose words are of length at least k and are maximal with the property $D^* \subseteq S^{\otimes}$. The more general problem of computing

maximal regular languages D such that $D^* \subseteq R$, where R is any regular language, was previously addressed by Kari and Seki in [15]. However, the method described there leads to an algorithm that has an exponential number of steps. The method we detail, introduced in [18], uses the structural characterization for $S^{\otimes}$ introduced in [13], where a deterministic finite state automaton called a "*trie*", that accepts $S^{\otimes}$, is constructed. Strongly connected components of this trie are examined to obtain the desired languages D. The characterization of nontrivial languages D has the additional requirement that the strongly connected component must contain a fork state.

Since it was possible to construct a finite state automaton that accepts $S^{\otimes}$, $S^{\otimes}$ is a regular language. We prove that $S^{\otimes}$ also belongs to a subclass of regular languages called *strictly locally testable* languages. Further, we connect this to forbidding-enforcing systems, introduced by Ehrenfeucht and Rozenberg in [7], and use the results in [9] to show that because $S^{\otimes}$ is strictly locally testable, there exists a finite fe-system such that $L(\mathcal{F}, \mathcal{E}) = S^{\otimes}$, making $S^{\otimes}$ an fe-language defined by a finite fe-system. Since there is an fe-system with empty enforcers, we provide a finite set that defines the language $S^{\otimes}$, making $S^{\otimes}$ an f-language defined by a finite set of forbidders. By [9], $S^{\otimes}$ is also *locally testable*, which also follows from the fact that it is *strictly locally testable*.

After Adleman [1] realized the potential of DNA computing, the field DNA coding became a very popular area of study. DNA computing relies on the notion that one is able to encode *input data* into a collection of DNA molecules and then perform a series of *operations* on them, resulting in a collection of modified DNA molecules containing *output data*. In practice, these collections of DNA molecules take the form of test tubes containing single-stranded DNA molecules suspended in a non-reactive liquid. The operations can take the form of mixing these test tubes; sometimes with other collections of encoded DNA molecules or sometimes with enzymes that are designed to cause DNA strands to separate or bond at specific junctions [10].

Regardless of the form they take, the success of a DNA computing operation is

generally contingent on ones ability to control the *bonds* that are formed between single-stranded DNA molecules. These *bonds* are formed thanks to the Watson-Crick *complementarity* property of the four nucleotides that make up any given DNA strand: Adenine, Thymine, Guanine, and Cytosine (often denoted by A, T, G, and C, or a, c, t, and g respectively). Specifically, A is complementary to T and G is complementary to C. Thus, a single strand of nucleotides can be represented by a word over the DNA alphabet $\{A, C, T, G\}$ and this automatically implies the complement strand by Watson-Crick complementarity. Complicating matters, DNA molecules also have an inherent *orientation*. This orientation is accounted for by placing the symbols "$5'-$" and "$-3'$" on opposite ends of a sequence of nucleotides. This can result in two ostensibly identical DNA molecules having very different chemical properties. For instance, the molecules $5' - GCTTAG - 3'$ and $3' - GCTTAG - 5'$ are not the same, despite describing the same sequence of nucleotides.

Methods for encoding DNA strands to avoid possible unintended bonds (or *hybridizations*) were needed, and so notions from Formal Language Theory were employed and expanded upon to accomplish this. In 2004, Jonoska and Mahalingham [11] introduced the concept of a θ-k code. Any language that is a θ-k code is assured to have no words with k-length subwords that are Watson-Crick complements of k-length subwords of other words in the language. Here, the involution mapping θ was introduced to describe the Watson-Crick complementarity property of DNA codes. In 2005 Kari, Konstantinidis, and Sosik [13] describe θ-k codes as *maximal bond-free languages* and provide a structural characterization for these languages using the subword closure operation. In this paper, we investigate the characterizations of maximal languages that can be applied to θ-k codes.

This paper is organized as follows: Chapter 2 contains Formal Language Theory and Automata Theory notions that will be used later on. In Chapter 3, we introduce the concept of the subword closure, and detail a deterministic finite automaton that accepts it. In Chapter 4, we demonstrate a method for finding maximal sets of words D, such that $D^* \subseteq S^{\otimes}$. In Chapter 5, we define and discuss many DNA codes

including θ-k codes. Next, two subsets of regular languages, strictly locally testable and locally testable languages, are discussed and defined in Chapter 6 and we discuss the connection between subword closure and fe-systems. We also prove that the subword closure is strictly locally testable. In Chapter 7, we discuss some real world applications of the subword closure, as well as, the maximal languages that were given a structural characterization. Finally, Chapter 8 gives some concluding remarks and suggestions for future research.

Chapter 2

Formal Languages and Automata

In this chapter, we recall relevant definitions and properties from Formal Language Theory and Automata Theory to set the notation used throughout the paper. In the first subsection, we define and give examples of language-theoretic terms used including alphabets, words, and subwords, as well, as some special notation that is used in papers that we reference heavily. The second subsection covers regular operations and regular expressions. The final subsection defines finite state automata and connects them to regular languages.

2.1 Formal Languages

An *alphabet* is a finite, nonempty set of symbols (letters), often denoted by Σ, A, or Δ. Symbols are generally denoted by $a, b, c, d, \ldots$. A *word* (*string*) is a finite sequence (*concatenation*) of symbols from Σ. The word containing no letters is called the *empty word* and is denoted by λ. Words are generally denoted by $s, t, u, v, w, x, y, z, \ldots$. If Σ is an alphabet, we define Σ^* as the set of all words over Σ, including λ, the empty word. Additionally, we define Σ^+ as the set of all nonempty words over Σ. Note that Σ^* and Σ^+ are infinite. A *language* is a subset of Σ^*. Alternatively, languages can be defined as a set of words, either finite or infinite. Languages are generally denoted using capitalized letters $L, K, H, \ldots$.

Example 2.1.1 Given $\Sigma = \{a, b\}$, the set of all words over Σ is the set of all

possible concatenations of a and b. Thus,

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, abb, bab, bba, bbb, aaaa, aaab, \dots\}. \quad \lozenge$$

As we will see later, the $*$ and $+$ operators can be extended to languages. To that end, observe that each word is comprised of a certain number of symbols. This gives rise to an inherent characteristic of words. We give a formal definition of this characteristic below, along with an example.

Definition 2.1.2 Let w be a word over Σ. The *length* of w is the number of symbols in it and is denoted by $|w|$. The length of the empty word, λ, is 0, i.e. $|\lambda| = 0$.

For example, If $\Sigma = \{a, b\}$ and $w = bbabba$, then the length of w is 6, i.e. $|w| = 6$. Note that the notation for the length of a word and the notation for the cardinality of a set are identical. However, this should not cause any confusion as context should be enough to differentiate between the two. We make use of this new definition for length by defining a commonly used set.

Definition 2.1.3 Let Σ be an alphabet. We define Σ^k as the set of words over Σ of length k, namely, $\Sigma^k = \{w \in \Sigma^* : |w| = k\}$.

To illustrate the difference between cardinality and length consider the following example.

Example 2.1.4 Let $\Sigma = \{a, b\}$. Then,

$$\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

So, we have that $|\Sigma^3| = 8$. Whereas, for every $w \in \Sigma^3$, $|w| = 3$. This is because

$$|aaa| = |aab| = |aba| = |abb| = |baa| = |bab| = |bba| = |bbb| = 3. \quad \lozenge$$

We now introduce an important operation on words and then define sets that make use of this operation. Given two words u and v over Σ, the *concatenation* of

u and v, denoted uv, is obtained by writing v after u. Consequently, uv is also a word over Σ.

Example 2.1.5 Let $\Sigma = \{a, b\}$, let the words $u = ab$ and $v = bb$.
Note that both u and v are words over Σ. We have that the concatenation of u and v, is $uv = abbb$. $\Diamond$

A *prefix* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^*$ such that $w = ux$, where $x \in \Sigma^*$. The *set of prefixes* of w is denoted by $Pref(w)$. A *suffix* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^*$ such that $w = xu$, where $x \in \Sigma^*$. The *set of suffixes* of w is denoted by $Suff(w)$. A *subword* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^*$ such that $w = xuy$, where $x, y \in \Sigma^*$. The *set of subwords* of w is denoted by $Sub(w)$.

There is also the notion of *proper* prefixes, suffixes, and subwords. A *proper prefix* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^+$ such that $w = ux$, where $x \in \Sigma^+$. The *set of proper prefixes* is denoted by $PPref(w)$. A *proper suffix* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^+$ such that $w = xu$, where $x \in \Sigma^+$. The *set of proper suffixes* of w is denoted by $PSuff(w)$. A *proper subword* of a word $w \in \Sigma^*$ is any word $u \in \Sigma^+$ such that $w = xuy$, where not both $x, y \in \Sigma^*$. The *set of proper subwords* of w is denoted by $PSub(w)$.

Example 2.1.6 If $w = aaab$, then

$$\begin{aligned}
Pref(w) &= \{\lambda, a, aa, aaa, aaab\},\\
PPref(w) &= \{a, aa, aaa\},\\
Suff(w) &= \{\lambda, b, ab, aab, aaab\}\\
PSuff(w) &= \{b, ab, aab\},\\
Sub(w) &= \{\lambda, a, b, aa, ab, aaa, aab, aaab\}, \text{ and}\\
PSub(w) &= \{a, b, aa, ab, aaa, aab\}.
\end{aligned}$$

$\Diamond$

Languages are sets, thus the Boolean operations of union, intersection, and complement follow naturally. A formalization of each is given below and these are all from [20].

Definition 2.1.7 Let L_1 and L_2 be two languages over Σ and L be any language over Σ. Define the *union, intersection, difference, and complement* of languages as follows:

$$L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$$
$$L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$$
$$L_1 \setminus L_2 = \{w \mid w \in L_1 \text{ and } w \notin L_2\}$$
$$L^c = \{w \mid w \notin L\} = \Sigma^* \setminus L$$

Both concatenation and the set of subwords can be extended to languages; however, since languages are sets, special consideration must be taken. The next definition formalizes the notion of concatenation of two languages.

Definition 2.1.8 Let L_1 and L_2 be two languages over Σ. The *concatenation* of L_1 and L_2 is defined as follows.

$$L_1L_2 = \{uv \mid u \in L_1 \text{ and } v \in L_2\}$$

Let L be a language over Σ. The *powers* of L are defined recursively as follows.

1. $L^0 = \{\lambda\}$
2. $L^{i+1} = LL^i$, for $i \geq 0$.

In the next example, we illustrate how to concatenate two different languages with each other, as well as, how to find a certain power of a language.

Example 2.1.9 Let $L_1 = \{a, ab\}$ and $L_2 = \{b, bb\}$. Then,

$$L_1L_2 = \{ab, abb, abbb\} \text{ and}$$
$$L_1^3 = L_1L_1^2 = L_1\{aa, aab, aba, abab\}$$
$$= \{aaa, abaa, aaab, abaab, aaba, ababa, aabab, ababab\}.$$

$\Diamond$

Note that concatenation of languages is not necessarily commutative. Consider the following example.

Example 2.1.10 Let $L_1 = \{a, ab\}$ and $L_2 = \{b, bb\}$. Then,

$$L_1L_2 = \{ab, abb, abbb\}, \text{ whereas}$$
$$L_2L_1 = \{ba, bab, bba, bbab\}.$$

$\Diamond$

Using the concatenation of languages and the union of languages, we can extend the notions of Σ^+ and Σ^* to languages, as well. Let Σ be an alphabet, and let L be a language over Σ. The $*$-*Kleene Closure*, L^*, and $+$-*Kleene Closure*, L^+, named after Stephen Kleene, who first defined them, are given as follows.

$$L^* = \bigcup_{i=0}^{\infty} L^i$$
$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

.

Note that the difference between L^* and L^+ is that λ is always in L^*, but not necessarily in L^+. Since $L^0 = \{\lambda\}$ for any L, λ is always in L^*. However, λ is in L^+, only if $\lambda \in L$. If $\lambda \notin L$, λ is not in L^+ and $L^+ = L^* \setminus \{\lambda\}$ in that case. In the case that $\lambda \in L$, then $L^* = L^+$. For clarification, consider the following example.

Example 2.1.11 Let $\Sigma = \{a, b\}$ and let $L = \{aa, ab, bb\}$ be a language over Σ. We have that,

$$L^+ = L \cup L^2 \cup L^3 \cup ...$$
$$L^+ = \{aa, ab, bb\} \cup \{aaaa, aaab, aabb, abaa, abab, abbb, bbaa, bbab, bbbb\} \cup \ldots,$$

whereas

$$L^* = \{\lambda\} \cup \{aa, ab, bb\} \cup$$
$$\{aaaa, aaab, aabb, abaa, abab, abbb, bbaa, bbab, bbbb\} \cup \ldots.$$

$\Diamond$

Each of $Pref(w)$, $Suff(w)$, and $Sub(w)$ can also be extended to languages by using the union of languages.

Definition 2.1.12 Let L be a language over the alphabet Σ then the set of sub-

words of L, the set of prefixes of L, and the set of suffixes of L, denoted $Sub(L)$, $Pref(L)$, and $Suff(L)$, respectively, is defined by:

$$Sub(L) = \bigcup_{w \in L} Sub(w);$$
$$Suff(L) = \bigcup_{w \in L} Suff(w); \text{ and}$$
$$Pref(L) = \bigcup_{w \in L} Pref(w).$$

Furthermore, we define $Sub_k(L)$, $Pref_k(L)$, and $Suff_k(L)$ as the set of subwords of L of length k, the set of prefixes of length k, and the set of suffixes of length k, respectively. Note that the set of proper subwords is $PSub(L) = Sub(L) \setminus L$, the proper prefixes are $PPref(L) = Pref(L) \setminus L$, and the proper suffixes are $PSuff(L) = Suff(L) \setminus L$.

In the next example, we illustrate the set of subwords of a language and also show how $Sub_k(L)$ and $Sub(L)$ differ.

Example 2.1.13 Let $L = \{abb, bbab\}$ Then,

$$Sub(L) = Sub(abb) \cup Sub(bbab) = \{\lambda, a, b, ab, ba, bb, abb, bab, bba, bbab\}.$$

Note that $Sub(L)$ only has three words of length 3. Thus, we have that

$$Sub_3(L) = \{abb, bba, bab\}.$$ $\Diamond$

There is a special notation that is used by Konstantinidis and Santean in [18] for the set of subwords that follow a certain prefix in L. In a future section we will draw heavily from this paper. For that reason, we give the definition for "$x^{-1}L$" below and with this, we conclude this subsection.

Definition 2.1.14 Let x be a word and L be a language, then

$$x^{-1}L \triangleq \{z \in \Sigma^* \mid xz \in L\}$$

We give an example of $x^{-1}L$ below.

Example 2.1.15 Let $L = \{bbbab, abab, abaab, abababa\}$ and $x = aba$.
Then, $x^{-1}L = \{b, ab, baba\}$. ◊

In effect, $x^{-1}L$ gives all words in L that have a prefix of x, without the prefix x. Note that if x is not a prefix of L, this set will be empty.

2.2 Regular Expressions

In the previous section, we defined the language operations union, concatenation, and the Kleene closures. Each of these operations is called a *regular operation*. A language L over Σ is called *regular* if it can be obtained from the empty language $\emptyset$, λ, and a, where a is any symbol from the alphabet Σ, by applying regular operations finitely many times. Applying union and concatenation, we can obtain all finite languages. The star operation is needed to produce infinite languages.

In [16], Kleene introduced a way to generate regular languages. A *regular expression* e over an alphabet Σ and the language $L(e)$ that it generates are defined recursively as follows:

1. $\emptyset$ is a regular expression denoting the language $L(\emptyset) = \emptyset$.
2. λ is a regular expression denoting the language $L(\lambda) = \{\lambda\}$.
3. For every $a \in \Sigma$, a is a regular expression and $L(a) = \{a\}$.
4. If p and q are regular expressions, then $p+q$, $p \cdot q$, and p^* are regular expressions and $L(p + q) = L(p) \cup L(q)$, $L(p \cdot q) = L(p)L(q)$, and $L(p^*) = (L(p))^*$.

Note that the language $L(a^*)$ where $a \in \Sigma$ is the language $L(a^*) = \{\lambda, a, aa, aaa, ...\}$. Also, the operation $*$ has higher precedence than $\cdot$, which has higher precedence than $+$, respectfully. By convention, the symbol $\cdot$ is usually omitted and pq is written instead of $p \cdot q$. We give a few examples of regular expressions and the languages that they generate below.

Example 2.2.1 Let $\Sigma = \{a, b\}$. Then,

$$(ab + a)(bb + \lambda) = \{abbb, ab, abb, a\}$$
$$(\lambda + a)^* b = \{b, ab, aab, aaab, aaaab, aaaaab, \dots\}$$
$$aa(a + b)^* = \{w \in \Sigma^* \mid \text{ the first two letters of } w \text{ are } a\}.$$

$\Diamond$

2.3 Finite State Automata

Our goal for this section is to characterize regular languages using structures similar to directed graphs, called automata. An automaton is an object that *recognizes* words and determines whether or not an input word belongs to a language. It *accepts* the words that belong to the language and *rejects* the words that don't. A *finite state automaton* (FSA) is a quintuple $M = (Q, \Sigma, I, T, \mathcal{E})$, where Q is a finite set of states, Σ is the input alphabet, $I \subseteq Q$ is the set of initial states, $T \subseteq Q$ is the set of terminal states, and $\mathcal{E} \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q$ is the set of transitions. The alphabet can be omitted if it is understood to be the set of symbols that can be seen on the labels. In this way, the FSA M can alternatively be denoted by the quadruple $(Q, I, T, \mathcal{E})$.

Example 2.3.1 Let

$$M = (\{q_0, q_1, q_2, q_3\}, \{q_0\}, \{q_3\}, \{(q_0, a, q_1), (q_1, a, q_2), (q_0, b, q_2), (q_2, b, q_2), (q_2, a, q_3)\}).$$

The FSA M is represented in Figure 2.1. $\Diamond$

If e is a non-empty transition in $\mathcal{E}$, then $e = (q, a, q')$, where $q, q' \in Q$ and $a \in \Sigma$. Further, q is the *source* of e and is denoted by $s(e) = q$; q' is the *target* of e and is denoted by $t(e) = q'$; and a is the *label* of e and is denoted by $l(e) = a$. The transition (q, λ, q') is called the *empty transition.*

A *path* or *transition sequence* in M is a sequence of transitions $p = e_1 e_2 e_3 \dots e_n = (q_0, a_1, q_1)(q_1, a_2, q_2)\dots(q_{n-1}, a_n, q_n)$ such that $s(e_{k+1}) = t(e_k)$ for $k = 1, 2, \dots, n-1$. Note that a path is a word in $\mathcal{E}^*$. We denote the *set of paths of* M with $\mathcal{P}$. For a

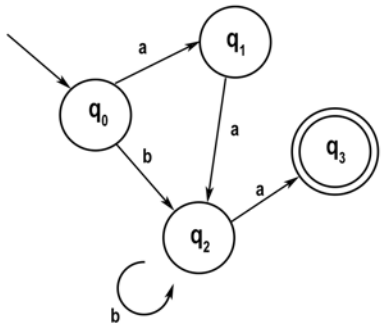

Figure 2.1: The graph of the FSA M from Example 2.3.1.

path $p = e_1e_2 \ldots e_n$, the source of p is $s(p) = s(e_1)$, the target of p is $t(p) = t(e_n)$, and the label of p is $l(p) = l(e_1)l(e_2) \ldots l(e_n)$.

A word $w \in \Sigma^*$ is *accepted* by M if there is a path p such that $s(p) \in I, t(p) \in T$, and $l(p) = w$. The language *recognized* by M is $L(M) = \{w \mid w \text{ is accepted by} M\}$. We introduce the concept of a *strongly connected component* (or SCC) with respect to an FSA M. A SCC is a set of states Q that is maximal with the property that there is a path in M between any pair of states in Q. An SCC Q is called *nontrivial* if there is at least one transition between some states in Q. In Figure 2.2, we illustrate the difference between components of a graph that are strongly connected and components that aren't.

Example 2.3.2 Consider Figure 2.2. The sets of states $Q_1 = \{q_0, q_1, q_2\}$ and $Q_2 = \{q_3, q_4\}$ are SCCs; however, $Q_3 = \{q_1, q_3, q_4\}$ is not an SCC. This is because there is no path from state q_3 to state q_1. $\Diamond$

A state in an automaton is called a *fork state* if there are at least two transitions going out of that state. In other words, if q is fork state, then there are at least two transitions that begin in q and end in a state other than q. In Figure 2.3, every state in the FSA is a fork state except for state q_3. This is because q_3 does not have at least two transitions going out of it. A *cycle* is a path in an FSA in which the first and last states of the path are equal. We call the special cycle (p) where p is any state the *trivial cycle*.

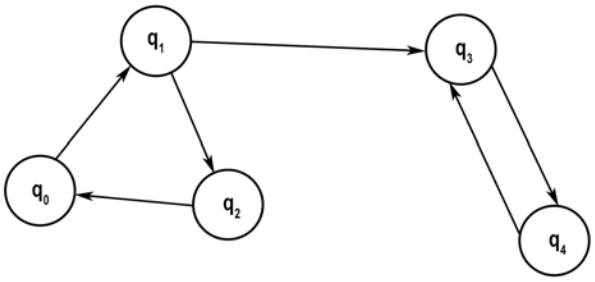

Figure 2.2: An automaton with two SCC.

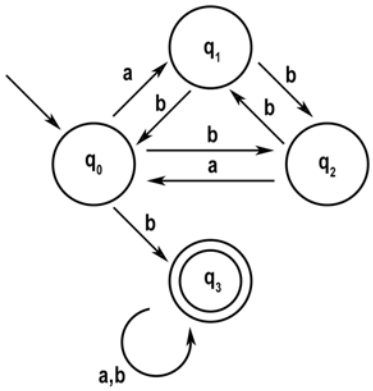

Figure 2.3: The states q_0, q_1, and q_2 are all fork-states, while q_3 is not.

Remark 2.3.3 A language L is regular if and only if there exists an FSA M such that $L = L(M)$.

By Remark 2.3.3, for every regular language there exists an FSA that recognizes it. The proof of Remark 2.3.3 can be found in [17]. We would like to show that regular languages are also recognized by a certain subtype of FSAs. Theorem 2.3.6, originally introduced in [16], accomplishes this. The proof of Theorem 2.3.6 requires two other results.

Theorem 2.3.4 *Every FSA is equivalent to an FSA which does not have any empty transitions.*

Proof. Let $M = (Q, I, T, \mathcal{E})$ be an FSA and $\mathcal{P}$ be the set of all paths in M. Construct $M' = (Q, I', T', \mathcal{E}')$ as follows.

$$\begin{aligned} I' &= I \cup \{q \in Q \mid \exists p \in \mathcal{P}, s(p) \in I, t(p) = q, l(p) = \lambda\} \\ T' &= T \cup \{q \in Q \mid \exists p \in \mathcal{P}, s(p) = q, t(p) \in T, l(p) = \lambda\} \\ \mathcal{E}' &= \{(q, a, q') \mid \exists p \in \mathcal{P}, s(p) = q, t(p) = q', l(p) = a, a \in \Sigma\} \\ &\quad \setminus \{(q, \lambda, q') \in \mathcal{E} \mid q, q' \in Q\} \end{aligned}$$

We need to show that $L(M) = L(M')$ by first showing that $L(M) \subseteq L(M')$. Let $p = e_1e_2e_3...e_k$ be a path in M with $s(p) \in I$, $t(p) \in T$, and $l(p) = a_1a_2a_3...a_s$ in $L(M)$. (If $l(p) = \lambda$, then $t(e_k) \in I' \cap T$ and $\lambda \in L(M')$). We need to show that p is a path in M' from an initial state to a terminal state. Let $e_{j_1}e_{j_2}...e_{j_s}$ be a "subpath" of p such that $l(e_{j_i}) = a_i$ and let $s(e_{j_i}) = p_{j_i}$. Then by construction $p_{j_1} \in I'$, $t(e_{j_s}) \in T'$, and $e'_{j_i} = (p_{j_i}, a_i, p_{j_{i+1}}) \in \mathcal{E}'$ for all $i = \{1, ..., s-1\}$ and $e_{j_s} \in \mathcal{E}'$. Hence, the path $p' = e'_{j_1}e'_{j_2}...e'_{j_{s-1}}e_{j_s}$ is a path in M' from an initial state in M' to a terminal state in M' with label $l(p') = l(p)$. Thus, $L(M) \subseteq L(M')$.

Conversely, assume that $w \in L(M')$. Then, there is a path $p = e_1e_2e_3...e_k$ in M' from an initial state in M' to a terminal state in M' with label $l(p) = w$. Every edge $e_i \in p$ is a transition of the kind (q, a, q') where $a \in \Sigma$ and $q, q' \in Q$. By the

construction of M', every such transition can be substituted by a path with label a from q to q' in M. Thus, there is a path with label w in M and it can be chosen from initial to a terminal state by concatenating a prefix path and a suffix path both with label λ, if necessary. ■

We have established that it is possible to take an FSA and transform it into an equivalent FSA that has no empty transitions. We use this result to show that any FSA is equivalent to an FSA with no empty transitions and only a single initial state.

Theorem 2.3.5 *Every FSA is equivalent to an FSA with no empty transition and only one initial state.*

Proof. By Theorem 2.3.4, we can assume that the given $M = (Q, I, T, \mathcal{E})$ has no empty transitions and it is possible to show that it is equivalent to an automaton with only one initial state and no empty transitions. We do this by letting $M' = (Q', I_0, T', \mathcal{E}')$ where $Q' = Q \cup \{q_0\}$, $I_0 = \{q_0\}$, such that $q_0 \notin Q$ and $\mathcal{E}' = \mathcal{E} \cup \{(q_0, a, q) \mid \exists q_I \in I, (q_I, a, q) \in \mathcal{E}\}$. The terminal states $T' = T$ if $\lambda \notin L(M)$ and $T' = T \cup \{q_0\}$ if $\lambda \in L(M)$. We need to show that $L(M) = L(M')$. First we show that $L(M) \subseteq L(M')$.

Assume $w \in L(M)$. Say $w = \lambda$, if this is the case then $w \in L(M')$ as well, because M' has only one initial state q_0 and $q_0 \in T'$. Now say that $w \neq \lambda$. Then, there is a path $p = e_1e_2e_3...e_k$ in M with $s(p) \in I$, $t(p) \in T$, and $l(p) = w = a_1a_2a_3 \ldots a_k$ in $L(M)$. We need to show that p is a path in M' from an initial state to a terminal state. Let $p_1 = e_2e_3e_4...e_k$ be the "subpath" of p without the transition e_1. Thus, we have that $t(p_1) \in T$ and $l(p_1) = a_2a_3a_4 \ldots a_k$. Consider e_1 and e_2, By assumption $e_1 = (q_I, a_1, q_1)$ and $e_2 = (q_1, a_2, q_2)$ where $q_I \in I$ and $q_1, q_2 \in Q$. Allow $e_1' = (q_0, a_1, q_1)$. By construction, $e_1' \in \mathcal{E}'$. Then, $p' = e_1'p_1$ is a path in M' with $s(p') \in I_0 = \{q_o\}$, $t(p') \in T'$. Therefore, p' is a path in M' that starts in an initial state and ends in a terminal state with $l(p) = l(p') = w$. So $L(M) \subseteq L(M')$.

Conversely, let $w \in L(M')$. Then, there is a path $p' = e'_1 e_2 e_3 \dots e_k$ with $s(p') \in I_0$, $t(p') \in T'$, and $l(p') = w = a_1 a_2 a_3 \dots a_k$. We have that $e'_1 = (q_0, a_1, q_1)$ and $e_2 = (q_1, a_2, q_2)$. By construction, there exists $q_I \in I$ and $(q_I, a_1, q_1) \in \mathcal{E}$. Allow $p = e_1 e_2 e_3 \dots e_k$. We have that p is a path in M that begins in an initial state and ends in a terminal state with $l(p) = l(p') = w$. So $L(M) \supseteq L(M')$. Thus, $L(M) = L(M')$. ∎

By Theorems 2.3.4 and 2.3.5, we can assume that given an FSA M, it has only one initial state and no empty transitions. An FSA for which the next state is always uniquely determined is called a *deterministic finite state automaton* (DFA). Also, an FSA is called *complete* if for all $a \in \Sigma$ and for all $q \in Q$ the set $\{q' \mid (q, a, q') \in \mathcal{E}\}$ is either singleton or empty. Theorem 2.3.6 establishes a connection between regular languages and DFAs.

Theorem 2.3.6 *A language L is regular if and only if it is recognized by a DFA.*

Proof. It is clear that if a language is recognized by a DFA, then (since a DFA is an FSA) it is regular. We only need to show that if L is a regular language, then there is a DFA that recognizes it. Since we know that if L is regular, then there exists an FSA that recognizes it, we only need to show that for every FSA, there is an equivalent DFA.

Let $M = (Q, \Sigma, I, T, \mathcal{E})$ be an FSA. By Theorem 2.3.5, we can assume that M has only one initial state, i.e. $I = \{q_0\}$ and no empty transitions. Construct $M' = (M', \Sigma, I', T', \mathcal{E}')$ as follows. The new set of states $Q' = \mathcal{P}(Q)$ is the power set of Q. The initial set of states $I' = \{I\} = \{\{q_0\}\}$ consist only of one initial state, namely the unique initial state of M. The terminal states T' are all subsets of Q that contain a terminal state in M, i.e., $S \subset Q$ is terminal in M' if $S \cap T \neq \emptyset$. The set of transition functions is defined as follows:

$$\begin{aligned}(S, a, S') \in \mathcal{E}' \text{ if and only if } S' &= \{q' \mid \exists q \in S, (q, a, q') \in \mathcal{E}\} \\ &= \{t(e) \mid s(e) \in S, l(e) = a, e \in \mathcal{E}\}.\end{aligned}$$

Since S and a uniquely determine S', M' is deterministic. Hence each transaction in $\mathcal{E}'$ can be expressed by a transition function δ. We now have to show that every word accepted by M is accepted by M' and vice versa.

Let $w = a_1a_2a_3...a_k$ be a word accepted by M. Then, there is a path $(q_0, a_1, q_1)(q_1, a_2, q_2)\dots(q_{k-1}, a_k, q_k)$ with $q_k \in T$ and by the definition of M', $q_1 \in \delta(\{q_0\}, a_1) = S_1$, $q_2 \in \delta(\{S_1\}, a_2) = S_2$, $\dots$, $q_k \in \delta(\{S_{k-1}\}, a_k) = S_k$, and $S_k \in T'$. Therefore, $L(M) \subseteq L(M')$.

Conversely, let $w \in L(M')$. Then there is a path $p = e_1e_2...e_k$ in M' with $l(p) = w$ from state $\{q_0\}$ to a terminal state $S_k \in T'$. Let S_i be the terminal states $t(e_i)$ of e_i. The result is a state $q_k \in S_k \cap T$ and a transition $(q_{k-1}, a_k, q_k) \in \mathcal{E}$ with $q_{k-1} \in S_{k-1} = \{t(e_{k-1}) \mid t \text{ is a terminal state of } e_{k-1}\}$. By induction, for each $i = \{0, 1, ..., k-1\}$ there is a transition $(q_i, a_{i+1}, q_{i+1}) \in \mathcal{E}$ such that $q_i \in S_i$ and $q_{i+1} \in S_{i+1}$. Therefore, $w \in L(M)$. ■

Chapter 3

Subword Closure

In this chapter we define the concept of the subword closure and discuss a method for generating automata that accept the subword closure. We provide examples of such automata and briefly consider the implications of such a method. The rest of the chapter describes a method for finding languages D that are maximal with the property that the $*$-Kleene Closure, D^*, of that language D is contained within the subword closure, $S'^{\otimes}$, of some given subword constraint S. This method makes use of the automaton that accepts the subword closure of S. We provide theorems that prove that the method discussed ensures that such a language D can be made to be nonempty or nontrivial.

3.1 Subword Closure and Trie

Any nonempty finite set of words S of length $k > 0$ over Σ is called a *subword constraint*, and is used to define $S^{\otimes}$, the *subword closure* of S. The subword closure of S is the set of words for which all subwords of length k are in S. We give a concise definition using set notation below:

$$S^{\otimes} = \{w \in \Sigma^* \mid \text{ if } u \text{ is a subword of } w \text{ and } |u| = k \text{ then } u \in S\}$$

It is trivial to see that $S' \subseteq S$ if and only if $S'^{\otimes} \subseteq S^{\otimes}$. We have that if $S' \neq S$, then $S'^{\otimes} \neq S^{\otimes}$.

Example 3.1.1 Let $S = \{aa, ba, ab\}$, then

$$S^{\otimes} = \{w \in \Sigma^* \mid w \text{ does not have consecutive } b\text{'s }\}.$$ $\Diamond$

The subword closure has two properties that we should consider.

Remark 3.1.2 Let S be a subword constraint where each word in S is of length k. We have that:

- If $w \in S$, then every subword of w is also in $S^{\otimes}$.
- If $xu, vy \in S^{\otimes}$ and $|u| = |v| = k$, then we have that $xuvt \in S^{\otimes}$ if and only if $uv \in S^{\otimes}$.

We demonstrate a method that, given a subword constraint S, will construct a DFA accepting $S^{\otimes}$. We first define a special type of DFA, a *trie*.

Definition 3.1.3 A *trie*, T, is a DFA with the following structure:

$$T = (\{[p] \mid p \in Pref(L)\}, \Sigma, [\lambda], \{[x] \mid x \in L\}, \mathcal{E})$$

where L is a finite language, $Pref(L)$ is the set of all prefixes of L and the set of transitions $\mathcal{E}$ is given by:

$$\mathcal{E} = \{[p]a \to [pa] \mid p \in Pref(L), a \in \Sigma, pa \in Pref(L)\}.$$

For clarity, we adopt the convention that "u" refers to the word u and "$[u]$" refers to the state with label u. We can extend this convention to sets of states. Thus, if L is a language, $[L] = \{[x] \mid x \in L\}$. Observe that each state $[p]$ represents the prefix p of the input word that has been read so far by the automaton. Also observe that there is a path from $[\lambda]$ to $[x]$ for each $x \in L$ and each such $[x]$ is a terminal state. This implies that the trie accepts L.

3.2 Subword Closure as an Automaton

Let $Trie(S)$ be the *trie* recognizing the language S, where S is a subword constraint with the length of each word being some fixed k. $Trie(S)$ is the complete DFA with $Trie(S) = (Q, I, T, \mathcal{E})$ where each component is defined in the way outlined in Definition 3.1.3. For clarity, we represent $\mathcal{E}$ as a function δ such that $\delta([u], \sigma) = [u\sigma]$, whenever $u\sigma$ is a prefix of S of length at most k and with all other values of δ being $[sink]$. We now prove that there is a DFA that accepts $S^{\otimes}$. The following lemma is taken from [13].

Lemma 3.2.1 *Let T be a trie accepting only words of the same length. There is a DFA $T^{\otimes}$ accepting the language $L(T)^{\otimes}$.*

Proof. For a word w of the form aw_1, with $a \in \Sigma$, we denote by $\dot{w}$ the word w_1. The states $T^{\otimes}$ are exactly those of T and the initial and terminal states of $T^{\otimes}$ are exactly those of T as well. The DFA $T^{\otimes}$ contains all the transitions of T with the addition of: For each terminal state $[w]$ of T and for each $a \in \Sigma$, then $\delta^{\otimes}([w], a) = [\dot{w}a]$ if and only if $[\dot{w}a]$ is a terminal state of T. We show that $L(T)^{\otimes} \subseteq L(T^{\otimes})$. The proof of the converse is trivial to see.

First note that every word $w \in L(T)^{\otimes}$ is of the form $ua_1 \dots a_m$ with $|u| = k$, $m \geq 0$, and each $a_i \in \Sigma$. Let $u_0 = u$ and $u_i = \dot{u}_{i-1}a_i$ for all $i = 1, \dots, m$. Then, for each index i, the word u_i is in $Sub_k(w)$, which implies that u_i is in $L(T)$. Thus, on input w the DFA $T^{\otimes}$ will behave as follows: the prefix u of w will take $T^{\otimes}$ to the terminal state $[u]$ and, in general, if $ua_1 \dots a_{i-1}$ takes $T^{\otimes}$ to the terminal state $[u_{i-1}]$, then, as $\dot{u}_{i-1}a \in L(T)$, the prefix $ua_1 \dots a_i$ of w would take $T^{\otimes}$ to the terminal state $[u_i]$. Thus, w will be accepted by $T^{\otimes}$ and, therefore, $w \in L(T^{\otimes})$. ∎

By [3], we have that $Trie(S)^{\otimes}$ is a complete DFA. For an example of $Trie(S)$, consider the following.

Example 3.2.2 Let $S = \{aaaa, aaab, aaba, abaa, abbb, baaa, baba, babb, bbab, bbbb\}$. $Trie(S)$ is the complete DFA with $Trie(S) = (Q, I, T, \mathcal{E})$ where $Q = [Pref(S)]$,

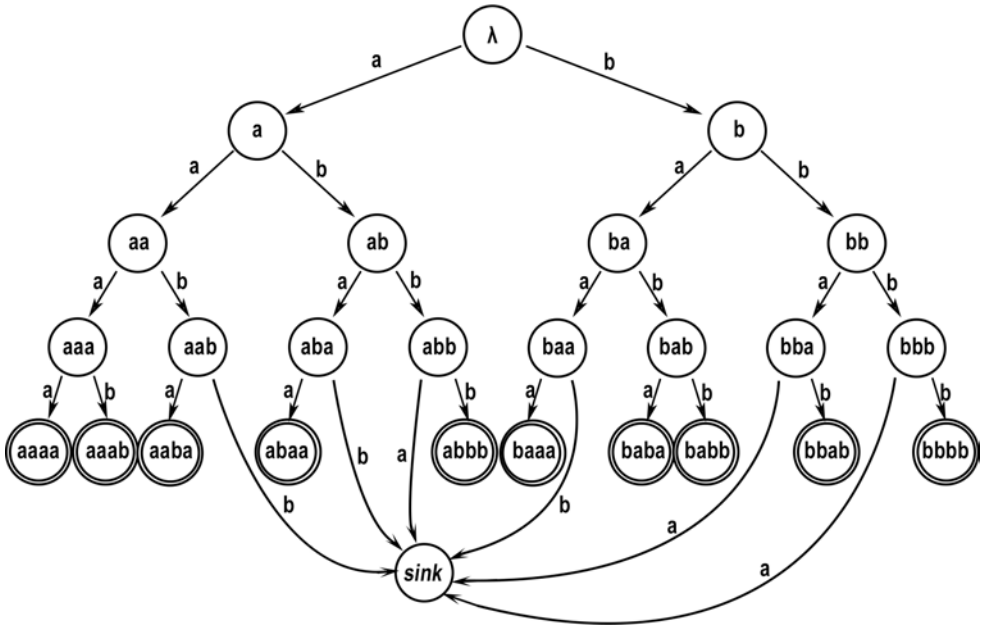

Figure 3.1: Graph of $Trie(S)$.

$I = [\lambda]$, $T = [S]$, and $\mathcal{E}$ is represented by the transition function δ where $\delta([u], \sigma) = [u\sigma]$, whenever $u\sigma$ is a prefix of S of length at most k and with all other values of δ being $[sink]$. The complete DFA $Trie(S)$ is represented in Figure 3.1. $\Diamond$

We demonstrate how $\mathcal{E}$ is populated. Consider $([aba], a, [abaa]) \in \mathcal{E}$. Allowing $u = aba$ and $\sigma = a$, we can see that $\delta([aba], a) = [abaa]$. This is because $abaa$, or $u\sigma$, is a prefix of S of length four. Consider $([aab], b, [sink]) \in \mathcal{E}$. Allowing $u = aab$ and $\sigma = b$. We can see that $u\sigma = aabb$ is not a prefix of S of length at most four. For this reaseon, we create a transition from $[aab]$ to $[sink]$ and give it the label b.

Following the construction outlined in Lemma 3.2.1 the DFA $Trie(S)^{\otimes}$ accepting $S^{\otimes}$ is obtained from $Trie(S)$ like so: the set of states, Q, is the same; the initial state, I, is the same; the set of terminal states, T, now includes all states except for $[sink]$; the transition function $\delta^{\otimes}$ of $Trie(S)^{\otimes}$ is the same as δ except for the following: For each $u \in S$ and $\sigma \in \Sigma$, if $u \in \Sigma u_1$ and $u_1\sigma \in S$, then $\delta^{\otimes}([u], \sigma) = [u_1\sigma]$. In effect, this change to δ ensures that the last k symbols read drive the automaton to a state in $[S]$. The example below demonstrates the DFA $Trie(S)^{\otimes}$ obtained from $Trie(S)$ given in Example 3.2.2.

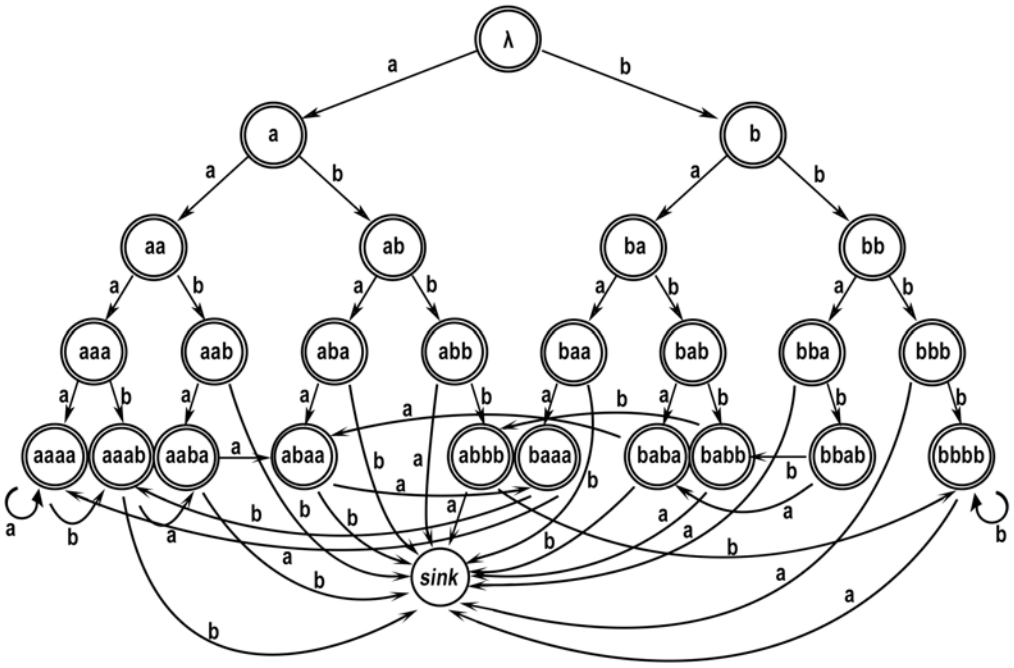

Figure 3.2: Graph of $Trie(S)^{\otimes}$

Example 3.2.3 Let $S = \{aaaa, aaab, aaba, abaa, abbb, baaa, baba, babb, bbab, bbbb\}$. $Trie(S)^{\otimes}$ is the complete DFA with $Trie(S)^{\otimes} = (Q, I, T', \mathcal{E}')$ where Q is the same as in Example 3.2.2, $I = [\lambda]$, $T' = Q \setminus \{[sink]\}$, and the transition function $\delta^{\otimes}$ of $Trie(S)^{\otimes}$ is represented by the edges of the DFA given in Figure 3.2. $\Diamond$

We demonstrate how $\mathcal{E}'$ is obtained from $\mathcal{E}$. Consider the transition $([aaba], a, [abaa]) \in \mathcal{E}'$. Allowing $u = aaba$, $\sigma = a$, and $u_1 = aba$ we can see that $aaba \in \Sigma aba$ and $abaa \in S$, therefore, $\delta^{\otimes}([aaba], a) = [abaa]$. We are effectively concatenating u and σ, obtaining a new word $u_1\sigma$ by dropping all but the last k symbols (four in this case), and then creating a transition between $[u]$ and the resulting $[u_1\sigma]$ with label σ.

Remark 3.2.4 We give a few useful properties of $Trie(S)^{\otimes}$ below.

– If $([u], \sigma_1, p_1, \ldots, \sigma_k, p_k)$ is a path in $Trie(S)^{\otimes}$ and $p_k \neq [sink]$ then the state p_k must be $[\sigma_1 \ldots \sigma_k]$.

– If $w \in S^{\otimes}$ and $|w| \geq k$, then $\delta^{\otimes}([\lambda], w) = \delta^{\otimes}([x], w_1) = [y]$, where x is the prefix of w of length k and y is the suffix of w of length k.

– The DFA $Trie(S)^{\otimes}$ can be computed in linear time with respect to the size of S, this size is the sum of the lengths of all words in that set.

– Any nontrivial SCC $[Q]$ of $Trie(S)^{\otimes}$ is such that $Q \subseteq S$.

We use $Trie(S)^{\otimes}$ to characterize structurally the languages D that are maximal with the property that $D^* \subseteq S^{\otimes}$. We will refer to the properties discussed in Remark 3.2.4 to accomplish this.

Chapter 4

Maximal D's With $D^* \subseteq S^{\otimes}$

In this chapter we discuss both a method for structurally characterizing a nonempty language D and a method for structurally characterizing a nontrivial language D', both of whose words are of length at least k and are maximal with the property with $D^* \subseteq S^{\otimes}$ (and $D'^* \subseteq S^{\otimes}$). Each method was introduced by Konstantinidis and Santean in [18]. Recall that a strongly connected component with respect to a DFA M, is a set of states Q that is maximal with the property that there is a path in M between any pair of states in Q.

4.1 Preliminaries

Definition 4.1.1 Let Q be a nonempty subset of S such that $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$. For any nonempty subset $X \subseteq Q$, we define

$$Q_X^{>} \triangleq \{v \in Q \mid \forall x \in X : \delta^{\otimes}([v], x) = [x]\} = \{v \in Q \mid \forall x \in X : vx \in S^{\otimes}\}.$$

By Definition 4.1.1, it can be understood that if $X \subseteq Y \subseteq Q$, then $Q_X^{>} \supseteq Q_Y^{>}$. The set $Q_X^{>}$ can be thought of as the set of states in $[Q]$ for which any input word $x \in X$ drives the automaton $Trie(S)^{\otimes}$ to the state $[x]$. We can simplify Figure 3.2 by eliminating all states $[x]$ such that $[x] \notin [S]$, except for $[sink]$. Although this means that the resulting graph is no longer a DFA, because we eliminate the initial state, this process can be reversed with no major effect. The graph obtained by

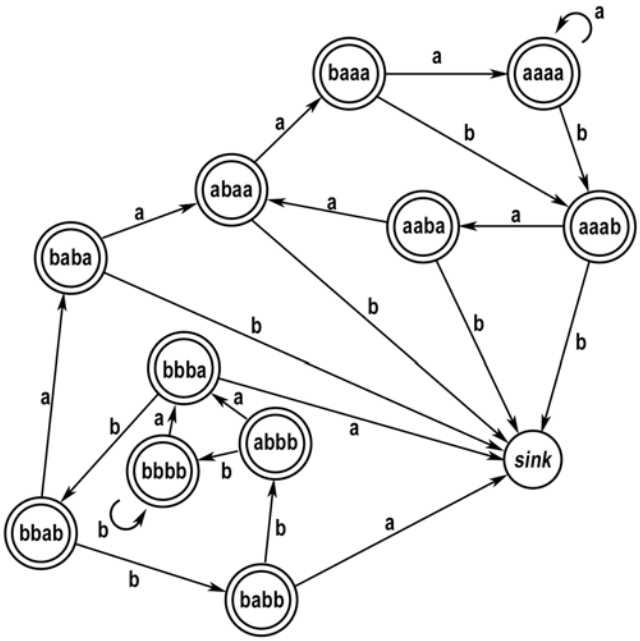

Figure 4.1: Simplified graph obtained by eliminating all states that are not in S (except for the sink) from Figure 3.2.

eliminating these states can be seen in Figure 4.1.

Example 4.1.2 Consider the simplified graph given in Figure 4.1. Let $Q = \{aaaa, aaab, aaba, abaa, baaa\}$. We determine $Q^{>}_{\{abaa\}}$ by considering each state in $[Q]$ individually.

- $[aaaa]$: Since $\delta^{\otimes}([aaaa], abaa) = [abaa]$ and $abaa \in \{abaa\}$, we have that $aaaa \in Q^{>}_{\{abaa\}}$.
- $[aaab]$: Since $\delta^{\otimes}([aaab], abaa) = [sink]$ and $sink \notin \{abaa\}$, we have that $aaab \notin Q^{>}_{\{abaa\}}$.
- $[aaba]$: Since $\delta^{\otimes}([aaba], abaa) = [sink]$ and $sink \notin \{abaa\}$, we have that $aaba \notin Q^{>}_{\{abaa\}}$.
- $[abaa]$: Since $\delta^{\otimes}([abaa], abaa) = [abaa]$ and $abaa \in \{abaa\}$, we have that $abaa \in Q^{>}_{\{abaa\}}$.

– $[baaa]$: Since $\delta^{\otimes}([baaa], abaa) = [abaa]$ and $abaa \in \{abaa\}$, we have that $baaa \in Q^{>}_{\{abaa\}}$.

Thus, we have that $Q^{>}_{\{abaa\}} = \{abaa, baaa, aaaa\}$. $\Diamond$

Example 4.1.2 considered only when X is a singleton set. The following example considers when X is more complex.

Example 4.1.3 Consider the simplified graph M given in Figure 4.1, let $Q = \{aaaa, aaab, aaba, abaa, baaa\}$ and $X = \{baaa, abaa\}$. We determine $Q^{>}_{X}$ by considering each state in $[Q]$ individually.

– $[aaaa]$: $\delta^{\otimes}([aaaa], baaa) = [baaa]$, and $\delta^{\otimes}([aaaa], abaa) = [abaa]$.
Since $baaa, abaa \in X$, we have that $aaaa \in Q^{>}_{X}$.

– $[aaab]$: $\delta^{\otimes}([aaab], baaa) = [sink]$, and $\delta^{\otimes}([aaab], abaa) = [sink]$.
Since $sink \notin X$, we have that $aaab \notin Q^{>}_{X}$.

– $[aaba]$: $\delta^{\otimes}([aaba, baaa) = [sink]$, and $\delta^{\otimes}([aaba], abaa) = [sink]$.
Since $sink \notin X$, we have that $aaba \notin Q^{>}_{X}$.

– $[abaa]$: $\delta^{\otimes}([abaa], baaa) = [sink]$, and $\delta^{\otimes}([abaa], abaa) = [abaa]$.
Since $sink \notin X$, we have that $abaa \notin Q^{>}_{X}$.

– $[baaa]$: $\delta^{\otimes}([baaa], baaa) = [baaa]$, and $\delta^{\otimes}([baaa], abaa) = [abaa]$.
Since $baaa, abaa \in X$, we have that $baaa \in Q^{>}_{X}$.

Thus, we have that $Q^{>}_{X} = \{baaa, abaa\}$. $\Diamond$

The maximal languages D that we desire can be obtained from $Q^{>}_{X}$. In particular, we are interested in the words that drive the automaton to a state in $Q^{>}_{X}$.

Definition 4.1.4 Let $[Q]$ be a strongly connected component of $Trie(S)^{\otimes}$, with $Q \subseteq S$ and $X \subseteq Q$, then

$$\langle Q, X \rangle_x \triangleq \{w \in \Sigma^* \mid \delta^{\otimes}([x], w) \in [Q^{>}_{X}]\}$$

for all $x \in X$.

In effect, $\langle Q, X\rangle_x$ is the set of words $w \in \Sigma^*$ that, starting in state $[x]$ drive the automaton to a state in $[Q_X^>]$, for all $x \in X$. The set $\langle Q, X\rangle_x$ is generally written as a regular expression, we give an example of this below.

Example 4.1.5 Consider the simplified graph M given in Figure 4.1, let $Q = \{aaaa, aaab, aaba, abaa, baaa\}$ and $X = \{baaa, abaa\}$. Recall from Example 4.1.3, that $Q_X^> = \{baaa, aaaa\}$. We have that

$$\langle Q, X\rangle_{\{baaa\}} = (\lambda + aa^*)(baaa(\lambda + aa^*))^*$$

$$\langle Q, X\rangle_{\{abaa\}} = a(\lambda + aa^*)(baaa(\lambda + aa^*))^*. \qquad \Diamond$$

We define a language that, as we will see later, is a major structural characterization of languages D that are maximal with the property $D^* \subseteq S^{\otimes}$.

Definition 4.1.6 Let $[Q]$ be a strongly connected component of $Trie(S)^{\otimes}$, with $Q \subseteq S$ and $X \subseteq Q$, then

$$\langle Q, X\rangle \triangleq \bigcup_{x \in X} x\langle Q, X\rangle_x$$

for all $x \in X$.

Building on Examples 4.1.3 and 4.1.5, we can obtain an example of a language $\langle Q, X\rangle$.

Example 4.1.7 Consider the simplified graph M given in Figure 4.1, let $Q = \{aaaa, aaab, aaba, abaa, baaa\}$ and $X = \{baaa, aaaa\}$. Recall from Example 4.1.5 that

$$\langle Q, X\rangle_{\{baaa\}} = (\lambda + aa^*)(baaa(\lambda + aa^*))^*$$

and

$$\langle Q, X\rangle_{\{abaa\}} = a(\lambda + aa^*)(baaa(\lambda + aa^*))^*.$$

We have that

$$\langle Q, X\rangle = baaa(\lambda + aa^*)(baaa(\lambda + aa^*))^* + abaaa(\lambda + aa^*)(baaa(\lambda + aa^*))^*. \qquad \Diamond$$

4.2 Obtaining maximal D's with $D \subseteq S^{\otimes}$

In this section, we present two theorems that prove that if $D = \langle Q, X \rangle$, then D is maximal with $D^* \subseteq S^{\otimes}$. The first theorem considers when D is nonempty and the second considers when D is nontrivial. These theorems also show that each word in D has a prefix in X and a suffix in $Q_X^>$. Both theorems were originally presented by Konstantinidis and Santean in [18]. We state the theorems first and leave the proofs of each for later on.

Theorem 4.2.1 *Let S be any subword constraint of some length k, and let D be any nonempty language whose words are of length k. Then D is maximal with $D^* \subseteq S^{\otimes}$ if and only if there are nonempty subsets X, Y, Q of S such that*

$$D = \langle Q, X \rangle = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y,$$

and $X, Y \subset Q$, and $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$, $Y = Q_X^>$.

Theorem 4.2.2 accounts for D being nontrivial by requiring that Q contain a fork state. Recall from earlier, that a state in an automaton is called a fork state if there are at least two transitions going out of that state.

Theorem 4.2.2 *Let S be any subword constraint of some length k, and let D be any nontrivial language whose words are of length k. Then, D is maximal with $D^* \subseteq S^{\otimes}$ if and only if there are nonempty subsets X, Y, Q of S such that*

$$D = \langle Q, X \rangle = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$$

and $X, Y \subseteq Q$, $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$ containing a $[Q]$-fork state, $Y = Q_X^>$, and X is maximal with "$X \subseteq Q$ and $Q_X^> = Y$".

Note that 'X is maximal with "$X \subseteq Q$ and $Q_X^> = Y$" can be rephrased as "for any $z \in Q - X, Q_X^> \neq Q_{X \cup \{z\}}^>$." The proofs of Theorems 4.2.1 and 4.2.2 can be

obtained from a few technical lemmata. Lemma 4.2.3 gives an impression of what $D^* \subseteq S^{\otimes}$ means without requiring that D is maximal with that property. The lemma provides two conditions for $D^* \subseteq S^{\otimes}$, one is a necessary condition and the other is a sufficient condition. For this lemma, recall Definition 2.1.14, the special notation that we defined earlier, $x^{-1}L = \{z \in \Sigma^* \mid xz \in L\}$.

Lemma 4.2.3 *Let D be a nonempty language whose words are of length at least k.*

1. *If $D^* \subseteq S^{\otimes}$, then $D = \bigcup_{x \in X} x(x^{-1}D)$ and $x(x^{-1}D)y \subseteq S^{\otimes}$, for all $x, y \in X$ where X is the set of all prefixes of D of length k.*

2. *If there is a subset X of S and languages D_x, for all $x \in X$ such that $D = \bigcup_{x \in X}(xD_x)$ and $xD_xy \subseteq S^{\otimes}$ for all $x, y \in X$, then $D^* \subseteq S^{\otimes}$.*

Proof. For the first statement, consider $\bigcup_{x \in X} x(x^{-1}D)$. By definition of $x^{-1}D$, $\bigcup_{x \in X} x(x^{-1}D) = \bigcup_{x \in X} xz$, where $z \in \Sigma^*$ and $xz \in D$. In other words, it is equal to the union of all of the words in D that have a word $x \in X$ as a prefix. Since X is the set of all prefixes of D, this is simply the set D. Assume $D^* \subseteq S^{\otimes}$, and take any arbitrary $x, y \in X$ and $u \in x^{-1}D$. Then, $xu \in D$ by definition. Similarly, there exists a word of the form $yv \in D$, with $v \in y^{-1}D$. Since $xuyv \in D^2 \subseteq D^* \subseteq S^{\otimes}$, we have that $xuyv \in S^{\otimes}$. Hence, by Remark 3.1.2, $xuy \in S^{\otimes}$, as required.

For the second statement, we show $D^* \subseteq S^{\otimes}$ by proving $D^n \subseteq S^{\otimes}$ for all $n \geq 0$, using induction on n. First, assume there is a set $X \subseteq S$, and languages D_x, for all $x \in X$ such that $D = \bigcup_{x \in X}(xD_x)$ and $xD_xy \subseteq S^{\otimes}$ for all $x, y \in X$. Obviously, $D^0 \subseteq S^{\otimes}$, as $D^0 = \{\lambda\}$. For any $n \geq 1$, assume that $D^{n-1} \subseteq S^{\otimes}$. Consider any word $w \in D^n$, w can be written as $xuv \in S^{\otimes}$, with $x \in X$, $u \in D_x$, and $v \in D^{n-1}$. This is because $xu \in D$, by assumption. Note here that, as $xuy \in S^{\otimes}$ for all $x, y \in X$ and $u \in D_x$, replacing y by x gives us $xux \in S^{\otimes}$. It follows that $xu \in S^{\otimes}$. If $n = 1$ then v is empty and, in this case, $w = xu \in S^{\otimes}$, as required. If $n > 1$, then $v = yv_1$ for some $y \in X$ and $v_1 \in \Sigma^*$, by assumption. Note that $yv_1 \in S^{\otimes}$ by inductive hypothesis. Recall also that $xu, xuy \in S^{\otimes}$. It follows from Remark 3.1.2 that $w = xuyv_1 \in S^{\otimes}$, as required. ∎

The following lemma provides a few properties of $\langle Q, X\rangle$, the first shows that its $*$-Kleene closure is contained within the subword closure. The second statement shows that each word in $\langle Q, X\rangle$ is in the subword closure, has a prefix in X, and a suffix in $Q_X^>$. The third and final statement dictates that if $Q_X^>$ is nonempty, then for all $x \in X$, $\langle Q, X\rangle_x$ is nonempty as well.

Lemma 4.2.4 *Let X, Q be nonempty subsets of S such that $X \subseteq Q$ and $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$.*

1 $\langle Q, X\rangle^* \subseteq S^{\otimes}$

2 $\langle Q, X\rangle = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Q_X^>$

3 *If $Q_X^> \neq \emptyset$ then, for all $x \in X$, we have that $\langle Q, X\rangle_x \neq \emptyset$.*

Proof. For the first statement, it is sufficient to show that $x\langle Q, X\rangle_x y \subseteq S^{\otimes}$, for all $x, y \in X$. This is shown to be a sufficient condition for "$\langle Q, X\rangle^* \subseteq S^{\otimes}$" in the second statement of Lemma 4.2.3. Let $w \in \langle Q, X\rangle_x$. Then, $\delta^{\otimes}([x], w) \in [Q_X^>]$ and, therefore, $\delta^{\otimes}([\lambda], xw) = [v]$ for some $v \in Q_X^>$. This is due to one of the properties of $Trie(S)^{\otimes}$ detailed in Remark 3.2.4. As $y \in X$, we also have $\delta^{\otimes}([v], y) \neq [sink]$. This is because $[sink] \notin X$, because $X \subseteq Q$, Q is a SCC of $Trie(S)^{\otimes}$, and it is not possible for $[sink]$ to be a part of any SCC. Hence, $\delta^{\otimes}([\lambda], xwy) \neq [sink]$ and, therefore, $xwy \in S^{\otimes}$, as required.

For the second statement, let $D = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Q_X^>$. For the direction $D \subseteq \langle Q, X\rangle$, note that every word $w \in D$ is of the form $xw_1 = w_2 y$, with $x \in X$ and $y \in Q_X^>$. Then, as $w_2 y \in S^{\otimes}$, we have $\delta^{\otimes}([\lambda], w_2 y) = [y]$. Replacing yw_2 with xw_1, we have $\delta^{\otimes}([\lambda], xw_1) \in [Q_X^>]$. Which implies $w_1 \in \langle Q, X\rangle_x$, by definition. Since $w = xw_1$ and $w_1 \in \langle Q, X\rangle_x$, we have $w \in \langle Q, X\rangle$, as required. Conversely, consider any word $w \in \langle Q, X\rangle$. This direction is proven if we show that $w \in S^{\otimes}$, $w \in X\Sigma^*$, and $w \in \Sigma^* Q_X^>$. Since $w \in \langle Q, X\rangle \subseteq \langle Q, X\rangle^*$, by the first statement of this lemma, $w \in S^{\otimes}$. Also, $w = xw_1$ for some $x \in X$ and $w_1 \in \langle Q, X\rangle_x$, by definition of $\langle Q, X\rangle$. Hence, $w \in X\Sigma^*$. Moreover, $\delta^{\otimes}([x], w_1) = [y]$ for some $y \in Q_X^>$. This implies that

xw_1 must end with the word y. Hence, $w \in \Sigma^* Q_X^{>}$.

For the third statement, take any $x \in X$. As $Q_X^{>} \neq \emptyset$, we can pick any $v \in Q_X^{>}$. As both $x, v \in Q$, there is a path from $[x]$ to $[v]$ having some label $w \in \Sigma^*$. This implies that $w \in \langle Q, X \rangle_x$. ■

The next lemma establishes that any nonempty D with $D^* \subseteq S^{\otimes}$ must be a subset of some language of the form $\langle Q, X \rangle$.

Lemma 4.2.5 *If D is a nonempty language whose words are of length at least k and $D^* \subseteq S^{\otimes}$, then*

$$D \subseteq \langle Q, X \rangle$$

for some nonempty subsets Q, X of S with $X \subseteq Q$ and $[Q]$ a nontrivial strongly connected component of $Trie(S)^{\otimes}$.

Proof. First note that by the first statement of Lemma 4.2.3, we can write $D = \bigcup_{x \in X} x(x^{-1}D)$, such that X is the nonempty set of prefixes of length k of D and $x(x^{-1}D)y \subseteq S^{\otimes}$, for all $x, y \in X$. The rest of the proof consists of two parts, that when combined establish the truth of the lemma. In the first part, we show that $X \subseteq Q$ such that $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$, and in the second part we show that $x^{-1}D \subseteq \langle Q, X \rangle_x$, for all $x \in X$.

For the first part, consider any (possibly equal) $x, y \in X$. By Lemma 4.2.3, since $x(x^{-1}D)y \subseteq S^{\otimes}$ and $y(y^{-1}D)x \subseteq S^{\otimes}$, there are words $u_1 \in x^{-1}D$ and $u_2 \in y^{-1}D$ such that $xu_1y \in S^{\otimes}$ and $yu_2x \in S^{\otimes}$. We need to show that there are nonempty paths from $[x]$ to $[y]$, and from $[y]$ to $[x]$. This, however, follows easily from the definition of $Trie(S)^{\otimes}$ and the fact that there are paths in this automaton with nonempty labels xu_1y and yu_2x.

For the second part, consider any word $w \in x^{-1}D$. Then, $xw \in D$. We need to show that $\delta^{\otimes}([x], w) \in [Q_X^{>}]$. It is sufficient to show that $\delta^{\otimes}([x], w) = [v]$ for some $v \in Q$ such that $\delta^{\otimes}([v], y) = [y]$ for all $y \in X$. Indeed, as $xw \in S^{\otimes}$, $\delta^{\otimes}([\lambda], xw) = [v]$ for some $v \in S$, so $\delta^{\otimes}([x], w) = [v]$ and, therefore, is a path from $[x]$ to $[v]$. Now,

as $xwx \in S^{\otimes}$, we have that $\delta^{\otimes}([v], x) = [x]$ and, therefore, $v \in Q$. Now take any $y \in X$. As $xwy \in S^{\otimes}$, we have that $\delta^{\otimes}([\lambda], xwy) \neq [sink]$, which implies that $\delta^{\otimes}([v], y) = [y]$, as required. ■

Lemma 4.2.6 is the final technical lemma needed to be introduced before the proof of Theorem 4.2.1 and describes the conditions under which $Q_X^{>}$ contains $P_Z^{>}$, and vise versa.

Lemma 4.2.6 *Let $X, Z, Q,$ and P be nonempty subsets of S such that $X \subseteq Q$, $Z \subseteq P$, where $[Q]$ and $[P]$ are nontrivial strongly connected components of $Trie(S)^{\otimes}$, and $Q_X^{>} \neq \emptyset$. Then*

1. *If $\langle Q, X\rangle \subseteq \langle P, Z\rangle$, then $X \subseteq Z$.*
2. *If $Z \subseteq Q$ and $\langle Q, X\rangle = \langle Q, Z\rangle$, then $X = Z$.*

Proof. For the first statement, take any $x \in X$. As $Q_X^{>} \neq \emptyset$, there is a word $w \in \langle Q, X\rangle_x$, refer to Lemma 4.2.4. So $xw \in \langle Q, X\rangle$ and, therefore, $xw \in \langle P, Z\rangle$. Then, $xw \in z\langle Q, X\rangle_x$, for some $z \in Z$, which implies that $z = x$. Hence, $x \in Z$.

The second statement follows directly by applying the finite statement twice. ■

Having proved all of the necessary lemmata, we proceed with the proof of Theorem 4.2.1.

Proof of Theorem 4.2.1. $\Longrightarrow$: Suppose that D is maximal with $D^* \subseteq S^{\otimes}$. Then, $D \subseteq \langle Q, X\rangle$ according to Lemma 4.2.5. At the same time, Lemma 4.2.4 states that $\langle Q, X\rangle^* \subseteq S^{\otimes}$. As D is maximal, we have that, in fact, $D = \langle Q, X\rangle$. Let $Y = Q_X^{>}$. By Lemma 4.2.4, we have $D = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$, as required. Note also, that as D is nonempty, we have that Y is nonempty as well, as required.

It remains to be shown that X is maximal with "$X \subseteq Q$ and $Q_X^{>} = Y$". Suppose that $X \subseteq Z \subseteq Q$ and $Q_Z^{>} = Y$. Then we need to show that $Z = X$. Because of the second statement of Lemma 4.2.6, it suffices to show that $\langle Q, X\rangle = \langle Q, Z\rangle$. This is

because if this were the case, then $D = S^{\otimes} \cap Z\Sigma^* \cap \Sigma^* Y = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$, which implies that every $z \in Z$ is also in X. In turn, $\langle Q, X\rangle = \langle Q, Z\rangle$ would follow by the maximality of D if we show that $D \subseteq \langle Q, X\rangle$. Take any $w \in D = \langle Q, X\rangle$. Then $w = xw_1$ for some $x \in X$ and $w_1 \in \langle Q, X\rangle_x$, which implies $\delta^{\otimes}([x], w_1) \in [Q_X^{>}] = [Q_Z^{>}]$. Also, as $x \in Z$ we have that $w_1 \in \langle Q, X\rangle_x$ and, therefore, $xw_1 \in \langle Q, X\rangle$, as required.

$\underline{\Longleftarrow}$: By the first statement of Lemma 4.2.4, we have that $D^* \subseteq S^{\otimes}$. To show that D is maximal, we assume $D \subseteq B$ and $B^* \subseteq S^{\otimes}$, for some language B, and we deduce that $B = D$. By Lemma 4.2.5, we have that $B \subseteq \langle P, Z\rangle$, where Z and P are nonempty subsets of S, $Z \subseteq P$, and $[P]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$. This implies that $\langle Q, X\rangle \subseteq \langle P, Z\rangle$. It suffices to show that $P = Q$ and $X = Z$. By the first statement of Lemma 4.2.6, we get $X \subseteq Z$, so there is a state belonging to both $[Q]$ and $[P]$. This implies $P = Q$ as $[P]$ and $[Q]$ are strongly connected components. Also, obviously $Q_Z^{>} = P_Z^{>}$. As X is maximal with "$X \subseteq Q$ and $Q_X^{>} = Y$" and $X \subseteq Z \subseteq Q$, to show $X = Z$, it suffices to show that $Q_X^{>} = Q_Z^{>}$.

First, by definition of $Q_X^{>}$, $X \subseteq Z$ implies $Q_Z^{>} \subseteq Q_X^{>}$. For the converse inclusion, take any $v \in Q_X^{>}$. Also, take any $x \in X$. As $x, v \in Q$, there is a path from $[x]$ to $[v]$ having some label w. This implies $w \in \langle Q, X\rangle_x$. As $\delta^{\otimes}([\lambda], xw) = [v]$, there is a word w' such that $xw = w'v$. So $w'v \in \langle Q, X\rangle$ and, therefore, $w'v \in \langle Q, Z\rangle$. Then, by the second statement of Lemma 4.2.4, we have $w'v \in \Sigma^* Q_Z^{>}$. Thus, $v \in Q_Z^{>}$ as required. ■

The following two lemmata are necessary for the proof of Theorem 4.2.2, which is similar to Theorem 4.2.1 except that it involves a fork state in the strongly connected component $[Q]$. Recall from earlier that a state in an automaton is called a fork state if there are at least two transitions going out of that state.

Lemma 4.2.7 *If $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$ then, for every $v \in Q$ and $n \geq 1$, there is $u \in Q$ and a path of length n from $[u]$ to $[v]$.*

Proof. As $[Q]$ is nontrivial, there exists a path of some length $\ell \geq 1$ from $[v]$ to $[v]$. Obviously this path can be iterated arbitrarily many times to obtain a long path of length at least n from $[v]$ to $[v]$. Thus, there has to be some state $[u]$ in that sufficiently long path as required. ∎

The following lemma describes properties that result from requiring that $[Q]$ contains a fork state. Since Theorem 4.2.2 involves a fork state in the SCC $[Q]$, it will be useful in the proof of the theorem.

Lemma 4.2.8 *Let $[Q]$ be a nontrivial strongly connected component of $Trie(S)^{\otimes}$ with $Q \subseteq S$*

1 *If $[v]$ is a $[Q]$-fork state having transitions to some distinct states $[x_1], [x_2] \in [Q]$, then $\langle Q, X\rangle$ is a nontrivial language, where $X = \{x_1, x_2\}$.*

2 *There is a subset X of Q such that $\langle Q, X\rangle$ is nontrivial if and only if a $[Q]$-fork state exists.*

Proof. For the first statement, the premise implies that there are distinct symbols $\sigma_1, \sigma_2 \in \Sigma$ such that $\delta^{\otimes}([v], \sigma_1) = [x_1]$ and $\delta^{\otimes}([v], \sigma_2) = [x_2]$. Also, by Lemma 4.2.7, there is a state $[u]$ in $[Q]$ and a word x of length $k-1$ such that $\delta^{\otimes}([u], x) = [v]$. This implies that $x_1 = x\sigma_1$ and $x_2 = x\sigma_2$, and, therefore, $u \in Q_X^{>}$. Then, the sets $\langle Q, X\rangle_{x_1}$ and $\langle Q, X\rangle_{x_2}$ are nonempty, which implies that there are words w_1, w_2 such that $x_1 w_1, x_2, w_2 \in \langle Q, X\rangle$. Finally, as $x_q w_1 x_2 w_2 \neq x_2 w_2 x_1 w_1$, the language $\langle Q, X\rangle$ is nontrivial.

For the second statement, we consider both directions separately.

$\underline{\Longleftarrow:}$ Consider a $[Q]$-fork state $[v]$. Then, $[v]$ has two outgoing transitions to some states $[x_1], [x_2] \in [Q]$. By the definition of $Trie(S)^{\otimes}$, these two states are different. By the first statement of this lemma, we have that $\langle Q, X\rangle$ is a nontrivial language,

where $X = \{x_1, x_2\}$.

$\underline{\Longrightarrow}$: Note that as $\langle Q, X\rangle$ is nontrivial, the set $Q_X^>$ is nonempty. So let $v \in Q_X^>$. If X has at least two distinct elements x_1, x_2 then, as there are paths from $[v]$ to $[x_1]$ and $[x_2]$, it follows that a $[Q]$-fork state must exist in these paths, as required. Now suppose $X = \{x\}$, but assume to the contrary that no fork-state exists. Then the component $[Q]$ is a single cycle and, therefore, $Q_X^>$ consists of exactly one element, which is v. Now there is one shortest path from $[x]$ to $[v]$ having some label w and exactly one shortest path from $[v]$ to $[x]$ having some label u. Then it is easy to see that all paths from $[x]$ to $[v]$ have labels in $w(uw)^*$ which implies that $\langle Q, X\rangle = xw(uw)^*$, contradicting the premise that $\langle Q, X\rangle$ is nontrivial. ■

With all of the necessary lemmata proven, we can now proceed with the proof of Theorem 4.2.2.

Proof of Theorem 4.2.2. $\underline{\Longrightarrow}$: Apply Theorem 4.2.1: there are nonempty subsets X, Y, Q of S such that $D = \langle Q, X\rangle = S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$ and $X, Y \subseteq Q$, $[Q]$ is a nontrivial strongly connected component of $Trie(S)^{\otimes}$, $Y = Q_X^>$ and X is maximal with "$X \subseteq Q$ and $Q_X^> = Y$." It remains to show that $[Q]$ contains a fork-state, but this follows directly from the second statement of Lemma 4.2.8.
$\underline{\Longleftarrow}$: This statement is a weaker statement than the one in Theorem 4.2.1. The proof is identical. ■

4.3 Algorithms that obtain maximal D's with $D^* \subseteq S^{\otimes}$

We now consider algorithms that solve the following two problems. Each algorithm from this section is taken from [18].

(P1) Given a subword constraint S, compute a nonempty language D that is maximal with $D^* \subseteq S^{\otimes}$.

(P2) Given a subword constraint S, compute a nontrivial language D that is maximal with $D^* \subseteq S^{\otimes}$.

We solve these problems by considering the following subproblems.

(SP1) Given a nonempty subset X of Q, compute the set $Q_X^>$.

(SP2) Given nonempty subsets Z, Y of Q such that $Q_Z^> = Y$, compute X such that $Z \subseteq X$ and X is maximal with "$X \subseteq Q$ and $Q_X^> = Y$".

(SP3) Given nonempty subsets X, Y of Q compute a deterministic automaton accepting $S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$.

For the remainder of this section, we abbreviate $Trie(S)^{\otimes}$ as T. We also establish a bijective encoding from the words in Q onto the set $\bar{Q} = \{0, 1, \ldots, |Q| - 1\}$, such that for $v \in Q$, $\bar{v}$ is the code of $v \in \bar{Q}$. Hashing texhniques can be used in the encoding and decoding functions, and can therefore be done in time $O(1)$. Now we describe the algorithm ASP1(T, X, Q) that solves (SP1).

ASP1(T, Q, X):
Take any pair $v \in Q$ and $x \in X$, and test whether $\delta^{\otimes}([v], x) \neq [sink]$. If, for the current v, the test is true for all $x \in X$, then v is added in $Q_X^>$. It can be seen that this algorithm can be completed in $O(|Q||X|k)$ time.

As it turns out, it is necessary for (SP1) solved multiple times to solve either (P1) or (P2), as we require that our solution be maximal. We require a a preprocessing step to compute a $|Q| \times |Q|$ Boolean array BQ such that $BQ[\bar{v}, \bar{v}']$ is true if and only if $\delta^{\otimes}([v], v') \neq [sink]$. This array can be computed in time $O(|Q|^2)$, as it involves $|Q|^2$ steps and, in each step, we run the DFA T on an input word of length k. Thus, the algorithm works as described earlier, except that now the test $\delta^{\otimes}([v], x) \neq [sink]$ is reduced to whether $BQ[\bar{v}, \bar{x}]$ is true. Therefore, assuming that BQ is available, the algorithm runs in time $O(|Q||X|)$.

To solve (SP2), we assume that $Q_Z^> = Y$, and we compute X by initializing it to Z, and then by repeatedly adding into X a new element from $V = Q \setminus X$, provided that the condition $Q_X^> = Y$ remains true. The algorithm that solves (SP2) is given below.

ASP2(T, Q, Z, Y):

1. $X = Z$; $V = Q \setminus X$
2. **while** $(V \neq \emptyset)$
3. **do**
4. Pick $v \in V$;
5. Use **ASP1**$(T, Q, X \cup \{v\})$ to compute $Y' = Q^{>}_{X \cup \{v\}}$;
6. **if** $(Y' = Y) X = X \cup \{v\}$;
7. $V = V \setminus \{v\}$
8. return X;

Remark 4.3.1 Given a subset Z of Q, the above algorithm computes in time $O(|Q|^2(|Q| - |Z|))$ a subset X of Q such that $Z \subseteq X$ and X is maximal with "$X \subseteq Q$ and $Q^{>}_X = Y$".

We now describe the algorithm that solves (SP3), **ASP3**(T, Q, X, Y).

ASP3(T, Q, X, Y):

Assuming that T, Q, X, Y are all given. The required deterministic automaton T' that recognizes $S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$ can be constructed by changing T in linear time in terms of the sizes of the given structures.

- The states of T' are $[sink]$, all states in $[Q]$ and all $[z]$ with a z as a prefix of X.
- The initial state is $[\lambda]$, and the set of terminal states in $[Y]$.
- The transition of T' are all of the transitions of T involving only the above states.

It is easy to see that the automaton T' accepts precisely those words in $S^{\otimes}$, that have a prefix in X and a suffix in $Q_X^{>}$. We are now ready to consider (P1) once more, consider below A1, the algorithm that solves (P1). Note that A1 runs in polynomial time.

A1(S):

1. Compute $T = Trie(S^{\otimes})$;
2. Compute the strongly connected components of T;
3. Pick a nontrivial component $[Q]$, exit if none exist.
4. Compute the Boolean array BQ.
5. Compute two nonempty subsets Z, Y of Q such that $Q_Z^{>} = Y$;
6. Use **ASP2**(T, Q, Z, Y) to compute a maximal X with $Q_X^{>} = Y$;
7. Use **ASP3**(T, Q, X, Y) to compute and return the automaton for $S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$.

The algorithm that solves (P2) is similar to the one used for (P1). Note that the only differences are found in steps 3 and 5 where we require a fork state to be found. Similar to A1, A2 also can be completed in polynomial time.

A2(S):

1. Compute $T = Trie(S^{\otimes})$;
2. Compute the strongly connected components of T;
3. Pick a SCC $[Q]$ containing a $[Q]$-fork state, exit if none exist;
4. Compute the Boolean array BQ.
5. Compute two nonempty subsets Z, Y of Q such that $|Z| \geq 2$ $Q_Z^{>} = Y$;
6. Use **ASP2**(T, Q, Z, Y) to compute a maximal X with $Q_X^{>} = Y$;

7. Use **ASP3**(T, Q, X, Y) to compute and return the automaton for $S^{\otimes} \cap X\Sigma^* \cap \Sigma^* Y$.

Chapter 5

DNA Code Words

In this chapter, we discuss and define the concept of a code, then we define and give examples of different types of codes. We then introduce the concept of involution mappings and combine this with codes to create involution codes. We conclude by defining different types of involution codes and consider how each code is useful in the field of DNA computing.

5.1 Codes

We introduce the concept of a *code*, which is a language such that every word in the +-Kleene Closure of the language has a unique factorization. Put differently, a language X is a code if every word in X^+ can be written as a unique factorization of words from X. We give a formalization of this concept below. Each of the following definitions are taken from [2].

Definition 5.1.1 Let $X \subseteq \Sigma^+$ be a language. Then X is a code if, for every word $w \in X^+$, there is exactly one factorization of words in X whose concatenation is equal to w.

To illustrate the difference between languages that are codes and languages that aren't, we consider the following example.

Example 5.1.2 Let $\Sigma = \{a, b\}$.

Let $X = \{aa, ab, bb\}$.

Also, let $Y = \{ab, b, bb\}$.

The language X is a code, as every word in X^+ can be written as a unique factorization of words in X. For instance, $aabbabaa \in X^+$ and $aabbabaa = (aa)(bb)(ab)(aa)$. However, the language Y is not a code as $abbb \in Y^+$, but $abbb = (ab)(bb) = (ab)(b)(b)$. This word has two distinct factorizations over Y^+, disqualifying Y from being a code. $\Diamond$

Observe that if a language contains the empty word, λ, then that language cannot be a code. This is because for any word $w \in L$, $w = w\lambda = \lambda w$. Moreover, codes are not closed under complement. This is because for any code X, $\lambda \notin X$. Since $\lambda \notin X$ we have that $\lambda \in X^c$. Which makes X^c not a code.

Definition 5.1.3 Let $X \subseteq \Sigma^+$ be a code. Then we call X:

A *prefix code* if and only if $X \cap X\Sigma^+ = \emptyset$;

A *suffix code* if and only if $X \cap \Sigma^+X = \emptyset$;

An *infix code* if and only if $X \cap (\Sigma^*X\Sigma^+ \cup \Sigma^+X\Sigma^*) = \emptyset$; and

A *comma-free code* if and only if $X^2 \cap \Sigma^+X\Sigma^+ = \emptyset$.

Next, we present examples of codes that *do not* belong to each type of code mentioned in Definition 5.1.3, as well as justifications for why.

Example 5.1.4 Let $X_1 = \{a, b, aa, ab\}$, $X_2 = \{a, b, aa, ba\}$, $X_3 = \{a, b, ab\}$, $X_4 = \{aa, bb\}$. Then:

X_1 is not a prefix code, as $a \in X$ is a proper prefix of both $aa, ab \in X$;

X_2 is not a suffix code, as $a \in X$ is a proper suffix of both $aa, ba \in X$;

X_3 is not an infix code, as $a \in X$ is a proper prefix of both $ab \in X$ and $b \in X$ is a proper suffix of $ab \in X$; and

X_4 is not a comma-free code, as $aa, bb \in X$ are proper subwords of $aaaa, bbbb \in X^2$. $\Diamond$

Next, we present examples of codes that *do* belong to each type of code mentioned in Definition 5.1.3, as well as justifications for why.

Example 5.1.5 Let $Y_1 = \{a, ba, bb\}$, $Y_2 = \{a, ab, bb\}$, $Y_3 = \{aa, ab, bb\}$, $Y_4 = \{bba, abaa\}$. Then:

Y_1 is a prefix code, as no code word in Y_1 is a proper prefix of the other code words;

Y_2 is a suffix code, as no code word in Y_2 is a proper suffix of the other code words;

Y_3 is an infix code, as no code word in Y_3 is a proper prefix or a proper suffix of the other code words; and

Y_4 is a comma-free code, as no code word in Y_4 is a proper subword of the other words in $Y_4^2 = \{bbabba, bbaabaa, abaabba, abaaabaa\}$. For every $w \in Y_4$, $pwp \notin Y_4^2$ for all $p, q \in \Sigma^+$. $\Diamond$

5.2 Involution Mappings

Because of how single-stranded DNA molecules attach together (or *hybridize*) by forming bonds between complementary strands, any map of DNA hybridization must account for these complementary bonds. We introduce the concept of an *involution* which is a mapping that is equivalent to the identity mapping when composed with itself. A formal definition of an involution is given below. We also define two different subtypes of involution mappings.

Definition 5.2.1 A mapping $\theta : \Sigma \to \Sigma$ is called an *involution mapping* if

$$\theta^2(a) = \theta(\theta(a)) = a, \forall a \in \Sigma.$$

An involution mapping is called *morphic* if $\theta : \Sigma^* \to \Sigma^*$ such that $\theta(uv) = \theta(u)\theta(v)$ for every $u, v \in \Sigma^*$.

An involution mapping is called *antimorphic* if $\theta : \Sigma^* \to \Sigma^*$ such that $\theta(uv) = \theta(v)\theta(u)$ for every $u, v \in \Sigma^*$.

Example 5.2.2 Let θ be a morphic involution mapping with $\theta(a) = b$ and

$\theta(b) = a$. Then,

$$\theta(bbab) = \theta(b)\theta(b)\theta(a)\theta(b) = aaba$$

Now let θ be an antimorphic involution mapping with $\theta(a) = b$ and $\theta(b) = a$. Then,

$$\theta(bbab) = \theta(b)\theta(a)\theta(b)\theta(b) = abaa \qquad \Diamond$$

Just as Σ^+ and Σ^* can be extended to languages, we can extend involution mappings to languages as well by mapping them word by word.

Definition 5.2.3 Let $w \in L$ be a word in some language over an alphabet Σ, and let θ be an involution mapping. Then,

$$\theta(L) = \bigcup_{w \in L} \theta(w).$$

We give an example of the involution mapping of a language L below.

Example 5.2.4 Say $L = \{abb, bbbab\}$, and $\Sigma = \{a, b\}$. If θ is a morphic involution mapping with with $\theta(a) = b$ and $\theta(b)$, then $\theta(L) = \{\theta(abb)\} \cup \{\theta(bbbab)\}$
$= \{\theta(a)\theta(b)\theta(b)\} \cup \{\theta(b)\theta(b)\theta(b)\theta(a)\theta(b)\} = \{baa, aaaba\}$.

Whereas, if θ is an antimorphic involution mapping, then $\theta(L) = \{\theta(abb)\} \cup \{\theta(bbbab)\} = \{\theta(b)\theta(b)\theta(a)\} \cup \{\theta(b)\theta(a)\theta(b)\theta(b)\theta(b)\} = \{aab, abaaa\}$. $\Diamond$

As mentioned earlier, the map of DNA hybridization, ν, is an involution. It is defined by $\nu(A) = T$, $\nu(T) = A$, $\nu(G) = C$, and $\nu(G) = C$ over the DNA alphabet $\Delta = \{A, T, G, C\}$. We can verify that Δ is an involution mapping. Note that $\nu(\nu(A)) = \nu(T) = A$, $\nu(\nu(T)) = \nu(A) = T$, $\nu(\nu(G)) = \nu(C) = G$, and $\nu(\nu(C)) = \nu(G) = C$.

Watson-Crick complementarity dictates that different bonded DNA strands have a reverse orientation; that is to say, the $5'$ end of one strand attaches to the $3'$ end of the other. We account for this reversing by defining another involution $\rho : \Delta^* \longrightarrow \Delta^*$ inductively. For every $x \in \Delta$, $\rho(x) = x$, and for every $u \in \Delta^*$, $\rho(xu) = \rho(u)\rho(x)$. In general, for $u, v \in \Delta^*$, $\rho(uv) = \rho(v)\rho(u)$.

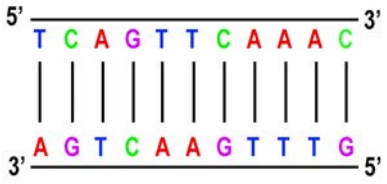

Figure 5.1: DNA molecule from Example 5.2.5.

Watson-Crick complementary is represented by the composition of the involutions ν and ρ, with $\nu \circ \rho = \rho \circ \nu$. For a given DNA strand, we define the Watson-Crick complement as the antimorphic involution $\hat{\theta} : \Delta^* \longrightarrow \Delta^*$ such that $\hat{\theta} = \rho \circ \nu = \nu \circ \rho$. The next example applies $\hat{\theta}$ to a DNA strand, and demonstrates the fact that $\hat{\theta}$ is an involution. Figure 5.1 provides a visual representation of what is happening on a molecular level.

Example 5.2.5 The single strand $5' - TCAGTTCAAAC - 3'$ would map to $5' - GTTTGAACTGA - 3'$.

$$\begin{aligned}\hat{\theta}(TCAGTTCAAAC) &= \hat{\theta}(C)\hat{\theta}(A)\hat{\theta}(A)\hat{\theta}(A)\hat{\theta}(C)\hat{\theta}(T)\hat{\theta}(T)\hat{\theta}(G)\hat{\theta}(A)\hat{\theta}(C)\hat{\theta}(T) \\ &= GTTTGAACTGA\end{aligned}$$

The fact that $\hat{\theta}$ is an antimorphic involution is demonstrated below.

$$\begin{aligned}\hat{\theta}(\hat{\theta}(TCAGTTCAAAC)) &= \hat{\theta}(\hat{\theta}(C)\hat{\theta}(A)\hat{\theta}(A)\hat{\theta}(A)\hat{\theta}(C)\hat{\theta}(T)\hat{\theta}(T)\hat{\theta}(G)\hat{\theta}(A)\hat{\theta}(C)\hat{\theta}(T)) \\ &= \hat{\theta}(GTTTGAACTGA) \\ &= \hat{\theta}(A)\hat{\theta}(G)\hat{\theta}(T)\hat{\theta}(C)\hat{\theta}(A)\hat{\theta}(A)\hat{\theta}(G)\hat{\theta}(T)\hat{\theta}(T)\hat{\theta}(T)\hat{\theta}(G) \\ &= TCAGTTCAAAC\end{aligned}$$

$\Diamond$

5.3 Involution Codes

In this section, we combine involutions and codes to define and study codes that are generalizations of the involution map of DNA. Each of these definitions are taken from [12]. In each example, θ is either a morphic or antimorphic involution with $\theta : \{a, b\} \longrightarrow \{a, b\}$, where $\theta(a) = b$ and $\theta(b) = a$. Each involution code is designed to prevent a certain kind of unwanted hybridization. After defining each involution code, we give an example of that code and describe the unwanted hybridization that that code prevents.

Definition 5.3.1 The code $X \subseteq \Sigma^+$ is θ-infix if and only if

$$\Sigma^*\theta(X)\Sigma^+ \cap X = \emptyset \text{ and } \Sigma^+\theta(X)\Sigma^* \cap X = \emptyset.$$

Alternatively, a code is θ-infix if no word in $\theta(X)$ is a proper subword of any word in X.

Example 5.3.2 Let θ be a morphic involution.
The code $X = \{bb, bab, abbb, bbabbab\}$ is θ-infix as $\theta(X) = \{aa, aba, baaa, aabaaba\}$ and no code word in $\theta(X)$ is a proper subword of any code word in X. The code $Y = \{aaab, bbbaa\}$ is *not* a θ-infix code because $\theta(Y) = \{bbba, aaabb\}$ and
$bbba \in \theta(Y)$ is a proper subword of $bbbaa \in Y$. $\Diamond$

The unwanted hybridization that θ-infix codes prevent is the one where one strand is a complementary subword of another. A visual representation of the unwanted hybridization on the molecular level is given in Figure 5.2.

Definition 5.3.3 We call $X \subseteq \Sigma^+$ a *θ-subworb-k* code if and only if for all $u \in \Sigma^k$,

$$\Sigma^* u \Sigma^i \theta(u) \Sigma^* \cap X = \emptyset$$

for $i \geq 1$.

Put differently, a *θ-subworb-k* code does not contain complemetary k-length subwords seperated by one or more letters.

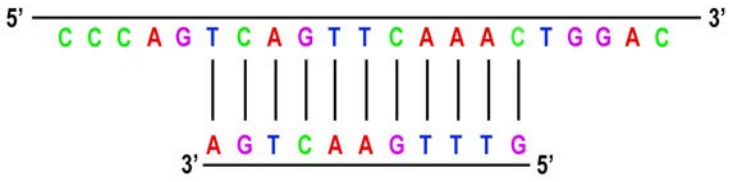

Figure 5.2: Type of hybridization that is prevented by θ-infix codes.

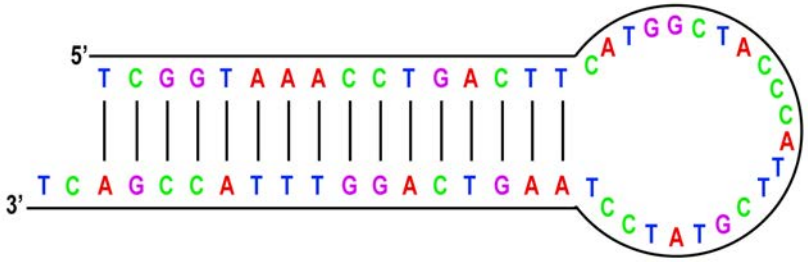

Figure 5.3: Type of hybridization that is prevented by θ-subword-k codes.

Example 5.3.4 Let θ be an antimorphic involution. The code $X = \{abbababa, bbbabbab\}$ is θ-subword-k for $k > 3$, whereas the code $Y = \{baababbab, aabaaba\}$ is not because $baa \in \Sigma^3$, $\theta(baa) = \theta(a)\theta(a)\theta(b) = bba$, and $(baa)ba(bba)b \in Y$. Note however, that Y is θ-subword-k for $k > 4$. $\Diamond$

The unwanted hybridization that θ-subword-k codes prevent is the one where one strand wraps onto itself and forms a hairpin. A visual representation of the unwanted hybridization on the molecular level is given in Figure 5.3.

Definition 5.3.5 The code $X \subseteq \Sigma^+$ is a θ-k code if and only if

$$Sub_k(X) \cap Sub_k(\theta(X)) = \emptyset$$

for $k > 0$.

In other words, a θ-k code has no k-length subwords that are Watson-Crick complements of k-length subwords of other code words.

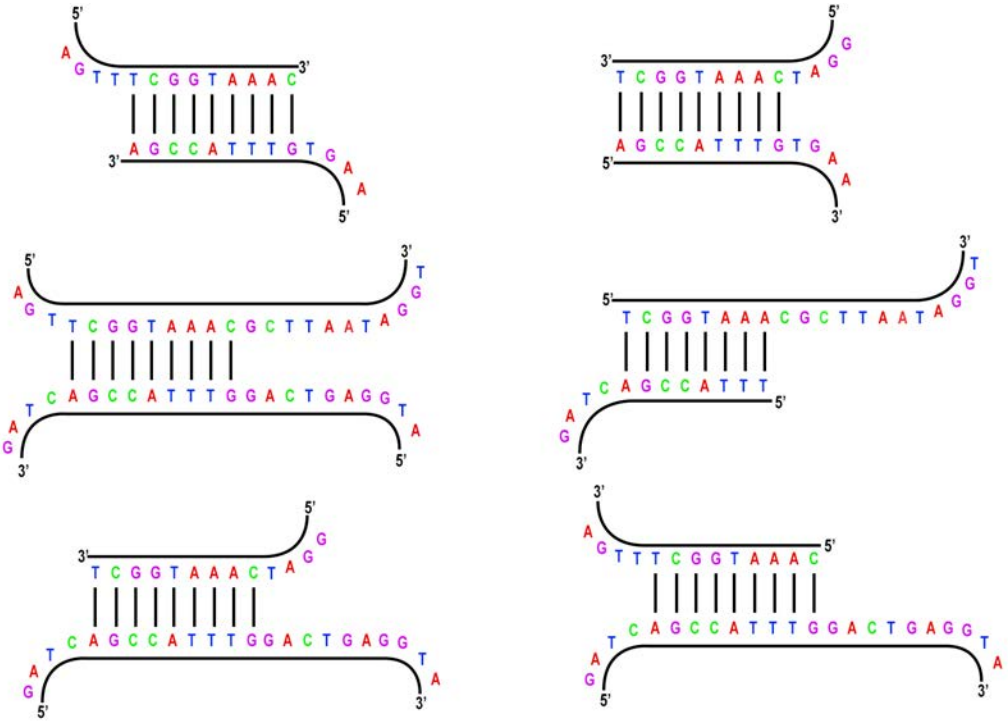

Figure 5.4: Type of hybridizations that are prevented by θ-k codes.

Notice how closely related θ-k codes seem to be to the subword closure. Indeed, the subword closure was originally introduced by Kari et al. in [14], as a characterization for *maximal bond-free languages*, which themselves were introduced earlier by Jonoska and Mahalingam in [11] as θ-k codes.

Example 5.3.6 Let θ be a morphic involution. If $X = \{abba, babba\}$, it follows that $\theta(X) = \{baab, abaab\}$. The code X is not θ-2, as $Sub_2(X) = \{ab, ba, bb\}$ and $Sub_2(\theta(X)) = \{aa, ab, ba\}$. So $Sub_2(X) \cap Sub_2(\theta(X)) = \{ab, ba\} \neq \emptyset$. Note that X is θ -3 however. As $Sub_3(X) = \{abb, bab, bba\}$ and $Sub_3(\theta(X)) = \{aab, aba, baa\}$, meaning $Sub_3(X) \cap Sub_3(\theta(X)) = \emptyset$. $\Diamond$

The unwanted hybridizations that θ-k codes prevent are many, and are depicted in Figure 5.4. Notice in Example 5.3.6, X is a θ-4 code. When $k = 4$, $Sub_4(X) = \{abba, babb\}$ and $Sub_4(\theta(X)) = \{baab, abaa\}$. The intersection of these two sets is empty, i.e. $Sub_4(X) \cap Sub_4(\theta(X)) = \emptyset$. Similarly, for $k = 5$, $Sub_5(X) = \{babba\}$ $Sub_5(\theta(X)) = \{abaab\}$. The intersection of these two sets are also empty, and

therefore X is θ-5. This observation can be generalized for $k > 3$, as per Proposition 5.3.7.

Proposition 5.3.7 *Let $X \subseteq \Sigma^+$ be a θ-k code. Then X is a θ-k' code for all $k' > k$.*

Proof. Assume that X is a θ-k code. In other words, $Sub_k(X) \cap Sub_k(\theta(X)) = \emptyset$. Let $k < k'$. Suppose that X is not a θ-k'. This implies that $Sub_{k'}(X) \cap Sub_{k'}(\theta(X)) \neq \emptyset$. It follows that there exists a $w \in \Sigma^{k'}$, $x \in X$,$y \in \theta(X)$, and $u_1, u_2, v_1, v_2 \in \Sigma^*$ such that $x = u_1 w v_1$ and $y = u_1 w v_2$. The word w can be factored into individual symbols from Σ, i.e. $w_i \in \Sigma$, $x = u_1 w_1 w_2 ... w_k w_{k+1} ... w_{k'} v_1$ and $y = u_2 w_1 w_2 ... w_k w_{k+1} ... w_{k'} v_2$. Notice that $w_{k+1} ... w_{k'} v_1, w_{k+1} ... w_{k'} v_2 \in \Sigma^*$ where $w_i \in \Sigma$. This implies that $Sub_k(X) \cap Sub_k(\theta(X)) \neq \emptyset$, which is a contradiction with the assumption that X is a θ-k code. Therefore, X is a θ-k' code for all $k' > k$. ∎

These θ-k codes are extremely desirable. This is because the definition of θ-k codes implies that both X and $\theta(X)$ are θ-infix and θ-subword-k as well. For this reason, θ-k codes also prevent all unwanted hybridizations that we have discussed thus far.

Proposition 5.3.8 *Let $X \subseteq \Sigma^+$, if X is a θ-k code, then X and $\theta(X)$ are both θ-infix and θ-subword-k.*

Proof. <u>θ-infix:</u> Suppose that x is a θ-k code but it is not a θ-infix code. Then there exists $x, y \in X$ and $u, v \in \Sigma^*$ (where u and v cannot both be the empty word λ), such that $x = u\theta(y)v$. It follows that $\theta(y) \in Sub(x) \subseteq Sub(X)$. This means that there exists a $k < |x|$ such that $Sub_k(X) \cap Sub_k(\theta(X)) \neq \emptyset$. This contradicts X being a θ-k code. Therefore, X is a θ-infix code.

Similarly, if X is a θ-k, then $\theta(X)$ is a θ-infix code.

<u>θ-subword-k:</u> Suppose that X is a θ-k code but is not a θ-subword-k code. Let θ be a morphic involution. This means that there exists $u \in \Sigma^k$, such that

$\Sigma^* u \Sigma^i \theta(u) \Sigma^* \cap X \neq \emptyset$ for some $i > 1$. In other words, there exists $a, b \in \Sigma^*$, $c \in \Sigma^i$ and $x \in X$ such that $auc\theta(u)b = x$. This equality implies $\theta(u) \in Sub_{|\theta(u)|}(\theta(x))$. Apply θ to x, since θ is morphic we obtain $\theta(a)\theta(u)\theta(c)u\theta(b) = \theta(x)$. This implies that $\theta(u) \in Sub_{|\theta(u)|}(\theta(x))$. Since $\theta(u) \in Sub_{|\theta(u)|}(x) \cap Sub_{|\theta(u)|}(\theta(x))$, this contradicts that X is a θ-k code. Therefore, X is a θ-subword-k code.

Similarly, if X is a θ-k code, then $\theta(X)$ is a θ-subword-k code. ■

Chapter 6

Strictly Locally Testable Languages and FE Systems

In this chapter we define and discuss the two subclasses of regular languages known as strictly locally testable languages and locally testable languages. We then demonstrate that the subword closure is SLT. Next we define and discuss the concept of forbidders and enforcers, forbidding and enforcing sets, and the languages that forbidding and enforcing sets create (f-languages and e-languages, respectively). We then discuss the languages that result from combining forbidding and enforcing languages (fe-languages) and provide examples of each type of language that we discuss. We then make a connection between fe-languages and locally testable languages. We conclude by demonstrating that the subword closure is an fe-language.

6.1 SLT and LT Languages

Originally studied by Mcnaughton and Papert in [21], *locally testable languages* (or LT languages) are a family of regular languages whose membership is determined by the triple $(Pref_k(w), Suff_k(w), Int_k(w))$. For $k \geq 0$ and $w \in \Sigma^*$ such that $|w| \geq k$ (or $w \in \Sigma^{\geq k}$), let $Int_k(w)$ (interior subwords) denote the set of subwords of w of length k that occur at other than prefix and suffix positions. We can extend Int_k to languages. Let L be a language, then $Int_k(L) = \bigcup_{w \in L} Int_k(w)$. We give a formalized definition of locally testable languages, as well as k-testable languages in the definition below.

Definition 6.1.1 A language $L \subseteq \Sigma$ is called *k-testable* if and only if for any words $x, y \in \Sigma^*$, the conditions $Pref_k(x) = Pref_k(y)$, $Suff_k(x) = Suff_k(y)$, and $Int_k(x) = Int_k(y)$ imply that $x \in L$ if and only if $y \in L$. A language is called *locally testable* if it is k-testable for some integer $k \geq 1$.

A language is *strictly locally testable* (or SLT) when we can specify sets P, S, and I of "allowed" prefixes, suffixes and internal subwords such that a word is in the language if and only if its subwords are within these sets. A formalized definition of strictly locally testable languages, as well as *strictly k-testable languages* is given below.

Definition 6.1.2 A language $L \subseteq \Sigma^*$ is called *strictly k-testable* if there exist finite sets $P, S, I \subseteq \Sigma^*$ such that for all $w \in \Sigma^{\geq k}$, we have $w \in L$ if and only if $Pref_k(w) \subseteq P$, $Suff_k(w) \subseteq S$, and $Int_k(w) \subseteq I$. A language is called *strictly locally testable* if it is strictly k-testable for some integer $k \geq 1$.

By the definition of SLT languages, it can be seen that LT languages are the boolean closure of SLT languages. In fact, $SLT \subseteq LT$. As such, there are languages that are LT, but not SLT. We give an example of such a language below.

Example 6.1.3 Let $\Sigma = \{a, b\}$. The language $L = aa\Sigma^*aa \cup bb\Sigma^*bb$ is 2-testable and hence, locally testable. A word w belongs to L if and only if $Int_2(w) \neq \emptyset$ and $Pref_2(w) = Suff_2(w) = \{aa\}$ or $Pref_2(w) = Suff_2(w) = \{bb\}$. The language L, however, is not SLT. As both $aauaa$ and $bbvbb$ belong to L for any $u, v \in \Sigma^*$, any triple (P, S, I) defining L must allow both aa and bb as a prefix, as well as a suffix. Thus, $P = S = \{aa, bb\}$. Also, $I = \Sigma^2$ since any $u, v \in \Sigma^*$ are allowed. Thus, $aazbb$ will be allowed for any $z \in \Sigma^*$, contradicting the definition of L. This shows that L is not strictly 2-testable and, therefore, not strictly k-testable for all $k \geq 2$. $\Diamond$

6.2 Subword Closure as an SLT Language

Recall from Section 3.2 the subword closure of a nonempty, finite set of words (subword constraint) S. Also recall from Theorem 2.3.6 that a language is regular

if it is accepted by a DFA. In [14], $S^{\otimes}$ is shown to be regular by the DFA $Trie(S)^{\otimes}$. In this section we determine that the subword closure belongs to the strictly locally testable subclass of regular languages.

Proposition 6.2.1 *Let X be a subword constraint, a nonempty, finite set of words of length k. The subword closure of X, $X^{\otimes}$, is strictly locally testable with $P = X$, $S = X$, and $I = X$.*

Proof. Suppose $w \in X^{\otimes}$, $|w| \geq k$.

By definition of subword closure, we have that $Pref_k(w) \subseteq X$, $Suff_k(w) \subseteq X$, and $Int_k(w) \subseteq X$.

Conversely, suppose $w \in \Sigma^*$, with $|w| \geq k$ and with $Pref_k(w) \subseteq X$, $Suff_k(w) \subseteq X$, and $Int_k(w) \subseteq X$. Since $Pref_k(w) \cup Suff_k(w) \cup Int_k(w) = Sub_k(w)$, we have that $Sub_k(w) \subseteq X$. Therefore, $w \in X^{\otimes}$.

Thus, $X^{\otimes}$ is strictly locally testable with $P = X$, $S = X$, and $I = X$. ∎

By Proposition 6.2.1 we can see that the subword closure is indeed in the family of strictly locally testable languages. Note that Proposition 6.2.1 does not imply that the maximal languages D we characterized in Chapter 4 are SLT. We conjecture that this is the case but proving it requires further research.

6.3 Fe-systems

When they were originally defined in [5], [6], and [7] a forbidding-enforcing system (fe-system) was used to define a *family* of languages. We instead consider the fe-systems model introduced in [8] in which an fe-system defines a single language.

A *forbidding set* $\mathcal{F}$ is a family of finite nonempty subsets of Σ^+, each element of which is called a *forbidder*. We say a word w is *consistent* with a forbidder F, if $F \not\subseteq Sub(w)$. A word w is *consistent with a forbidding set* $\mathcal{F}$ if w is consistent with all $F \in \mathcal{F}$. We denote the set of all words consistent with $\mathcal{F}$, $L(\mathcal{F})$. Such a language is called a *forbidding language* or *f-language*. Note that $L(\mathcal{F}) = \Sigma^*$ if and only if

$\mathcal{F} = \varnothing$ and that $\lambda \in L(\mathcal{F})$ for every $\mathcal{F}$. An example of a forbidding set can be found below.

Example 6.3.1 Let $\Sigma = \{a, b\}$ and $\mathcal{F} = \{\{a^m b^n\}, \{b^m a^n\} \mid m = n\}$.Then, $L(\mathcal{F})$ contains words such that any consecutive string of a's and b's must be of different lengths. $\Diamond$

We call any pair (x, Y) an *enforcer* if $x \in \Sigma^*$ and Y is a finite nonempty subset of Σ^+ such that x is a proper subword of each $y \in Y$. If $x \in \Sigma^+$, a word w *satisfies an enforcer* (x, Y) if every occurrence of x in w is embedded in some word from Y. In other words, if $w = uxv$ for some $u, v \in \Sigma^*$ then there exists $y \in Y$ and $u_1, u_2, v_1, v_2 \in \Sigma^*$ such that $u = u_1u_2, v = v_2v_1$, and $y = u_2xv_2$.

If $x \notin Sub(w)$ then w satisfies the enforcer (x, Y) trivially. When $x = \lambda$, the enforcer (λ, Y) is called *brute*. In this case, a word w satisfies the enforcer if there exists a word $y \in Y$ such that y is a subword of w.

We call a set of enforcers $\mathcal{E}$ an *enforcing set*. We say w *satisfies* $\mathcal{E}$ if and only if w satisfies every enforcer in $\mathcal{E}$. The language of all words that satisfy $\mathcal{E}$ is denoted by $L(\mathcal{E})$ and is called an *enforcing language* or *e-language*. Note that $L(\mathcal{E}) = \Sigma^*$ if and only if $\mathcal{E} = \varnothing$.

Example 6.3.2 Let $\Sigma = \{a\}$ and $L = \{a^{2n} \mid n \geq 0\}$. Then, the enforcing set $\mathcal{E} = \{(\lambda, \{a, aa\} \cup \{(a^{2^i+1}, \{a^{2^{i+1}}\}) \mid i \geq 1\}$ defines L, i.e. $L = L(\mathcal{E})$. $\Diamond$

We combine both forbidding sets and enforcing sets into a single system. A *forbidding-enforcing system* is an ordered pair $(\mathcal{F}, \mathcal{E})$ such that $\mathcal{F}$ is a forbidding set and $\mathcal{E}$ is an enforcing set. The language $L(\mathcal{F}, \mathcal{E})$ consists of all words that are both consistent with $\mathcal{F}$ and satisfy $\mathcal{E}$, i.e. $L(\mathcal{F}, \mathcal{E}) = L(\mathcal{F}) \cap L(\mathcal{E})$. Such a language L is called a *forbidding-enforcing language* or an fe*-language*.

Example 6.3.3 Consider the following fe-languages, L_1 and L_2.

1. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_1 = \{(\lambda, \{a\})\} \cup \{(a^i, \{a^{i+1}, a^ib^i\}) \mid i \geq 1\}$. Then, $L_1 = L(\mathcal{F}, \mathcal{E}_1) = \{a^nb^m \mid n \leq m \text{ and } n, m \geq 1\}$.

2. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_2 = \{(\lambda, \{b\})\} \cup \{(b^i, \{b^{i+1}, a^i b^i\}) \mid i \geq 1\}$. Then, $L_2 = L(\mathcal{F}, \mathcal{E}_2) = \{a^n b^m \mid n \geq m \text{ and } n, m \geq 1\}$. $\Diamond$

The following remark gives some useful properties of forbidding and enforcing sets and the languages that they define.

Remark 6.3.4 Let $\mathcal{F}$ and $\mathcal{F}'$ be forbidding sets, $\mathcal{E}$ and $\mathcal{E}'$ be enforcing sets, and u and v be words.

1. If $u \in Sub(w)$ and w is consistent with $\mathcal{F}$, then u is consistent with $\mathcal{F}$.
2. If $\mathcal{F}' \subseteq \mathcal{F}$, then $L(\mathcal{F}) \subseteq L(\mathcal{F}')$.
3. If $\mathcal{E}' \subseteq \mathcal{E}$, then $L(\mathcal{E}) \subseteq L(\mathcal{E}')$.
4. If $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{E}' \subseteq \mathcal{E}$, then $L(\mathcal{F}, \mathcal{E}) \subseteq L(\mathcal{F}', \mathcal{E}')$.
5. $L(\mathcal{F} \cup \mathcal{F}') = L(\mathcal{F}) \cap L(\mathcal{F}')$
6. $L(\mathcal{E} \cup \mathcal{E}') = L(\mathcal{E}) \cap L(\mathcal{E}')$
7. $L(\mathcal{F} \cup \mathcal{F}', \mathcal{E} \cup \mathcal{E}') = L(\mathcal{F}, \mathcal{E}) \cap L(\mathcal{F}', \mathcal{E}')$

Consider $\mathcal{F}$, $\mathcal{E}_1$, and $\mathcal{E}_2$ from Example 6.3.3. By property 7 of Remark 6.3.4, we have that $L = L_1 \cap L_2 = \{a^n b^n \mid n \geq 1\} = L(\mathcal{F}, \mathcal{E} \cup \mathcal{E}_2)$.

6.4 Finite Language FE-Systems and SLT

As it turns out, the languages that are defined by finite forbidding sets are regular languages. In fact, it can be shown that the languages that are defined by finite forbidding sets are locally testable languages, a subclass of regular languages. Consider the following theorem taken from [9].

Theorem 6.4.1 *Let $\mathcal{F}$ be a finite forbidding set. Then $L(\mathcal{F})$ is a locally testable language.*

Proof. Let $\mathcal{F}$ be a finite forbidding set. If $\mathcal{F} = \emptyset$, then $L(\mathcal{F}) = \Sigma^*$. The language Σ^* is strictly locally testable, and hence, locally testable. Suppose $\mathcal{F} = \{\{u\}\}$, i.e. $\mathcal{F}$ has only one forbidder $F = \{u\}$ which is a singleton. Then, $L(\mathcal{F}) = \Sigma^* \setminus \Sigma^* u \Sigma^*$. Let $|u| = k$ and $P = S = I = \Sigma^k \setminus \{u\}$. Since every word $w \in L(\mathcal{F})$ with $|w| \geq k$ has a k-length prefix and a k-length suffix that are not u and none of its interior subwords are u, it follows that $L(\mathcal{F})$ is strictly locally testable. Consider a nonempty finite $\mathcal{F}$ with not necessarily singleton forbidders. Each forbidder $F \in \mathcal{F}$ prohibits the combined presence of all its elements as subwords, so it defines the regular language $L(F) = \bigcup_{F \in \mathcal{F}} \Sigma^* \setminus \Sigma^* u \Sigma^*$, a union of local languages. Then, by Remark 6.3.4, $L(\mathcal{F}) = \bigcap_{F \in \mathcal{F}} L(F)$, i.e., $L(\mathcal{F})$ is a finite intersection of finite unions of strictly locally testable languages and is therefore locally testable. ∎

Similar to finite forbidding sets, the languages that are defined by finite enforcing sets are also locally testable languages.

Theorem 6.4.2 *Let $\mathcal{E}$ be a finite enforcing set. Then $L(\mathcal{E})$ is locally testable.*

Proof. Assume that $\mathcal{E}$ is finite. If $\mathcal{E} = \emptyset$, then $L(\mathcal{E}) = \Sigma^*$, which is strictly locally testable, and hence, locally testable. Otherwise, $\mathcal{E}$ contains at least one enforcer (x, Y). Let k be twice the length of the longest string in any enforcer $\mathcal{E}$. Thus, $k = 2 \cdot max\{|y| \mid y \in Y \text{ for some } (x, Y) \in \mathcal{E}\}$. As $\mathcal{E}$ is finite and nonempty, k is well-defined.

Consider $w \in A^{\geq k}$. We show that the sets $Pref_k(w)$, $Suff_k(w)$, and $Int_k(w)$ determine whether $w \in L(\mathcal{E})$, and therefore, we show that $L(\mathcal{E})$ is locally testable.

First, we consider enforcing sets without brute enforcers. For every $(x, Y) \in \mathcal{E}$ we have to make sure that every occurrence of x in a word is embedded in some $y \in Y$. We prove that $L(\mathcal{E})$ is strictly locally testable by specifiying the sets P, S, and I. The set I consists of all words of length k which either don't have x in the middle or every occurrence of x in the middle is enclosed by some $y \in Y$. The set P consists of all words that either don't contain x in the first half or every occurrence of x in the first half is in some $y \in Y$. S can be defined similarly.

In the case that $\mathcal{E}$ contains brute enforcers, for every enforcer (λ, Y), we determine membership of w by considering the sets $Pref_k(w)$, $Suff_k(w)$, and $Int_k(w)$ and checking whether there is a word in any of these sets that has some $y \in Y$ as a subword. Such a language is locally testable. ■

Following directly from Theorem 6.4.1, Theorem 6.4.2, and Remark 6.3.4, we can conclude that the languages that are defined by fe-systems are regular languages, and further, they are locally testable languages, as locally testable languages are closed under intersection.

6.5 Subword Closure Defined by a Finite FE-System

Recall from Chapter 3 the subword closure of a subword constraint S,

$$S^{\otimes} = \{w \in \Sigma^* \mid \text{if } u \text{ is a subword of } w \text{ and } |u| = k, \text{ then } u \in S\}.$$

In this section we attempt to determine whether or not $S^{\otimes}$ is an fe-language. Consider the following theorem from [9]:

Theorem 6.5.1 *Let L be a strictly locally testable language, then there exists a finite fe-system $(\mathcal{F}, \mathcal{E})$ such that $L = L(\mathcal{F}, \mathcal{E})$.*

Proof. Assume that L is strictly locally testable. Then there exists $k \geq 1$ and sets $P, S, I \subseteq \Sigma^k$ such that $Pref_k(L) \subseteq P$, $Suff_k(L) \subseteq S$, and $Int_k(L) \subseteq I$. We construct a finite fe-system $(\mathcal{F}, \mathcal{E})$ and show that $L = L(\mathcal{F}, \mathcal{E})$. The construction below considers the various intersections and unions among the three sets. If any k-letter subword is not in any of the three sets, it must be a forbidden subword $(\mathcal{F}_1)$. The forbidders $\mathcal{F}_2$ ensure that if a k-letter subword is a prefix-only and not a suffix or an interior subword, then it appears at the beginning of the word only, and hence, any extension to the left is forbidden. Similarly, $\mathcal{F}_3$ forbids any extension to the right of subwords which are suffixes only. If a k-letter subword is not an interior

subword, but it is both a prefix and a suffix, then simultaneous extension on both sides is forbidden by $\mathcal{F}_4$. Construct:

$\mathcal{F}_1 = \{\{u\} \mid u \in \Sigma^k \setminus (P \cup S \cup I)\}$;

$\mathcal{F}_2 = \{\{au\} \mid u \in P \setminus (S \cup I) \text{ and } a \in \Sigma\}$;

$\mathcal{F}_3 = \{\{ub\} \mid u \in S \setminus (P \cup I) \text{ and } b \in \Sigma\}$; and

$\mathcal{F}_4 = \{\{aub\} \mid u \in (P \cap S) \setminus I \text{ and } a, b \in \Sigma\}$.

We construct enforcing sets to ensure that interior subwords which are not prefixes or suffixes are indeed interior and are being extended on both sides ($\mathcal{E}_1$). Similarly, suffixes that are not prefixes must be extended to the left ($\mathcal{E}_2$) and prefixes that are not suffixes must be extended to the right ($\mathcal{E}_3$). So that:

$\mathcal{E}_1 = \{(x, \{axb \mid a, b \in \Sigma\}) \mid x \in I \setminus (P \cup S)\}$;

$\mathcal{E}_2 = \{(x, \{ax \mid a \in \Sigma\}) \mid x \in S \setminus P\}$; and

$\mathcal{E}_3 = \{(x, \{xb \mid b \in \Sigma\}) \mid x \in P \setminus S\}$.

Then, let $\mathcal{F} = \bigcup_{i=1}^{4} \mathcal{F}_i$, and let $\mathcal{E} = \bigcup_{i=1}^{3} \mathcal{E}_i$. By the above construction, $L = L(\mathcal{F}, \mathcal{E})$. ∎

Recall from Proposition 6.2.1, that the subword closure is SLT. Therefore, by Theorem 6.5.1, there exists an fe-system $(\mathcal{F}, \mathcal{E})$ such that $S^{\otimes} = L(\mathcal{F}, \mathcal{E})$. As it turns out, $S^{\otimes}$ isn't only an fe-language, it is an f-language. We provide a finite forbidding system that defines a language equal to $S^{\otimes}$.

Proposition 6.5.2 *Let S be a subword constraint, a nonempty set of k-length words. The subword closure of S, $S^{\otimes}$, is an f-language.*

Proof. Follows almost directly from Theorem 6.5.1. After considering each forbbider and enforcer in the construction, we are left with only one forbbider that is nonempty, and no enforcers that are nonempty. Thus, if $\mathcal{F} = \{\{w\} \mid w \in \Sigma^k \setminus S\}$, then $L(\mathcal{F}) = S^{\otimes}$. ∎

Note that this proof of Proposition 6.5.2 requires the fact $S^{\otimes}$ is SLT. It is possible to show that $S^{\otimes}$ is an f-language without this fact. It can be done by showing that

$L(\mathcal{F}) = S^{\otimes}$. Note also that by Theorem 6.4.1, we have that $S^{\otimes}$ is LT. This is consistent with what we found earlier, as SLT languages are contained within LT languages.

Chapter 7

Applications

In this chapter we discuss some real-world applications of subword closed languages. We also discuss applications for the maximal languages D that we have given a complete structural characterization. We give specific examples of these applications.

7.1 Applications of Subword Closed Languages

The subword closure was originally introduced by Kari et al. in [14] as a characterization for *maximal bond-free languages*, which themselves were introduced earlier by Jonoska and Mahalingam in [11] as θ-k codes. In Section 5.3 we briefly discussed why θ-k codes are so desirable, as they prevent all of the unwanted hybridizations that θ-infix codes and θ-subword-k codes prevent, as well as some that they do not. This property of the subword closure disallowing unwanted hybridizations has been tested extensively in [19].

DNA coding, or sequence design, became popular after researchers became aware of the possibility of DNA computing [1]. This area of study has produced many different methods for solving complex problems. These include, but are not limited to; k-colorability problems and satisfiabiliy problems.

The subword closure in particular can be used for other, non-computational purposes. For example, in DNA databases, data is generally encoded as single-stranded DNA molecules and then stored in DNA test tubes, see Figure 7.1. In this use, it is preferable that no DNA molecules in the test tubes contain two short

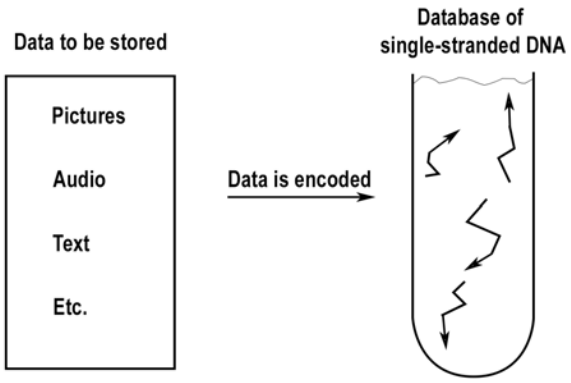

Figure 7.1: Each arrow represents a DNA strand with 5' to 3' orientation.

chains of k-length pairwise complementary nucleotides, as this would allow for a bond to form between these molecules, possibly causing corruption, and, therefore, a potential loss of data. Refer to Section 5.3 for examples of such a bonds.

The subword closure is also able to preserve properties in it's subwords that the subword constraint had. Consider the subword constraint

$$S = \{TCT, GTC, AAC, GAA, TTA, AGG, CGA\}.$$

In this example, the symbols A, G, C, and T represent the nitrogenous bases adenine, guanine, cytosine and thymine, respectively. S is constructed in such a way that no two words in S are complementary. For example, AGA is not found in S because it is the complement of TCT. As a result, the subword closure $S^{\otimes}$ contains arbitrarily long words that contain no two subwords of length at least 3 that are complementary. For example, the word $TCT(GAA)^n \in S^{\otimes}$, for all positive integers n.

Another example is provided by Zhang in [22]. Let S be a set of words with a certain continuity constraint; that is, words of length k in which no more than a certain number of consecutive a/t, or g/t nucleotides occur. Then, $S^{\otimes}$ will contain only words whose k-length subwords contain no more than a certain number of of consecutive a/t, or g/t nucleotides.

7.2 Applications of maximal D's with $D^* \subseteq S^{\otimes}$

These languages D, that are maximal with the property that $D^* \subseteq S^{\otimes}$, are useful when encoding data onto strands of DNA molecules.

Because the maximal languages D are represented as finite automata, it is easy to compute for any given ℓ, the set $D(\ell)$ of all words in D of length ℓ. If we need to encode sequences of n different objects, we can choose an ℓ such that $D(\ell)$ contains at least n words $w_1, \ldots, w_n$ of length ℓ. Then, any data sequence encoded into a DNA strand of the form $w_{i_1} w_{i_2} \ldots$ which is in $D(l)^*$, is also in $S^{\otimes}$.

Consider the following example taken from [18]. Say that we want to store an 8-bit color image of size 1024×512 pixels into $S^{\otimes}$. One way would be to choose a subset of appropriate size D_1 of $S^{\otimes}$, and to define 1024 single DNA strands $w_0, w_1, \ldots, w_{1023}$, with each one corresponding to one pixel row of the image. We have that each w_i is of the form

$$w_i = w_{i,0} w_{i,1} w_{i,2} \ldots w_{i,512}$$

such that each w_{i_j} is in $D_1(l)$, the word $w_{i,0}$ encodes the row number, and each $w_{i,j}$ with $j \geq 1$ encodes the color of the pixel $(i, j-1)$. It is conceivable that data of any form can be encoded onto single stranded DNA molecules. Storing data in this way has advantages over traditional methods, as data stored this way requires less space and less power.

Chapter 8

Concluding Remarks

We have discussed the problem of characterizing certain maximal languages within the subword closure of a given subword constraint. The characterization presented is a complete structural characterization. However, there might be several such maximal languages within the subword closure. Thus, the problem of computing "good" languages according to a preferred criteria, for example, that there are a large number of words in the maximal language of a given length, requires additional research. Also, the maximal language is not required to be a code. Thus, the problem of finding such maximal languages within the subword closure that are also codes requires further research. We have also shown that the subword closure is SLT. The relationship between the maximal languages D or their *-Kleene closures D^* and SLT is not known. This is a topic of our future research.

References

[1] Adleman, L.M., *Molecular computation of solutions to combinatorial problems*, Science 226 (1994) 1021-1024.

[2] Berstel, J., Perrin, D., *Theory of Codes*, Academic Press Inc., Orlando, FL (1985).

[3] Crochemore, M., Hancart, C., *Automata for matching patterns*, In: Rozenberg G., Salomaa A. (Eds.) Handbook of Formal Languages, Volume 1, pp. 399-462, Springer, Berlin (1997).

[4] Cui, B., Konstantinidis, S., *DNA coding using the subword closure operation.* In: Garzon, M.H. and Yan, H. (Eds.) DNA Computing. DNA13, Lecture Notes in Computer Science 4848 (2008) 284-289.

[5] Enhrenfeucht, A., Hoogeboom, H.J., Rozenberg, G., van Vugt, N. *Forbidding and enforcing* In: Winfree, E. and Gilford, D.K. (Eds.) DNA Based Computers V, AMS DIMACS 54 (2001) 195-206.

[6] Enhrenfeucht, A., Hoogeboom, H.J., Rozenberg, G., van Vugt N., *Sequences of languages in forbidding-enforcing families*, Soft Computing 5 (2) (2001) 121-125.

[7] Ehrenfeucht, A., Rozenberg, G., *Forbidding-enforcing systems*, Theoretical Computer Science 292 (2003) 611-638.

[8] Genova, D., *Defining languages by forbidding-enforcing systems*, In: L owe, B., Normann, D., Soskov, I., Soskova, A. (Eds.) CiE 2011, Lecture Notes in Computer Science 6735 (2011) 92-101 doi: 10.1007/978-3-642-28332-1_25.

[9] Genova, D., Hoogeboom, H.J., *Finite Language Forbidding-Enforcing Systems*, In: Kari, J., Manea, F., Petre, I. (Eds.) Unveiling Dynamics and Complexity, CiE 2017, Lecture Notes in Computer Science 10307 (2017) 258-269 doi: 10.1007/978-3-319-58741-7-25.

[10] Head, T., *Formal Language Theory and DNA: An analysis of the generative capacity of specific recombinant behaviors*, Bulletin of Mathematical Biology 49 (6) (1987) 737-759 doi: 10.1007/bf02481771.

[11] Jonoska, N., Mahalingham, K., *Languages of DNA based code words.*, In: Chen, J. and Reif, J. (Eds.) 9th International Meeting on DNA Based Computers, DNA9, Lecture Notes in Computer Science 2943 (2004) 61-73.

[12] Jonoska, N., Mahalingam, K., Chen, J. *Involution codes: with application to DNA coded languages*, Natural Computing 4 (2005) 141-162 doi: 10.1007/s11047-004-4009-9.

[13] Kari, L., Konstantinidis, S., Sosik, P. *Bond-free Languages: formalizations, maximality and construction methods*, Tech. Report 2004-01, Department of Mathematics and Computer Science, Saint Mary's University, Halifax, Canada (2004).

[14] Kari, L., Konstantinidis, S., Sosik, P. *Bond-free Languages: formalizations, maximality, and construction methods*, International Journal of Foundations of Computer Science 16 (5) (2005) 1039-1070.

[15] Kari, L., Seki, S., *Schema for parallel insertion and deletion: revisited* International Journal of Foundations of Computer Science 22 (7) (2011) 1655-1668.

[16] Kleene, S. C., *Representation of events in nerve nets and finite automata*, Automata Studies, Princeton University Press, Princeton NJ (1996) 2-42.

[17] Simovici, D., Tenney, R., *Theory of Formal Languages with Applications*, World Scientific (1999)

[18] Konstantinidis, S., Santean, N., *Computing Maximal Kleene Closures that are embeddable in a given subword-closed language*, Natural Computing 12 (2013) 211-222.

[19] Mahalingham, K., *Involution Codes with Application to DNA Strand Design*, Ph.D. Thesis, Department of Mathematics, University of South Florida, Tampa FL, USA (2004).

[20] Mateescu, A., Salomaa, A., *Formal Languages: an Introduction and a Synopsis*, In: Rosenberg, G., Salomaa, A. (Eds.) Handbook of Formal Languages, Volume 1, pp. 1-38 (1997).

[21] McNaughton, R., Papert, S., *Counter-Free Automata*, MIT Press, Cambridge, MA (1971).

[22] Zhang D.Y. *Towards Domain-Based Sequence Design for DNA Strand Displacement Reactions*, In: Sakakibara, Y. and Mi, Y. (Eds.) DNA Computing and Molecular Programming, DNA16, Lecture Notes in Computer Science 6518 (2011) 176-186.

About the Author

Rhys Jones began his undergraduate education at the University of North Florida in the fall of 2014. He enjoyed taking many undergraduate mathematics courses, but his favorite ones were Discrete Mathematics and Advanced Calculus. Rhys also studied Knot Theory during his undergraduate education and chose this area for his undergraduate thesis. Under the direction of Dr. Genova, he completed his Capstone Paper and Undergraduate Exit Presentation titled "Seifert Surfaces and the Alexander-Conway Polynomial". Rhys graduated with a Bachelor of Science in Mathematics and Statistics from the University of North Florida in Spring 2017.

During his graduate studies, Rhys worked as a teaching assistant for the University of North Florida's Mathematics Department. As a teaching assistant, Rhys taught recitation sessions for several classes, including College Algebra and Calculus, and graded work for a variety of undergraduate math classes, one of which was Discrete Biomathemtics. During this time he worked with Dr. Daniela Genova researching Maximal Bond-Free Languages. Rhys presented this research at the 52nd Joint Annual Meetings of the FL MAA and FTYCMA, which was held on the campus of Polk State College, Lakeland, FL on February 15-16, 2019. This presentation would later become the topic of this thesis.

Rhys has been accepted into the Ph.D. program in Mathematics at the University of South Florida. Studying Seifert Surfaces and the Alexander Polynomial sparked his initial interest in Knot Theory. He gained an interest in Formal Language Theory while taking the undergraduate course Discrete Biomathematics, and later, the graduate course Topics in Applied Algebra. Rhys hopes to continue studying Knot Theory and Formal Languages in the future, and is particularly interested in pursuing Knot Theory as it relates specifically to DNA.

Rhys lives in Jacksonville, FL with his family and two cats.