7-1-2004

Involution Codes with Application to DNA Strand Design

Kalpana Mahalingam
University of South Florida

Involution Codes: with Application to DNA Strand Design

by

Kalpana Mahalingam

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Mathematics
College of Arts and Sciences
University of South Florida

Major Professor: Nataša Jonoska, Ph.D.
Gregory McColm, Ph.D.
Masahico Saito, Ph.D.
Richard Stark, Ph.D.
Stephen Suen, Ph.D.

Date of approval:
July 1, 2004

Keywords: Codes, Watson-Crick Involution, DNA codes

Dedication

To my mom.

Acknowledgments

A journey in life is made easier when you have good company. This thesis is a result of four years of work whereby i have been accompanied and supported by many people. It is my wonderful aspect that i now have an opportunity to express my gratitude to all of them who traveled with me these four years.

The first person I would like to thank is my advisor Dr. Natasha Jonoska. She is the best advisor any student can have. She could not even realize how much I have learned from her. Besides of being an excellent supervisor, Natasha was as close as a relative and a good friend to me. I am really glad that I have come to get to know her in my life.

I would also like to thank the other members (Dr. Stephen Suen, Dr. Masahico Saito, Dr. Gregory McColm and Dr. Richard Stark) of my PhD committee who monitored my work and took effort in reading and providing me with valuable comments on earlier versions of this thesis. My special thanks to Dr. Junghuei Chen from University of Delaware for letting me use his laboratory and to teach me the experiments. I would also like to thank his students for being very helpful.

David Kephart: I have no words to thank him. Whenever I have a problem, the first person that comes to my mind is him. He was always there for me, right from teaching me Latex, discussing my research work, giving good tips about how to give a good speech and much more. Thanks David!!!

Life is not complete without good friends. I have wonderful friends who helped me a lot in so many different ways. Ed, Kalin, Ami, Kevin, Mitch, Daniela, Savita: Thank You All. I would also like to thank Nancy, Denise, Beverly, Mary Ann, Aya and Frances, who work in the Math office. A special thanks to Jim.

I am grateful to my uncle and aunt from Miami. If not for her food I would have starved most of the times. Thanks aunty!! I am also grateful to my brother, sis-in-law, mom, granny, my sis and my cousin Naresh for their moral support all these years. Special thanks to my sis Gayathri for putting up with my mood swings for the past one year.

A Special thanks to NSF for supporting me all summer all these four years through the grants EIA-0086015 and EIA-0074808.

Table of Contents

Involution Codes With Application To DNA Strand Design

Kalpana Mahalingam

Abstract

The set of all sequences that are generated by a biomolecular protocol forms a language over the four letter alphabet $\Delta = \{A, G, C, T\}$. This alphabet is associated with the natural involution mapping θ, $A \to T$ and $G \to C$ which is an antimorphism of Δ^*. In order to avoid undesirable Watson-Crick bonds between the words the language has to satisfy certain coding properties. Hence for an involution θ we consider involution codes: θ-infix, θ-comma-free, θ-k-codes and θ-subword-k-codes which avoid certain undesirable hybridization. We investigate the closure properties of these codes and also the conditions under which both X and X^+ are the same type of involution codes. We provide properties of a splicing system such that the language generated by the system preserves the desired properties of code words. Algebraic characterizations of these involution codes through their syntactic monoids have also been discussed. Methods of constructing involution codes that are strictly locally testable are given. General methods for generating such involution codes are given and the information capacity of these codes show to be optimal in most cases. A specific set of these codes was chosen for experimental testing and the results of these experiments are presented.

Chapter 1

Introduction

1.1 Background And Motivation

DNA is a crucial molecule in living cells. Proteins are the fundamental agents of life. The information that defines the primary structure of every protein is encoded in the DNA. DNA is a double stranded sequence of 4 nucleotides, and a portion of DNA sequence encodes the information that determines the sequence of amino acids of the protein encoded by that particular gene. DNA contains the genetic information that defines the proteins.

DNA consists of polymer chains, referred to as DNA strands. A chain is composed of nucleotides, and nucleotides may differ only in their bases. The information in a DNA molecule is stored in a sequence of nucleotides also called by their chemical group, base A,G,C,T (adenine, guanine, cytosine and thymine) joined together by phosphodiester bonds. A single strand of DNA, i.e. a chain of nucleotides, has also a "beginning"(usually denoted by 5') and an "end" (denoted by 3'), and so the molecule is oriented. The nucleotides come in complementary pairs(A is complementary to T and C is complementary to G). This is the well known Watson-Crick complementarity. Two single-stranded DNA molecules with opposite orientation join together (hybridize) through hydrogen bonds and form a double stranded molecule which in space appears in double helix. When double stranded molecules are heated to $95°C$, they denature into single-stranded molecules. If single stranded molecules are cooled, they seek their complement and re-anneal into double stranded form. These properties are used in many biomolecular experiments and are also used in many models of DNA computers and in DNA nanotechnology.

In most DNA computations, there are three basic stages. The first is encoding the input data using single stranded DNA, then performing the computation using bio-operations and finally decoding the result. In such computations, one of the main

problems is associated with the design of the oligonucleotides such that mismatched pairing due to Watson-Crick complementarity is minimized. In laboratory experiments, the complementarity of the bases may pose potential problems if some DNA strands can form non-specific hybridization and partially anneal to strands that are not their complete complements. This type of hybridization can occur during a polymerase chain reaction, self-assembly step or in the extraction process. Many authors have addressed this problem and proposed various solutions. A common approach has been to use Hamming distance as a measure for uniqueness [3, 7, 8, 11, 32]. Deaton et.al. [7, 11] used genetic algorithms to generate a set of DNA sequences that satisfy predetermined Hamming distance. Marathe et. al. [36] also used Hamming distance to compute combinatorial bounds of DNA sequences, and they used dynamic programing for design of the strands used in [32]. Seeman's program [44] generates sequences by testing overlapping subsequences to enforce uniqueness. This program is designed for producing sequences that are suitable for complex three-dimensional DNA structures, and the generation of suitable sequences is not as automatic as the other programs have proposed. Feldkamp et.al. [9] also uses the test for uniqueness of subsequences and relies on tree structures in generating new sequences. Ruben et.al. [43] use a random generator for initial sequence design, and afterwards check for unique subsequences with a predetermined properties based on Hamming distance. One of the first theoretical observations about number of DNA code words satisfying minimal Hamming distance properties was done by Baum [3]. Experimental separation of "good" codes that avoid intermolecular cross hybridization on big pool of random strands was reported in [6].

In [16], Kari et.al. introduce a theoretical approach to the problem of designing code words. Based on these ideas and code-theoretic properties, a computer program for generating code words is being developed [19, 26]. Another algorithm based on backtracking, for generating such code words is also developed by Li [30]. Every biomolecular protocol involving DNA or RNA generates molecules whose sequences of nucleotides form a language over the four letter alphabet $\Delta = \{A, G, C, T\}$. The Watson-Crick complementarity of the nucleotides defines a natural involution mapping θ, $A \mapsto T$ and $G \mapsto C$ which is an anti-morphism of Δ^*. Undesirable Watson-Crick bonds (undesirable hybridizations) can be avoided if the language satisfies

certain coding properties. In particular for DNA code words, no involution of a word is a subword of another word, or no involution of a word is a subword of a composition of two words. These properties are called θ-infix and θ-comma-free respectively. The case when a DNA strand may form a hairpin, (i.e. when a word contains a reverse complement of a subword) was introduced in [19] and is called θ-subword-k-code here.

For words representing DNA sequences we use the following convention. A word u over Δ denotes a DNA strand in its $5' \to 3'$ orientation. The Watson-Crick complement of the word u, also in orientation $5' \to 3'$ is denoted with $\overleftarrow{u}$. For example if $u = AGGC$ then $\overleftarrow{u} = GCCT$. There are two types of unwanted hybridizations: intramolecular and intermolecular. The intramolecular hybridization happens when two sequences, one being a reverse complement of the other appear within the same DNA strand (see Fig. 1.1). In this case the DNA strand forms a hairpin.

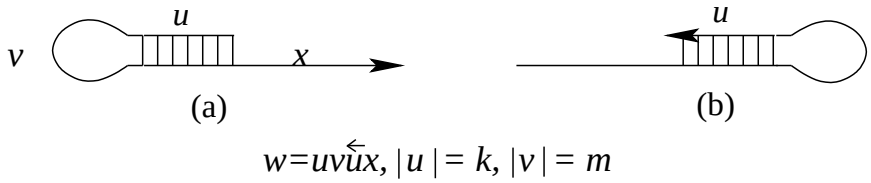

Figure 1.1: Intramolecular hybridization (θ-subword-k-code): (a) the reverse complement is at the beginning of the $5'$ end, (b) the reverse complement is at the end of the $3'$. The $3'$ end of the DNA strand is indicated with an arrow.

Two particular intermolecular hybridizations are of interest (see Fig.1.2). In Fig.1.2 (a) the strand labeled u is a reverse complement of a subsequence of the strand labeled v, and in the same figure (b) represents the case when u is the reverse complement of a portion of a concatenation of v and w.

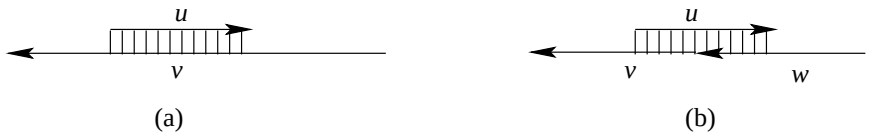

Figure 1.2: Two types of intermolecular hybridization: (a) (θ-infix) one code word is a reverse complement of a subword of another code word, (b) (θ-comma-free) a code word is a reverse complement of a subword of a concatenation of two other code words. The $3'$ end is indicated with an arrow.

DNA codes that avoid all kinds of unwanted partial bindings was introduced in [18] and was called θ-k-code. Note that for any word w over the alphabet Δ and for an anti-morphic involution θ, $\theta(w)$ denotes the Watson-Crick complementarity of the strand w. Hence, we together call θ-infix (comma-free) and θ-(subword)-k-codes to be involution codes for any involution map θ.

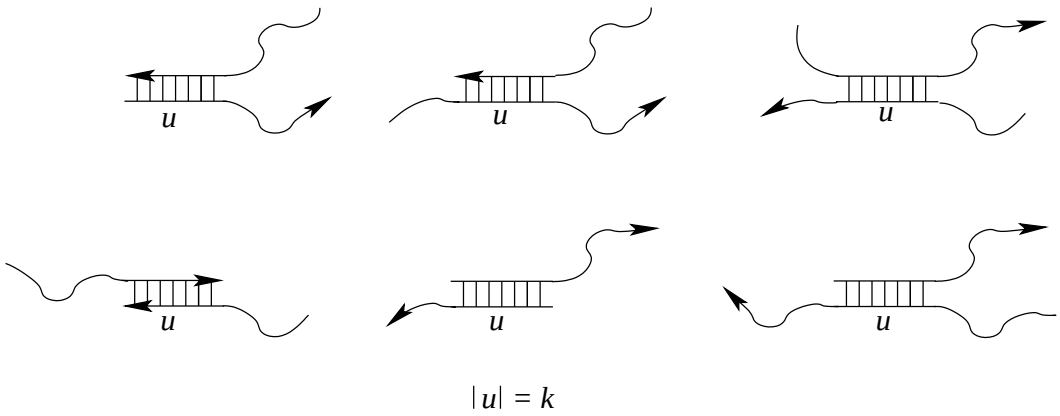

Figure 1.3: Various cross hybridizations of molecules one of which contains subword of length k and the other its complement.

In Section 1.2 of this chapter, we present some basic definitions required for this research. For more details, refer to [5, 10, 12, 16, 21, 29].

In Chapter II, Section 2.1 we make several observations about the closure properties of these involution codes. In particular, we concentrate on properties of languages that are preserved under concatenation. If a set of DNA strands has "good" coding properties then the same property will be preserved under arbitrary ligation of the strands. In Section 2.2 we provide necessary and sufficient conditions for a finite set of words to generate (by concatenations) an infinite set of code words. In Section 2.3 we investigate the properties of codes which are not θ-comma-free but can be split in to sub-codes which are θ-comma-free and can be used for bio-computing experiments. Section 2.4 investigates the necessary and sufficient conditions for preserving these "good" properties under splicing.

Chapter III investigates the algebraic characterization of the involution codes through their syntactic monoids. Section 3.2 discuss ways to construct involution codes that are strictly locally testable and we use the characterization of the syntactic monoid of locally testable languages to characterize these involution codes. Section

3.3 describes the properties of the syntactic monoid of certain type of involution codes X and the properties of syntactic monoid of $X^+ \cup \theta(X^+)$ when X^+ is θ-infix and θ-comma-free.

In Chapter IV, Section 4.2 describes several methods to generate involution codes. These sets of code words also provide sufficient informational entropy such that they can be used to encode binary strings bits $\rightarrow$ symbols. Sets of molecules obtained by the described methods in this section were tested for cross-hybridization experimentally. The results of the experiment are shown in Section 4.3.

1.2 Definitions and Basic Concepts

1.2.1 Codes

An alphabet set Σ is a finite non-empty set of symbols. A word u over Σ is a finite sequence of symbols in Σ. We denote by Σ^* the set of all words over Σ, and by Σ^+ the set of all non empty words over Σ. The empty word is denoted by 1. We note that with the concatenation operation on words, Σ^* is the free monoid and Σ^+ is the free semigroup generated by Σ. The length of a word $u = a_1...a_n$ is n for all $a_i \in \Sigma$ and is denoted by $|u|$. We denote by Σ^k the set of all words in Σ of length k.

Definition 1.2.1 (Code)
A subset X of the free monoid Σ^* is a code over the finite alphabet set Σ if for all $n, m \geq 1$ and for all $x_1, ..., x_n, x'_1, ..., x'_m \in X$, $x_1...x_n = x'_1...x'_m \Leftrightarrow n = m$ and $x_i = x'_i$ for $i = 1, ..., n$.

This means that X is a code if every word in X^+ can be written uniquely as a product of words in X. In other words X^+ is a free subsemigroup of Σ^+. Throughout this work we consider $X \subseteq \Sigma^+$ to be a code.

Example 1.2.2 For an alphabet set $\Sigma = \{a, b\}$, $X = \{ab, baa\}$ is a code. But $Y = \{a, ab, ba\}$ is not a code since the word $aba = (ab)a = a(ba) \in X^*$ has two distinct factorizations in X.

The following definition was used in [18, 20, 21].

Definition 1.2.3 *We define the prefix, suffix and subword of a word as*

$$\begin{aligned} \mathit{Pref}(w) &= \{u \mid \exists v \in \Sigma^*, uv = w\} \\ \mathit{Suff}(w) &= \{u \mid \exists v \in \Sigma^*, vu = w\} \\ \mathit{Sub}(w) &= \{u \mid \exists v_1, v_2 \in \Sigma^*, v_1 u v_2 = w\} \end{aligned}$$

We define prefix, suffix and subword of a language X as

$$\begin{aligned} Pref(X) &= \bigcup_{w \in X} Pref(w) \\ Suff(X) &= \bigcup_{w \in X} Suff(w) \\ Sub(X) &= \bigcup_{w \in X} Sub(w) \end{aligned}$$

We define proper prefix, proper suffix and proper subword as

$$\begin{aligned} PPref(X) &= Pref(X) \setminus X \\ PSuff(X) &= Suff(X) \setminus X \\ PSub(X) &= Sub(X) \setminus X \end{aligned}$$

Similarly we have for a word w,

$$\begin{aligned} Pref_k(w) &= Pref(w) \cap \Sigma^k \\ Suff_k(w) &= Suff(w) \cap \Sigma^k \\ Sub_k(w) &= Sub(w) \cap \Sigma^k \end{aligned}$$

Note that for $X, Y \subseteq \Sigma^*$, $XY = \{xy : x \in X, y \in Y\}$.

A set in which no element is a proper left (right) factor of another element is called a prefix (suffix) set. In other words, no two elements in the set are incomparable in the prefix (suffix) ordering.

Definition 1.2.4 (Prefix Code)
Let $X \subseteq \Sigma^+$. The set X is a prefix code if $X \cap X\Sigma^+ = \emptyset$.

Definition 1.2.5 (Suffix Code)
Let $X \subseteq \Sigma^+$. The set X is a suffix code if $X \cap \Sigma^+X = \emptyset$.

A set in which no word appear as a substring of any other word is called an infix code.

Definition 1.2.6 (Infix Code)
Let $X \subseteq \Sigma^+$. The set X is an infix code if $(\Sigma^*X\Sigma^+ \cup \Sigma^+X\Sigma^*) \cap X = \emptyset$.

In other words, X is an infix code if none of the words in X is a proper subword of any word in X. Note that if X is an infix code then X is both prefix and suffix code. A set in which no word appears as a substring across the concatenation of two words is called a comma-free code.

Definition 1.2.7 (Comma-free Code)
Let $X \subseteq \Sigma^+$. The set X is a comma-free code if $\Sigma^+ X \Sigma^+ \cap X^2 = \emptyset$.

Example 1.2.8 For $X = \{ab, baa\}$ over the alphabet set $\Sigma = \{a, b\}$, X is prefix, suffix and infix code. Note that X is not comma-free since $ba(ab)aa \in X^2$ with $ab \in X$.
For $Y = \{bbab, baa\}$ over the alphabet set $\Sigma = \{a, b\}$, Y is comma-free since $Y^2 = \{b^2ab^3ab, b^2ab^2a^2, ba^2b^2ab, ba^2ba^2\}$ and for all $y \in Y$, $pyq \notin Y^2$ for all $p, q \in \Sigma^+$.

Let X be a language such that $X \subseteq \Sigma^+$. We define the context of a word in X.

Definition 1.2.9 (Context of a Word)
For any word $w \in \Sigma^*$, context of w in X is given by $C_X(w) = \{(u, v) : uwv \in X, u, v \in \Sigma^*\}$.

Example 1.2.10 Let $X = \{bbab, baa\}$. Then $C_X(ba) = \{(b, b), (1, a)\}$.

Definition 1.2.11 *For $w \in \Sigma^*$, $R_X(w) = \{u \in \Sigma^* : wu \in X\}$ is the right context of w in X. Similarly, $L_X(w) = \{u \in \Sigma^* : uw \in X\}$ is the left context of w in X.*

The following was defined in [12] and used in [10].

Definition 1.2.12 (Solid Code)
A code X is a solid code if

(i) *X is an infix code*

(ii) $\mathrm{PPref}(X) \cap \mathrm{PSuff}(X) = \emptyset$.

Example 1.2.13 $X = \{ac, abb\}$ over the alphabet set $\Sigma = \{a, b, c\}$ is solid since for all $x, y \in X$, $x \notin \mathrm{PSub}(X)$ and $\mathrm{PPref}(x) \cap \mathrm{PSuff}(y) = \emptyset$.

Definition 1.2.14 (Relative Solidity)
A code X is solid relative to a language L if

(i) *$w = ypuqz$ in L with both u and puq in X can hold only if $pq = 1$ and*

(ii) *$w = ypuqz$ in L with pu and uq in X and $u \neq 1$ can hold only if $pq = 1$.*

In other words, if u has a non trivial context in X then it cannot have a non trivial context in L with that context of X.

Example 1.2.15 Let $X = \{ac, abb\}$ and $L = \{bac, bbc\}$ over the alphabet set $\Sigma = \{a, b, c\}$. Note that $b(ac) \in L$ with $ac \in X$ but $bac \notin X$ and $\text{PPref}(X) \cap \text{PSuff}(X) = \emptyset$ such that both the conditions of definition 1.2.14 are satisfied. Hence X is solid relative to L.

The following was defined in [12] and used in [10].

Definition 1.2.16 (Join Relative to a Language)
A string w in Σ^* is join relative to a language X, if $w \in \text{Sub}(X)$ and for all $(u, v) \in C_X(w)$ then both u and v are also in X.

Note that for an infix code X such that $X \subseteq \Sigma^+$ and for all $w \in X$, context of w is trivial in X and so w is join relative to X.

Example 1.2.17 Let $X = \{ac, abb\}$ over the alphabet set $\Sigma = \{a, b, c\}$. Both abb and ac are join relative to X since X is solid.

Definition 1.2.18 (Join of a Language)
A word w in a set $X \subseteq \Sigma^+$ is a join of X if w is join relative to X^*. $J(X)$ will denote the set of all joins in the set X.

Note 1.2.19 *Recall that when X is a code, $J(X)$ is comma-free subset of X (see [12]). $J(X)$ is not necessarily the maximal comma-free subset of X.*

Example 1.2.20 Let $X = \{aab, aba, bab\}$ over the alphabet set $\Sigma = \{a, b\}$. Note that $J(X) = \{aab\}$ but $Y = \{aab, bab\} \subseteq X$ is the maximal comma-free subset of X since $\Sigma^+ Y \Sigma^+ \cap Y^2 = \emptyset$.

Example 1.2.21 Let $X = \{ac, abb, bba\}$ over the alphabet set $\Sigma = \{a, b, c\}$. It is easy to observe that $\Sigma^+ ac \Sigma^+ \cap X^2 = \emptyset$ and $\{ac\}$ is the maximal comma-free subset of X. Hence $J(X) = \{ac\}$ and $X_1 = X \setminus J(X) = \{abb, bba\}$.

We define $X = X_0$ and $X_1 = X_0 \setminus J(X_0)$. Similarly we define a recursive chain of subsets X_i , $i \geq 0$ of X such that $X_i = X_{i-1} \setminus J(X_{i-1})$ where $J(X_{i-1})$ is the join of X_{i-1}. If there is a k such that $X_{k+1} = \emptyset$ then X is called a join code of level k (see [12]). If $X = \bigcup_{k=o}^{\infty} J(X_k)$ then X is called a join code of infinite level.

Example 1.2.22 Let $X = \{bac, cb\}$ over the alphabet set $\Sigma = \{a, b, c\}$ and $X^2 = \{bacbac, baccb, cbbac, cbcb\}$. Note that $J(X) = \{bac\}$ and hence $X_1 = X \setminus J(X) = \{cb\}$. Then $J(X_1) = \{cb\}$. Hence $X = J(X) \cup J(X_1)$ can be completely split in two subsets which are comma-free.

Example 1.2.23 Let $X = \{aab, aba\}$ over the alphabet set $\Sigma = \{a, b\}$ and $X^2 = \{a^2ba^2b, a^2baba, aba^3b, aba^2ba\}$. Note that $a(aba)ab, ab(aab)a \in X^2$. The set X does not have a comma-free subset and hence $J(X) = \emptyset$.

1.2.2 Involution Codes

An involution $\theta : \Sigma \to \Sigma$ of a set Σ is a mapping such that θ^2 equals identity mapping, $\theta(\theta(a)) = a$, for all $a \in \Sigma$. A mapping $\theta : \Sigma^* \to \Sigma^*$ is a morphism if $\theta(uv) = \theta(u)\theta(v)$ and an antimorphism if $\theta(uv) = \theta(v)\theta(u)$ for all $u, v \in \Sigma^*$. The mapping θ can be extended to a morphic or antimorphic involution of Σ^*. Note that if $\theta : \Sigma^* \mapsto \Sigma^*$ is an involution then $\theta : \Sigma \mapsto \Sigma$, $\theta(\theta(a_1a_2...a_n)) = a_1a_2...a_n$ since for all $u = a_1...a_n \in \Sigma^*$ with $a_i \in \Sigma$ if and only if $\theta(\theta(a_i)) = a_i$ for all $a_i \in \Sigma$.

Note that the mapping $\nu : \Delta \to \Delta$ defined by $\nu(A) = T$, $\nu(T) = A$, $\nu(C) = G$ and $\nu(G) = C$ is an involution on Δ and can be extended to a morphic involution of Δ^*. Since the Watson-Crick complementarity appears in a reverse orientation, we consider another involution $\rho : \Delta^* \to \Delta^*$ defined inductively, $\rho(s) = s$ for $s \in \Delta$ and $\rho(su) = \rho(u)\rho(s)$ for all $s \in \Delta$, $u \in \Delta^*$. This involution is an antimorphism such that $\rho(uv) = \rho(v)\rho(u)$. The Watson-Crick complementarity then is the antimorphic involution obtained with composition $\nu\rho = \rho\nu$. Hence for a DNA strand u we have that $\rho\nu(u) = \nu\rho(u) = \overleftarrow{u}$. The involution ρ reverses the order of the letters in a word. Throughout this work, we consider $\theta : \Sigma^* \to \Sigma^*$ to be either a morphic or antimorphic involution. Note that whenever $\theta : \Sigma^* \to \Sigma^*$ is an involution, then θ maps symbols to symbols such that $\theta^2(a) = a$ for all $a \in \Sigma$ and for all $x \in \Sigma^*$, $|\theta(x)| = |x|$.

Definition 1.2.24 *Let $X \subseteq \Sigma^+$ be a finite set.*

(i) *The set X is called* θ*-infix if* $\Sigma^*\theta(X)\Sigma^+ \cap X = \emptyset$ *and* $\Sigma^+\theta(X)\Sigma^* \cap X = \emptyset$.

(ii) *The set X is called* θ*-comma-free if* $\Sigma^+\theta(X)\Sigma^+ \cap X^2 = \emptyset$.

(iii) *The set X is called strictly* θ *if* $X \cap \theta(X) = \emptyset$

The notions of θ-prefix and θ-suffix code can be defined naturally from the notions defined above. The notion of θ-infix and θ-comma-free were originally called as θ-compliance and θ-free in [16]. The θ-infix and θ-comma-free codes avoid unwanted hybridizations presented in Fig.1.2

The following was defined in [19] and used in [18, 20, 21, 26]. The set of θ-subword-k-codes avoid the formation of hairpin structure as in Fig 1.1.

Definition 1.2.25 (Subword Code)

Let $X \subseteq \Sigma^+$ be a finite set. The set X is called θ*-k-m-subword code if for all $u \in \Sigma^k$ we have $\Sigma^* u \Sigma^i \theta(u) \Sigma^* \cap X = \emptyset$ for $1 \leq i \leq m$.*

The set X is called θ*-subword-k-code if $i \geq 1$.*

The following was defined in [18] and used in [20, 21]. The θ-k-codes avoid all kinds of unwanted hybridizations in Fig 1.3.

Definition 1.2.26 (θ-k-Code)

Let $X \subseteq \Sigma^+$. The set X is a θ*-k-code for some $k > 0$ if $Sub_k(X) \cap Sub_k(\theta(X)) = \emptyset$.*

Example 1.2.27 Let $X = \{aa, baa\}$ be a language over the alphabet set $\Sigma = \{a, b\}$ and for a morphic θ with $a \to b$ and $b \to a$, $\theta(X) = \{bb, abb\}$. Note that X is θ-infix since none of the subwords of X are in $\theta(X)$ and X is θ-comma-free since $X^2 = \{a^4, a^2ba^2, ba^4, ba^2ba^2\}$ and none of the words in $\theta(X)$ appears as a subword of any word in X^2. $\mathrm{Sub}_2(X) \cap \mathrm{Sub}_2(\theta(X)) = \emptyset$, hence X is θ-subword-2-code and θ-3-code.

Definition 1.2.28 (θ-Solid Code)

A code X is a θ*-solid code if*

(i) *X is θ-infix*

(ii) *$Pref(X) \cap Suff(\theta(X)) = \emptyset$ and $Suff(X) \cap Pref(\theta(X)) = \emptyset$.*

Definition 1.2.29 (θ-Solid Relative to L)
A code X is θ-solid relative to a language L if

(i) *$w = ypuqz$ in L with $u \in \theta(X)$ and $puq \in X$ then $pq = 1$ and*

(ii) *$w = ypuqz$ in L with $pu \in X$ and $uq \in \theta(X)$ and $u \neq 1$ then $pq = 1$.*

Note that from observation 1.2.19, for a code X, $J(X)$ is comma-free. In a similar way we would like to define $J_\theta(X)$ such that $J_\theta(X)$ is θ-comma-free. The authors in [10] define $J_\theta(X)$ as $J(X) \setminus \theta(J(X))$ for $\theta = \rho\nu$. $J_\theta(X)$ defined as above is not θ-comma-free in general for any (morphic or antimorphic) involution θ. For example, consider $X = \{aab, bab, abbb\}$. For an antimorphic involution θ with $a \to b$ and $b \to a$, $\theta(X) = \{abb, aba, aaab\}$. Note that $\Sigma^+\{aab, abbb\}\Sigma^+ \cap X^2 = \emptyset$ but $\Sigma^+X\Sigma^+ \cap X^2 \neq \emptyset$. Hence $J(X) = \{aab, abbb\}$ is comma-free subset of X and $J_\theta(X) = J(X)$ since $\theta(J(X)) \cap J(X) = \emptyset$. But $J_\theta(X)$ is not θ-comma-free since, $aab\theta(aab)b = aab.abbb \in (J_\theta(X))^2$.

We define $J_\theta(X)$ for a given set X such that $J_\theta(X)$ is θ-comma-free.

Definition 1.2.30 (θ-Join Relative to X)
A string w in Σ^* is a θ-join relative to a language X, if $w \in \mathrm{Sub}(X)$ and for all $(u, v) \in C_X(\theta(w))$, then both u and v are also in X.

Definition 1.2.31 (θ-Joins of X)
For a given $X \subseteq \Sigma^+$, we define $J_\theta(X) = \{w \in X : w$ is a θ-join relative to $X^*\}$. We call $J_\theta(X)$ θ-join of X.

Note 1.2.32 *If $X \cup \theta(X)$ is a code then $J_\theta(X)$ is θ-comma-free.*

Proof: Suppose $J_\theta(X)$ is not θ-comma-free, then there are $x, y, z \in J_\theta(X)$ such that $a\theta(z)b = xy$ for some $a, b \in \Sigma^+$. Since $z \in J_\theta(X)$ with $a\theta(z)b \in X^*$, $a, b \in X$ in which case xy has two distinct factorizations in $X \cup \theta(X)$ which contradicts that $X \cup \theta(X)$ is a code. If $a, b \notin X$ then $z \notin J_\theta(X)$. Hence $J_\theta(X)$ is θ-comma-free. □Note that $J_\theta(X)$ is not necessarily the maximal θ-comma-free subset of X.

Example 1.2.33 Let $X = \{abb, aab, aba\}$ over the alphabet set $\Sigma = \{a, b\}$ and for an antimorphic θ with $a \to b$ and $b \to a$, $\theta(X) = \{abb, bab, aab\}$. Note that $Y = \{aab, abb\}$ is the maximal θ-comma-free subset of X. But $J_\theta(X) = \{aab\}$.

Similarly as in 1.2.19 we write $X = X_0$ and $X_1 = X \setminus J_\theta(X)$. We define X_i, $i \geq 0$, a chain of descending subsets of X and $X_{k+1} = X_k \setminus J_\theta(X_k)$ where $J_\theta(X_k)$ is a θ-join of X_k. We call $J_\theta(X_k)$ as θ-comma-free code at level k. Also when θ is identity the θ-comma-free code at level 1 are nothing but $J(X)$. An example of a code X and involution θ such that $X = \bigcup_{i=0}^{2} J_\theta(X)$ is given section 2.3. If there is a k such that $X_{k+1} = \emptyset$ then X is called a θ-k-split code. If $X = \bigcup_{k=o}^{\infty} J_\theta(X_k)$ then X is called a θ-infinite-split code.

1.2.3 Syntactic Semigroups

Let X be a language such that $X \subseteq \Sigma^+$. We define the syntactic congruence of X.

Definition 1.2.34 (Syntactic Congruence)
The syntactic congruence of a set $X \subseteq \Sigma^+$ is denoted by P_X and is defined by $u \equiv v(P_X) \Leftrightarrow C_X(u) = C_X(v)$.

Definition 1.2.35 (Syntactic Monoid)
Syntactic monoid of X is the quotient monoid $M(X) = \Sigma^*/P_X$ with operation $[x][y] = [xy]$. For $x \in \Sigma^*$, $[x]$ denote the P_X equivalence class of x.

Definition 1.2.36 (Residue of X)
Let $W(X) = \{x \in \Sigma^* : C_X(x) = \emptyset\}$ i.e. $x \in W(X)$ iff $x \notin \mathrm{Sub}(X)$.

Note that if $W(X) \neq \emptyset$ then $W(X)$ represents a class for P_X and is the zero of $M(X)$.

Example 1.2.37 Let $X = \{ab^*\}$ over the alphabet set $\Sigma = \{a, b\}$. Then
$C_X(ab^*) = \{(1, b^*)\}$
$C_X(b^*) = \{(ab^*, b^*)\}$
Hence $M(X) = \{0, [1], [ab^*], [b^*]\}$.

Note that for a regular language X, $M(X)$ is the transition monoid (see [42]) of the minimal deterministic finite automaton (see [5, 42]) of X. The above definition of the syntactic congruence P_X can be defined for an arbitrary subset X of any semigroup S. If the syntactic congruence is the equality relation then we call the set X to be a disjunctive subset of S. If $X = \{x\}$ for some $x \in \Sigma^*$ and if P_X is

the equality relation then we say that x is a disjunctive element of S. For more on syntactic monoid we refer the reader to [5, 29, 42].

It is a well known fact that X is a regular language if and only if $M(X)$ is finite (see [29, 42]). For any set X and its syntactic monoid $M(X)$, $\eta : \Sigma^* \to M(X)$ is the natural surjective syntactic morphism defined by $x \to [x]$. Note that for any X, X is a union of P_X classes.

Chapter 2

Properties Of Coded Languages

In this chapter we look at various properties of involution codes. The results in Sections 2.1, 2.2 and 2.4 are published in [18, 21]. In Section 2.1 we deal with the closure properties of these involution codes under various operations. Section 2.2 gives us various ways of generating infinite set of involution codes with one of the "good" properties from a given finite set of involution codes with the same "good" property. Section 2.3 investigates the properties of codes that are not θ-comma-free but can be split into subcodes that are θ-comma-free. In Section 2.4 we look at the properties of the involution codes that are preserved under splicing operation.

2.1 Closure Properties

In this section we consider several closure properties of the involution codes under various operations. We concentrate on closure properties of union and concatenation of "good" code words. From practical point of view, we would like to know under what conditions two sets of available "good" code words can be joined (union) such that the resulting set is still a "good" set of code words. Also, whenever ligation of strands is involved, we need to consider concatenation of code words. In this case it is useful to know under what conditions the "good" properties of the code words will be preserved after arbitrary ligations. The following table shows closure properties of these languages under various operations.

Table 2.1 Closure Properties

	θ-subword-k-code	θ-infix	θ-comma-free	θ-k-code
Union	Yes	No	No	No
Intersection	Yes	Yes	Yes	Yes
Complement	No	No	No	No
Concatenation $(XY, X \neq Y)$	No	Y/N	No	No
Kleene *	No	No	Yes	No

Most of the properties included in the table are straight forward. We add a few comments for clarification.

- θ-subword-k-codes are closed under arbitrary union since every word in the union of θ-subword-k-codes do not form the hairpin structure.

- All involution codes except θ-subword-k-codes are not closed under union. For example let $X_1 = \{aa\}$ and $X_2 = \{bbb\}$ over the alphabet set $\Sigma = \{a, b\}$ and with antimorphic involution $\theta : a \mapsto b, b \mapsto a$. It is easy to check that X_1 and X_2 are θ-infix, θ-comma-free and θ-2-code. But $X_1 \cup X_2$ is not. Since $\theta(aa)$ is a subword of bbb , $X_1 \cup X_2$ is not θ-infix. Note that $(X_1 \cup X_2)^2 = \{aaaa, aabbb, bbbaa, bbbbbb\}$ and $aa\theta(aa)b \in (X_1 \cup X_2)^2$. Hence $(X_1 \cup X_2)$ is not θ-comma-free. Also $aa, \theta(aa) \in \text{Sub}(X_1 \cup X_2)$ which implies $(X_1 \cup X_2)$ is not θ-2-code.

- The involution codes are closed under intersection since a subset of an involution code with one of the "good" properties is again an involution code with the same property.

- It is easy to check that none of the involution codes are closed under complement.

- θ-subword-k-code languages are not closed under concatenation and Kleene*. For example consider the set $X = \{aba^2\} \subseteq \{a, b\}^*$ with the morphic involution $\theta : a \mapsto b, b \mapsto a$. Then X is θ-subword-2-code. But $X^2 = \{aba^3ba^2\}$ is not θ-subword-2-code, i.e. X^2 contains $(ab)a^3\theta(ab)a$.

- For a morphic θ, the family of θ-infix languages are closed under concatenation. *Proof:* Assume that X and Y are θ-infix for a morphic involution θ. Suppose XY

is not θ-infix then there exists $x_1, x_2 \in X$ and $y_1, y_2 \in Y$ such that $a\theta(x_1y_1)b = x_2y_2$ for some $a, b \in \Sigma^*$. Since θ is morphic, $a\theta(x_1)\theta(y_1)b = x_2y_2$. Then either $\theta(x_1)$ is a subword of x_2, or $\theta(y_1)$ is a subword of y_2. First case contradicts that X is θ-infix and the second case contradicts that Y is θ-infix.

- When θ is an antimorphism, θ-infix languages are not closed under concatenation. Consider the following example: $X_1 = \{a^2, ab\}$ and $X_2 = \{b^2, aba\}$ with the antimorphic involution $\theta : a \mapsto b, b \mapsto a$. Then $ab^3 \in X_1X_2$ and $bab^3 \in \theta(X_1X_2)$ and hence X_1X_2 is not θ-infix.

- It is also easy to check that if X and Y $(X \neq Y)$ are θ-comma-free (θ-k-code), then XY may not be θ-comma-free (θ-k-code). For example, take $X = \{aba\}$ and $Y = \{ab, bb\}$ with $\theta(X) = \{bab\}$ and $\theta(Y) = \{aa, ab\}$. Note that X and Y are both θ-comma-free and θ-3-code. $XY = \{abaab, ababb\}$ and $\theta(XY) = \{abbab, aabab\}$. But XY is not θ-comma-free since $ab\theta(ababb)aab \in (XY)^2$. Also $aba, \theta(aba) \in \mathrm{Sub}(XY)$, hence XY is not θ-3-code.

- Note that none of the involution codes other than θ-comma-free languages are closed under Kleene*. The proof will be given in Section 2.2

The next proposition which is a stronger version of Proposition 10 in [16], shows that for an antimorphic θ, concatenation of two distinct θ-infix or θ-comma-free languages is θ-infix or θ-comma-free whenever their union is θ-infix or θ-comma-free respectively. Necessary and sufficient conditions under which a θ-infix language is closed under Kleene* operation are considered in the next section.

Proposition 2.1.1 *Let $X, Y \subseteq \Sigma^+$ be two θ-infix (θ-comma-free) languages for a morphic or antimorphic θ. If $X \cup Y$ is θ-infix (θ-comma-free) then XY is θ-infix (θ-comma-free).*

Proof: Suppose XY is not θ-infix then there are $x_1, x_2 \in X$ and $y_1, y_2 \in Y$ such that $x_1y_1 = a\theta(x_2y_2)b$ for some $a, b \in \Sigma^*$ not both equal to 1. Then either $\theta(x_2)$ is a subword of x_1 or x_1y_1 or y_1. All cases contradict the assumption that $X \cup Y$ is θ-infix. Similarly we can prove for the θ-comma-free case. □

2.2 Generating Infinite Set of Code Words

It is easy to note that other than the θ-comma-free codes none of the θ-infix, θ-k-code and θ-subword-k-code are closed under arbitrary concatenation. In this section we investigate what are the properties of a finite set of "good" code words X that can generate an infinite set of code words X^+ with the same "good" properties. In practice, it is much easier to generate a relatively small set of code words that has certain properties (i.e. in case of DNA or RNA, mismatched hybridization is avoided), and if we know that any concatenation of such words would also satisfy the requirements, the process of generating code words could be rather simplified. Hence, we give necessary and sufficient conditions for X such that X^+ is θ-(subword)-k-code or θ-infix.

We have the following observations.

Lemma 2.2.1 In the following we assume that $k \leq \min\{ |x| : x \in X\}$.

(i) When θ is identity, X is an infix (comma-free) code if and only if X is θ-infix (comma-free).

(ii) When X is such that $X = \theta(X)$ then X is θ-infix (comma-free) if and only if X is infix (comma-free).

(iii) X is strictly θ-infix if and only if $\Sigma^*\theta(X)\Sigma^* \cap X = \emptyset$.

(iv) If X is strictly θ-comma-free then X and $\theta(X)$ are strictly θ-infix and $\theta(X)$ is θ-comma-free.

(v) If X is strictly θ-infix then X^* is both θ-prefix and θ-suffix code.

(vi) If X is θ-k-code, then X is θ-k'-code for all $k' > k$.

(vii) X is a θ-k-code if and only if $\theta(X)$ is a θ-k-code.

(viii) If X is strictly θ such that X^2 is θ-subword-k-code, then X is strictly θ-k-code.

(ix) If X is a θ-k-code then both X and $\theta(X)$ are θ-infix, θ-subword-k-code, θ-prefix-k and suffix-k-code for any $m \geq 1$. If $k \leq \frac{|x|}{2}$ for all $x \in X$ then X is θ-comma-free and hence avoids the cross hybridizations as shown in Fig.1.1 and 1.2.

(x) If X is a θ-k-code then X and $\theta(X)$ avoids all cross hybridizations of length k shown in Fig. 1.3 and so all cross hybridizations presented in Fig. 2 of [22].

Proof :

(i) When θ is identity, $\theta(X) = X$. Since X is infix, $\Sigma^+X\Sigma^* \cap X = \emptyset$ and $\Sigma^*X\Sigma^+ \cap X = \emptyset$ implies $\Sigma^+\theta(X)\Sigma^* \cap X = \emptyset$ and $\Sigma^*\theta(X)\Sigma^+ \cap X = \emptyset$ since $\theta(X) = X$. Similar proof works for $\theta(X)$-comma-free.

(ii) Similar to 2.2.1(i).

(iii) X is strictly θ-infix if and only if $\Sigma^+X\Sigma^* \cap X = \emptyset$ and $\Sigma^*X\Sigma^* \cap X = \emptyset$ and $X \cap \theta(X) = \emptyset$ if and only if $\Sigma^*\theta(X)\Sigma^* \cap X = \emptyset$.

(iv) X is not θ-infix then $x = a\theta(y)b$ for some $a \in \Sigma^+$. Hence $xx = a\theta(y)ba\theta(y)b$ which contradicts that X is θ-comma-free. Similar proof shows for $\theta(X)$ is θ-infix and θ-comma-free.

(v) To show that X^* is θ-prefix (i.e.) to show that $X^* \cap \theta(X^*)\Sigma^* = \emptyset$. Suppose X^* is not θ-prefix code then there exists $x_1x_2...x_n = \theta(y_1...y_m)b$ for $x_i, y_j \in X, i = 1, .., n, j = 1, ..., m$ and $b \in \Sigma^*$. For a morphic θ, $x_1x_2...x_n = \theta(y_1)...\theta(y_m)b$ implies either x_1 is a subword of $\theta(y_1)$ or $x_1 = \theta(y_1)$ or $\theta(y_1)$ is a subword of x_1. All cases contradict our assumption that X is strictly θ-infix. Similarly we can prove that X^* is θ-suffix code.

(vi) Since X is θ-k-code, $\mathrm{Sub}_k(X) \cap \mathrm{Sub}_k(\theta(X) = \emptyset$. Hence $\mathrm{Sub}_{k'}(X) \cap \mathrm{Sub}_{k'}(\theta(X) = \emptyset$ for all $k' \geq k$ which implies X is θ-k'-code.

(vii) X is θ-k-code if and only if $\mathrm{Sub}_k(X) \cap \mathrm{Sub}_k(\theta(X)) = \emptyset$ if and only if $\mathrm{Sub}_k(\theta(X)) \cap \mathrm{Sub}_k(\theta(\theta(X))) = \emptyset$ since θ is an involution, $\theta(\theta(X)) = X$. Hence X is θ-k-code if and only if $\theta(X)$ is θ-k-code.

(viii) Suppose X is not θ-k-code then there exists $x_1 \in \mathrm{Sub}_k(X) \cap \mathrm{Sub}_k(\theta(X))$ such that $x = ax_1b$ and $y = c\theta(x_1)d$ for some $x, y \in X$. Hence $xy = ax_1bc\theta(x_1)d$ which contradicts our assumption that X^2 is θ-subword-k-code.

(ix) Since X is a θ-k-code for $k \leq min\{|x| : x \in X\}$, X is θ-infix and θ-subword-k-code. Let $k \leq \frac{|x|}{2}$ for all $x \in X$. Suppose X is not θ-comma-free then there are

$x, y, z \in X$ such that $xy = a\theta(z)b$ for some $a, b \in \Sigma^+$. Hence $\theta(z) = x_2y_1$ for $x = x_1x_2$ and $y = y_1y_2$. Then either $|x_2| \geq k$ or $|y_1| \geq k$, both contradict that X is not a θ-k-code. □

The Lemma below shows that if X is θ-comma-free then X^2 is "almost" θ-infix. The difference is in $+$ vs $*$ in 1.2.24(i) of Definition 1.2.24.
All properties below refer to a finite set $X \subseteq \Sigma^+$.

Lemma 2.2.2 *If X is θ-comma-free then $X^2 \cap \Sigma^+\theta(X^2)\Sigma^+ = \emptyset$.*

Proof: Suppose lemma does not hold, then there are $x_1, x_2, y_1, y_2 \in X$ such that $x_1x_2 = a\theta(y_1y_2)b$ with $a, b \in \Sigma^+$. When θ is morphic, $x_1x_2 = a\theta(y_1)\theta(y_2)b$, and when θ is anti-morphic, we have $x_1x_2 = a\theta(y_2)\theta(y_1)b$. In both cases X would not be θ-comma-free. Hence $X^2 \cap \Sigma^+\theta(X^2)\Sigma^+ = \emptyset$. □

Note that the converse of the above need not be true. For example consider $X = \{a^3b, a^2b^2\}$ with θ being morphism $a \mapsto b, b \mapsto a$. Then $X^2 \cap \Sigma^+\theta(X^2)\Sigma^+ = \emptyset$ since all words in X^2 are of length 8, but X is not θ-comma-free since $a^2b^2a^3b = a^2\theta(a^2b^2)ab$.

Lemma 2.2.3 *If X is strictly θ-infix then X^n is strictly θ-infix for all $n > 1$.*

Proof: If the lemma does not hold then X^* is not strictly θ-infix for some n. This means that there are $x, y \in X^n$ such that $x = s\theta(y)t$ for some $s, t \in \Sigma^*$ (not both equal to 1). Let $x = x_1...x_n = s\theta(y_1...y_n)t$ with $x_i, y_i \in X$ then one of $\theta(y_i)$ is a subword of some x_j which is a contradiction to X being θ-infix. □

The Kleene $*$ closure of X contains the union of all X^n and in order for it to be θ-infix we need stronger properties. The next proposition is somewhat a stronger version of Lemma 1(ii) and Proposition 1 in [16]. Proposition 1 in [16] proves (i) $\Rightarrow$ (iii) of the following proposition.

Proposition 2.2.1 *The following are equivalent:*

(i) X is strictly θ-comma-free

(ii) X^+ is strictly θ-infix

(iii) X^+ is strictly θ-comma-free

Proof: (i)⇒(ii). Suppose X is strictly θ-comma-free. Hence, by observation 2.2.1(iv) X is strictly θ-infix. By Lemma 2.2.3 X^n is strictly θ-infix for all $n \geq 1$. Suppose X^+ is not strictly θ-infix then there exist $x, y \in X^+$ such that $x = a\theta(y)b$ for some $a, b \in \Sigma^*$ (not both equal to 1). Let $x = x_1...x_n$ and $y = y_1...y_m$, for $x_i, y_j \in X$, hence for some y_i either

(a) $\theta(y_i)$ is a subword of x_j for some j,

(b) $\theta(y_i)$ is a subword of $x_j x_{j+1}$,

(c) $\theta(y_i)$ is a subword of $x_{j-1} x_j x_{j+1}$

The cases (a) and (c) contradict the fact that X is θ-infix and (b) contradicts to X being θ-comma-free. Hence X^+ is strictly θ-infix.

(ii)⇒(iii). Given that X^+ is strictly θ-infix, suppose X^+ is not θ-comma-free. Then there exist $x, y, z \in X^+$ such that $xy = a\theta(z)b$ for some $a, b \in \Sigma^+$. Let $x = x_1...x_n$, $y = y_1...y_m$ and $z = z_1...z_r$ with $x_i, y_i, z_i \in X$. Then $\theta(z_i)$ is a subword of one of $x_j x_{j+1}$, x_j, y_s, $y_s y_{s+1}$ or $x_n y_1$. All cases contradict that X^+ is θ-infix. Hence X^+ is strictly θ-comma-free.

(iii)⇒(i). Obvious, since X is a subset of X^+.□

The following two properties are similar observations as Proposition 6 and Proposition 9 in [16]. We use the following definition that was introduced in [16] and used in [21].

Definition 2.2.4 *Let Σ be the alphabet set and θ be the involution and $X \subseteq \Sigma^*$. Define*

(i) $\underline{X_{is}} = PSuff(\theta(X)) \cap PPref(X)$

(ii) $\underline{X_{ip}} = PSuff(X) \cap PPref(\theta(X))$

(iii) $\underline{X_s} = \bigcup_{x \in PSuff(X)} R_{\theta(X)}(x)$

(iv) $\underline{X_p} = \bigcup_{x \in PPref(X)} L_{\theta(X)}(x)$

Where $R_X(x)$ = the set of all right context of x in X and $L_X(x)$ = the set of all left context of x in X.

Proposition 2.2.2 *Let $X \subseteq \Sigma^+$ then X is θ-infix and $X_{ip} X_{is} \cap \theta(X) = \emptyset$ if and only if X is θ-comma-free.*

Proof: Let X be θ-infix and $X_{ip}X_{is} \cap \theta(X) = \emptyset$. Suppose X is not θ-comma-free and there are $x, y, z \in X$ such that $xy = a\theta(z)b$ for some $a, b \in \Sigma^+$. Since X is θ-infix,$\theta(z)$ is not a subword of x or y. Let $\theta(z) = z_1z_2$ such that $az_1 = x$ and $z_2b = y$. But $z_1z_2 \in \theta(X)$, so $z_2b \in X$ implies $z_2 \in X_{is}$ and $az_1 \in X$ implies $z_1 \in X_{ip}$. Hence $z_1z_2 \in X_{ip}X_{is} \cap \theta(X)$ which contradicts the hypothesis. Conversely, let X be θ-comma-free. Suppose $xy \in X_{ip}X_{is} \cap \theta(X)$. Then, $x \in X_{ip}$ implies that there are $u, v \in \Sigma^+$ such that $ux \in X$ and $xv \in \theta(X)$. For $y \in X_{is}$ there exists $w, r \in \Sigma^+$ such that $wy \in \theta(X)$ and $yr \in X$. Hence $uxyr \in X^2$ with $xy \in \theta(X)$ which is a contradiction with X being θ-comma-free.□

Proposition 2.2.3 *If X is θ-infix and $X_pX_s \cap \theta(X) = \emptyset$ then X is θ-comma-free.*

Proof: Suppose X is not θ-comma-free. Then there exists $x, y, z \in X$ such that $xy = a\theta(z)b$ for some $a, b \in \Sigma^+$. Since X is θ-infix, $\theta(z)$ is not a subword of x or y. Let $\theta(z) = z_1z_2$ for some z_1, z_2 such that $az_1 = x$ and $z_2b = y$ which implies $z_2 \in X_s$ and $z_1 \in X_p$. Hence $z_1z_2 \in X_pX_s \cap \theta(X)$ which is a contradiction with the initial assumption.□

The converse of the above proposition is not true. For example, $X = \{bba, bbab\}$ is θ-comma-free for an antimorphic θ mapping $a \mapsto b$ and $b \mapsto a$ with $\theta(X) = \{baa, abaa\}$. But $X_s = \{baa, a, aa\}$ and $X_p = \emptyset$ which implies $baa \in X_pX_s \cap \theta(X)$.

The following proposition investigates the case when the property of subword-k-m-codes are preserved with Kleene*. It turned out that conditions under which a subword-k-m-code is closed under concatenation with itself are somewhat more demanding than the ones for θ-comma-free and θ-infix. Considering 2.2.1(viii), 2.2.1(ix), 2.2.1(x) in Observation 2.2.1 these properties might turn out to be quite important.

Proposition 2.2.4 *Let k, m be positive integers and $X \subseteq \Sigma^*$ be such that for every word $x \in X$, $|x| \geq 2k + m$. Let*

$$L = \bigcup_{l=1}^{m} \bigcup_{i+j=2k+l} Suff_i(X)Pref_j(X) \tag{2.1}$$

Then X^* is θ-subword-k-m-code if and only if X is θ-subword-k-m-code and for all $y \in L$, $Pref_k(y) \cap Suff_k(\theta(y)) = \emptyset$.

Proof: Assume that X is θ-subword-k-m-code and for all $y \in L$, $\text{Pref}_k(y) \cap \text{Suff}_k(\theta(y)) = \emptyset$. Suppose X^* is not θ-subword-k-m-code. Then there exists $x \in X^*$ such that $x = x_1 us\theta(u)x_2$ where $|s| = l \leq m$ and $|u| = k$. We claim that this is impossible.

If $us\theta(u)$ is a subword of some $y \in X$, then this contradicts the property that X is θ-subword-k-m-code. If $us\theta(u)$ is a subword of some x_1x_2 for $x_1, x_2 \in X$ then there is a p such that $Pref_k(p) \cap \text{Suff}_k(\theta(p)) \neq \emptyset$ which is again a contradiction with the hypothesis. If $us\theta(u)$ is a subword of some $x_1x_2x_3$ then we have $u_2su_3 = x_2$ for $x_1, x_2, x_3 \in X$ for some $u_1u_2 = u$ and $u_3u_4 = \theta(u)$ and since $|u_2| < k, |u_3| < k$ and $|x_2| \geq 2k + m$ which implies $|s| > m$ which is a contradiction. Hence X^* is θ-subword-k-m-code.

Conversely, note that if X^* is θ-subword-k-m-code then X is θ-subword-k-m-code. Suppose there exists $x \in L$ such that $\text{Pref}_k(x) \cap \text{Suff}_k(\theta(x)) \neq \emptyset$. Then x is such that x is either a subword of some $y \in X$ or a subword of some y_1y_2 with $y_1, y_2 \in X$. Both cases are contradictory to the fact that X^* is θ-subword-k-m-code.□

Example 2.2.5 The set $X = \{AGTCA, AAGCT\} \subseteq \Delta^*$ is θ-subword-2-1-code for $\theta = \rho\nu$. It is easy to verify that $L =$\{AAAGC, CAAAG, TCAAA, GTCAA, TAGTC, CTAGT, GCTAG, AGCTA, AGTCA, AAGCT\} and for all $y \in L$, $\text{Suff}_2(y) \cap \text{Pref}_2(\theta(y)) = \emptyset$. Hence X^* is θ-subword-2-1-code.

The conditions under which codes with one of the coding properties in Definition 1.2.24 are closed under Kleene* are discussed above. But when is a language θ-subword-k-code, θ-infix and θ-comma-free all at once? The next two propositions try to give an answer to this question. The condition in the first proposition is quite strong due to the strong requirements. However, the condition is only sufficient, and may not be necessary.

Proposition 2.2.5 *Let X be a θ-2-code such that for all $x \in X$, $|x| \geq 3$. Then both X and X^+ are strictly*

(i) *θ-subword-k-code for $k \geq 3$.*

(ii) *θ-infix and θ-comma-free.*

Proof:

(i) By induction on the powers of X. Since $\mathrm{Sub}_2(X) \cap \mathrm{Sub}_2(\theta(X)) = \emptyset$ and $k \geq 3$, $\Sigma^* u \Sigma^m \theta(u) \Sigma^* \cap X = \emptyset$ for all $u \in \Sigma^k$ and for $m \geq 1$. Hence $X^1 = X$ is θ-subword-k-code. Consider X^2 and suppose that there exists an $x \in X^2$ such that $x = y_1 u y_2 \theta(u) y_3$ for some $y_1, y_3 \in \Sigma^*$, and $u \in \Sigma^k$, $y_2 \in \Sigma^m$ for $m \geq 1$. Let $x = x_1 x_2$ for $x_1, x_2 \in X$.

It is not the case that $x_1 = y_1$ and $x_2 = u y_2 \theta(u) y_3$ since X is θ-subword-k-code.

So suppose that $x_1 = y_1 u_1$, $x_2 = u_2 y_2 \theta(u) y_3$ for some u_1, u_2 such that $u_1 u_2 = u$ and $u_1 \neq 1$. Since $k > 2$ one of u_1 or u_2 has length at least 2. This contradicts our assumption that $\mathrm{Sub}_2(X) \cap \mathrm{Sub}_2(\theta(X)) = \emptyset$. Now the inductive step is done similarly. Assume X^n is θ-subword-k-code. Suppose there exists an $x = x_1 \cdots x_{n+1} \in X^{n+1}$ such that $x = y_1 u y_2 \theta(u) y_3$ for some $y_1, y_3 \in \Sigma^*$, $u \in \Sigma^k$, $y_2 \in \Sigma^m$, $m \geq 1$. Suppose $u \in Sub(x_i x_{i+1} .. x_{i+t})$ with $|u| > 2$. Then there is a subword of u say $u_j \in Sub(x_j)$, such that $|u_j| \geq 2$, and $\theta(u_j) \in \mathrm{Sub}(X)$. This is a contradiction to the condition that $\mathrm{Sub}_2(X) \cap \mathrm{Sub}_2(\theta(X)) = \emptyset$. So for every i, X^i is θ-subword-k-code. Hence $X^+ = \bigcup_{i=1}^{\infty} X^i$ is θ-subword-k-code.

(ii) Since $\mathrm{Sub}_2(X) \cap \mathrm{Sub}_2(\theta(X)) = \emptyset$ X is both θ-infix and θ-comma-free and by proposition 2.2.1 X^+ is θ-infix and θ-comma-free. □

The following observation is straight forward.

Proposition 2.2.6 *Let X be a θ-k-code for $k \leq \min\{|x| : x \in X\}$. Then X^+ is strictly θ-k-code if and only if X^2 is a strictly θ-k-code.*

Proof. One way is obvious. Suppose X^* is not θ-k-code then for some $x = x_1 x_2 ... x_n$, $y = y_1 y_2 ... y_m \in X^*$, there exists $u \in \mathrm{Sub}_k(x)$ and $v \in \mathrm{Sub}_k(\theta(y))$ such that $u = v$. The four cases we need to consider are:

(i) u being a subword of x_i for some i and v is a subword of $\theta(y_j)$ for some j.

(ii) u a subword of x_i and v a subword of $\theta(y_j y_{j+1})$.

(iii) u a subword of $x_i x_{i+1}$ and v a subword of $\theta(y_j)$.

(iv) u a subword of $x_i x_{i+1}$ and v a subword of $\theta(y_j y_{j+1})$.

All cases contradict that X^2 is a θ-k-code. Hence X^* is a θ-k-code. □

2.3 Levels of Involution-Comma-Free Codes

When information is encoded in single stranded DNA molecules, it can be decoded by applying the Watson-Crick complements of the code. If the code is not θ-comma-free, then we can decode the sequence using θ-comma-free code at level 1 first, then continue decoding by θ-comma-free code at level 2 and proceed the same way until the entire sequence is read. If the code cannot be completely split into subsets that are θ-comma-free then decoding the entire sequence might not be possible. The codes that can be completely split into finite number of θ-join codes that are θ-comma-free subsets allow the segmentation of the code to be made in a sequence of finite steps for which each step has the simplicity of a θ-comma-free segmentation.

A sequence of such extractions provides a sequence of θ-comma-free codes. The level of each code is its sequence number. We note that the θ-join code of a set is θ-comma-free and is called as *θ-comma-free codes at level 1.* When θ is identity the θ-comma-free code at level 1 of a set X is the join of X in [10, 12]. It is more natural to call the k-th subset of X which is θ- join of the $(k-1)$th subset, as *θ-comma-free code at level k.* When a set X can be completely split into θ-join codes, then X is called a *θ-split code.* A set is a *θ-k-split code* if it can be completely split into k θ-join codes that are θ-comma-free subcodes. A set is a *θ-infinite-split code* if it can be completely split into infinite θ-comma-free subcodes. When θ is identity, examples of such split codes are given in Section 1.2.2.

When decoding a message with a comma-free code, there will be no mistake in parsing the message by starting from the middle of the string, since no word will appear as a substring across the concatenation of two words. If the code is not comma-free, in some cases we can split it into subcodes which are comma-free [10, 12]. When information is encoded in single stranded DNA molecules, the information can be decoded by applying complementary strands to the codes. In that case θ-comma-free provides unique place for hybridization of the code strand so that none of the Watson-Crick complements of the words appear as substrings across concatenation of two words. If the code is not θ-comma-free, we show below that we can sometimes extract subcodes, that are θ-comma-free.

In this section we have several observations about the θ-comma-free codes of

various levels. We also look at the relativization concepts of these codes. We begin the section with a few examples.

Example 2.3.1 Let $X = \{aab, aba\}$ over the alphabet set $\Sigma = \{a, b\}$ with $X^2 = \{aabaab, aababa, abaaab, abaaba\}$ and for an antimorphic θ, $a \mapsto b$ and $b \mapsto a$, $\theta(X) = \{abb, bab\}$. Then $J_\theta(X) = \{aab\}$ is θ-comma-free. Hence $X_1 = X \setminus J_\theta(X) = \{aba\}$. We can check that $J_\theta(X_1) = \{aba\}$ and hence $X = J_\theta(X) \cup J_\theta(X_1)$ is a *θ-2-split-code* .

Example 2.3.2 Let $X = \{ab, aab, aba\}$ over the alphabet set $\Sigma = \{a, b\}$ and for an antimorphic θ, $a \to b$ and $b \to a$, $\theta(X) = \{ab, abb, bab\}$. Note that $J_\theta(X) = \{aab\}$. Then $X_1 = \{ab, aba\}$ and $J_\theta(X_1) = \emptyset$ since $a\theta(aba)a, ab\theta(ab)a \in X_1^2$. Hence X cannot be completely split in to θ-comma-free codes. Thus X is not a θ-split-code.

Proposition 2.3.1 *If X is such that $X \cup \theta(X)$ is a code then $\theta(J_\theta(X))$ is solid relative to X^*.*

Proof: Suppose $\theta(J_\theta(X))$ is not solid relative to X^*, then one of the conditions in Definition 1.2.14 is not satisfied. If condition (i) of Definition 1.2.14 is violated then there exists $w = yp\theta(u)qz \in X^*$ such that $p\theta(u)q, \theta(u) \in \theta(J_\theta(X))$ implies $yp, qz, y, z \in X^*$ since $u \in J_\theta(X)$ which contradicts our assumption that $X \cup \theta(X)$ is a code. If condition (ii) of Definition 1.2.14 is violated then there exists $w = yp\theta(u)qz \in X^*$ such that $p\theta(u), \theta(u)q \in \theta(J_\theta(X))$ and hence $y, qz, yp, z \in X^*$ which is a contradiction to our assumption that $X \cup \theta(X)$ is a code. $\Box$.

We have the following observations which are not difficult to prove. We assume that for a set X, $X \cup \theta(X)$ is a code throughout the rest of this section.

Note 2.3.3 **(i)** *$J_\theta(X) = J(X)$ when θ is identity.*

(ii) *When X is θ-comma-free, $J_\theta(X) = X$.*

(iii) *X is solid if and only if $\theta(X)$ is solid.*

(iv) *$X \cup \theta(X)$ is θ-comma-free if and only if $X \cup \theta(X)$ is comma-free and hence $X \cup \theta(X)$ is an infix code.*

(v) *If X is solid relative to a language L then for any $Y \subseteq X$, Y is solid relative to L.*

(vi) *Let $X = Y \cup \theta(Y)$. X is θ-comma-free if and only if X is solid relative to X^*. (see proposition 2 in [10]).*

Note that the converse of 2.3.3(iv) above is not true. For example $X = \{aa, aba\}$ with θ being antimorphic involution mapping $a \to b$ and $b \to a$ we have, $\theta(X) = \{bb, bab\}$. It is easy to check that $X \cup \theta(X)$ is a code but not θ-comma-free. It was proved in [12] that X is comma-free if and only if it is solid relative to X^*. We prove a similar result for a θ-comma-free code.

Proposition 2.3.2 *Let X be such that X is strictly θ. Then X is θ-comma-free if and only if X is θ-solid relative to X^2.*

Proof: Assume that X is θ-comma-free. To show that X is θ-solid relative to X^2. Let $ypuqz \in X^2$ with $puq \in X$, $u \in \theta(X)$. Then $pq \neq 1$ is a contradiction to the fact that X is θ-infix code. If $pu \in X$ and $uq \in \theta(X)$, then the case when p or q is 1 contradict that X is θ-infix. If $yz = 1$ then $puq \in X^2$ with $pu \in X$. In this case $pu \in \mathrm{Sub}(X)$ or $uq \in \mathrm{Sub}(X)$ which is a contradiction that X is θ-infix. When $p \neq 1$, $q \neq 1$, $y \neq 1$ and $z = 1$ with $ypuqz = ypuq \in X^2$, $pu \in X$ and $uq \in \theta(X)$ then $yp \in \mathrm{Sub}(X)$ or $uq \in \mathrm{Sub}(X)$ contradict that X is strictly θ-infix. Lastly, $p \neq 1$, $q \neq 1$, $y = 1$ and $z \neq 1$ with $ypuqz = puqz \in X^2$ contradict that X is θ-comma-free.

Conversely, assume that X is θ-solid relative to X^2. Suppose X is not θ-comma-free then there are $x, y, z \in X$ such that $xy = a\theta(z)b$ for $a, b \in \Sigma^+$. The case when $\theta(z)$ is a subword of x or y contradicts the condition 1 of Definition 1.2.29. The case when $\theta(z) = z_1 z_2$ such that $az_1 = x$, $z_2 b = y$ and $az_1 z_2 b \in X^2$ which contradicts condition 2 of Definition 1.2.29. □

Proposition 2.3.3 *If X is a θ-solid code then X is strictly θ-comma-free.*

Proof: Note that since $\text{Pref}(X) \cap \text{Suff}(\theta(X)) = \emptyset$, X is strictly θ. Suppose X is not θ-comma-free then there are $x, y, z \in X$ such that $xy = a\theta(z)b$. Then either $\theta(z)$ is a subword of x or a subword of y which contradicts that X is θ-infix , or $\theta(z) = z_1z_2$ such that $az_1 = x$ and $z_2b = y$ which implies $z_1 \in \text{Pref}(\theta(X)) \cap \text{Suff}(X)$ and $z_2 \in \text{Pref}(X) \cap \text{Suff}(\theta(X))$ which contradicts condition 2 of Definition 1.2.28. □

Note that the converse of the above proposition holds when θ is identity (see [12]) but not for any general θ. For example let $X = \{aa, baa\}$ and for an antimorphic $\theta : a \to b, b \to a$, $\theta(X) = \{bb, bba\}$. It is easy to check that X is θ-comma-free. But $ba \in \text{Pref}(X) \cap \text{Suff}(\theta(X))$ which contradicts condition 2 of definition 1.2.28.

Corollary 2.3.4 *If X is a θ-solid code then X is θ-solid relative to X^*.*

Proposition 2.3.4 *If $X \cup \theta(X)$ is solid relative to X^* then X is θ-comma-free.*

Proof: Suppose X is not θ-comma-free, then there exists $x, y, z \in X$ such that $xy = a\theta(z)b$ for $a, b \in \Sigma^+$. Then either $\theta(z) = z_1z_2$ such that $az_1 = x$ and $z_2b = y$ or $\theta(z)$ is a subword of y such that $r\theta(z)b = y$. The first case implies $az_1, z_1z_2, z_2b \in X \cup \theta(X)$ where $az_1z_2b \in X^*$ which contradicts condition 2 of 1.2.14 with $p = a$, $u = z_1$, $q = z_2$ and the second case implies that $r\theta(z)b, \theta(z) \in X \cup \theta(X)$ which contradicts condition 1 of 1.2.14 with $p = r$, $u = \theta(z)$ and $q = b$. □

Converse of the above proposition also holds true when θ is identity mapping (see [12]). The converse of the above proposition is not true in general for any θ. For example let $X = \{a^3b, a^2\}$ with $\theta(X) = \{b^2, ab^3\}$ and $X^2 = \{a^3ba^3b, a^3ba^2, a^5b, a^4\}$. Note that $X^2 \cap \Sigma^+\theta(X)\Sigma^+ = \emptyset$, hence X is θ-comma-free. Also, $y = (aa)(a(aa)b)(aa)$ $\in X^*$, with $p = a$, $u = aa \in X$ and $q = b$ which contradicts condition 1 of 1.2.14. Hence X is not solid relative to X^*.

The following proposition gives us the condition under which a split code is closed under Kleene*.

Proposition 2.3.5 *Let $X \subseteq \Sigma^+$ be such that $X \cup \theta(X)$ is a code and X is θ-infix. Let $J_\theta(X_i) \subseteq X$ and $J_\theta(X_i^*) \subseteq X^*$ be the θ joins of X_i and X_i^* respectively. Then*

(i) *$J_\theta(X^*) = X^*J_\theta(X)X^* \cup \{w \in X^* : \theta(w) \notin \text{Sub}(X^*)\}$ and*

(ii) $J_\theta(X_i^*) = X^*J_\theta(X_i)X^* \setminus J_\theta(X_{i-1}^*)$ *for* $i \geq 1$.

Proof: Recall that $J_\theta(X)$ is θ-comma-free when $X \cup \theta(X)$ is a code.

(i) We first show that $X^*J_\theta(X)X^* \cup \{w \in X^* : \theta(w) \notin \mathrm{Sub}(X^*)\} \subseteq J_\theta(X^*)$. If $w \in X^*$ such that $\theta(w) \notin \mathrm{Sub}(X^*)$ then $w \in J_\theta(X^*)$ since $a\theta(w)b \notin X^*$ for all $a, b \in \Sigma^+$. Let $x \in J_\theta(X)$ and $w = y_1xy_2$ for $y_1, y_2 \in X^*$. Then for all $a, b \in \Sigma^+$ and for a morphic θ, $a\theta(y_1xy_2)b = a\theta(y_1)\theta(x)\theta(y_2)b$ which is not in X^* since $x \in J_\theta(X)$ and $\Sigma^+J_\theta(X)\Sigma^+ \cap (J_\theta(X))^2 = \emptyset$. Hence $w \in J_\theta(X^*)$.

Suppose $w \in J_\theta(X^*)$ and $\theta(w) \in \mathrm{Sub}(X^*)$. To show that $w \in X^*J_\theta(X)X^*$, let $w = x_1x_2...x_n$, $x_i \in X$ such that $\theta(x_1x_2...x_n) \in \mathrm{Sub}(X^*)$. Observe that one of $x_i \in J_\theta(X)$. If none of the $x_i \in J_\theta(X)$ then at least one of the $\theta(x_i)$ is a proper subword of a word in X which contradicts that X is θ-infix. Hence $J_\theta(X^*) = X^*J_\theta(X)X^* \cup \{w \in X^* : \theta(w) \notin \mathrm{Sub}(X^*)\}$.

(ii) We show that $J_\theta(X_i^*) = X^*J_\theta(X_i)X^* \setminus J_\theta(X_{i-1}^*)$. Let $w \in J_\theta(X_i^*)$ where $w = x_1x_2...x_n$ for $x_j \in X$. Note that $w \notin J_\theta(X_{i-1}^*)$ since $J_\theta(X_i^*) = X_{i-1}^* \setminus J_\theta(X_{i-1}^*)$ by definition. Need to show that $w \in X^*J_\theta(X_i)X^*$. For $w = x_1x_2...x_n$, if $x_j \notin J_\theta(X_i)$ for all j then at least one of the $\theta(x_i)$ is a proper subword of a word in X which contradicts that X is θ-infix.

Conversely, suppose $w \in X^*J_\theta(X_i)X^* \setminus J_\theta(X_{i-1}^*)$. To show that $w \in J_\theta(X_i^*)$, let $w = y_1xy_2$ for $x \in J_\theta(X_i)$ and $y_1, y_2 \in X^*$. Then $a\theta(y_1xy_2)b = a\theta(y_1)\theta(x)\theta(y_2)b \notin X^*$ for all $a, b \in \Sigma^=$ such that $a\theta(y_1)$ or $\theta(y_2)b$ is not in X^*.□

Corollary 2.3.5 *If X is such that X is θ-infix and θ-k-split code then X^* is also θ-l-split code for some $l \leq k$.*

Proof: It is clear from the above proposition if $X = \bigcup_{i=o}^{k} J_\theta(X_i)$ then $X^* = \bigcup_{i=0}^{k} J_\theta(X_i^*)$.
□

We have given some partial results on these θ-level codes. There are other new problems that need to be considered. It will be interesting to look in to the properties of a given infinite code such that it is a θ-split code of finite or infinite level. Also algorithms could be studied that would decide whether the code is a split code of finite level. It would be also interesting to find the algebraic characterizations of such codes.

2.4 Codes Preserved by Splicing

The operation of splicing was introduced as a generative mechanism in formal language theory by Tom Head in [13]. Splicing systems were introduced as a model for the cut and paste activity made possible through the action of restriction enzymes and a ligase on double stranded DNA molecules. They were further developed in computational models (see for example [15, 38]). Splicing two strings means to cut them at two points specified by given substrings and to concatenate the obtained fragments crosswise.

Consider the following two DNA molecules:

$$5'\text{-}CCCCCTCGACCCCC\text{-}3'$$
$$3'\text{-}GGGGGAGCTGGGGG\text{-}5'$$

$$5'\text{-}AAAAAGCGCAAAAA\text{-}3'$$
$$3'\text{-}TTTTTCGCGTTTTT\text{-}5'$$

The restriction enzymes are able to recognize specific substrings of double stranded DNA molecules and cut these molecules at the specific site at the middle of these double stranded molecules producing "blunt" ends or "sticky" ends. The two enzymes TaqI and SciNI cut these molecules at the unique sites occurring in them and the following four fragments are produced:

$$5'\text{-}CCCCCT \qquad CGACCCCC\text{-}3'$$
$$3'\text{-}GGGGGAGC \qquad TGGGGG\text{-}5'$$

$$5'\text{-}AAAAAG \qquad CGCAAAAA\text{-}3'$$
$$3'\text{-}TTTTTCGC \qquad GTTTTT\text{-}5'$$

The recombination of these four molecules gives the new molecules below:

$$5'\text{-}CCCCCTCGCAAAAA\text{-}3'$$
$$3'\text{-}GGGGGAGCGTTTTT\text{-}5'$$

$$5'\text{-}AAAAAGCGACCCCC\text{-}3'$$
$$3'\text{-}TTTTTCGCTGGGGG\text{-}5'$$

In this section we investigate properties of involution codes that are preserved under splicing operation. In other words, we characterize the set of splicing rules so that the resulting language of the splicing system keeps the properties of the involution codes. We consider the question of determining the properties of the code words such that under splicing they produce new "good" code words, in other words, during the computational process the "good" encoding is not lost.

A *splicing system* is an ordered triple $\gamma = (\Sigma, A, \mathcal{R})$ where Σ is a finite alphabet, $A \subset \Sigma^*$ is a set of words called *axioms* and $\mathcal{R} \subseteq (\Sigma^*)^4$ is a set of splicing rules where $(\Sigma^*)^{(4)}$ is the direct product of $\Sigma^* \times \Sigma^* \times \Sigma^* \times \Sigma^*$.

The splicing operation σ is defined such that if $r = (u_1, u_2, v_1, v_2)$ then

$$\begin{matrix} x = x_1u_1u_2x_2 \\ y = y_1v_1v_2y_2 \end{matrix} \quad \text{rule } r \text{ produces } (\Rightarrow_r) \quad \begin{matrix} w_1 = x_1u_1v_2y_2 \\ w_2 = y_1v_1u_2x_2 \end{matrix}$$

and in this case we write $(x, y) \Rightarrow_r (w_1, w_2)$. For a language $L \subseteq \Sigma^*$ and a set of rules $\mathcal{R}$ we say that $\sigma(L)$ is obtained by single step splicing of L if $\sigma(L) = \{w \mid \exists r \in \mathcal{R},\ \exists x, y \in L,\ (x, y) \Rightarrow_r (w, w')\}$. Then the *language generated by the splicing system* γ with axiom set A is $L(\gamma) = \cup_{n \geq 0} \sigma^n(A)$.

For the background on splicing systems we refer the reader to [15, 38].

The following were defined in [18]

Definition 2.4.1 Base for Splicing Rules

Let X be a set and $B \subset \mathrm{Sub}(X)$ be a subset of words with $k = \min\{|x| : x \in B\} \geq 0$. We say that a set B is a base for splicing rules for X *if it is strictly θ and satisfies the following two properties:*

(i) $\mathrm{Sub}_s(\theta(B^2)) \cap \mathrm{Sub}_s(B^2) = \emptyset$ *for all* $s \geq k$

(ii) $\theta(B^2) \cap \mathrm{Sub}(X) = \emptyset$.

Definition 2.4.2 θ**-rules**

Let X be a set of words and B a base for splicing rules for X. Then a set $\mathcal{R}_B(X)$ which is a subset of B^4 is called a set of θ-rules *for X.*

Several observations about θ-rules

(i) The θ-rules can always be reflexive. This means for each rule (u_1, u_2, v_1, v_2) in $\mathcal{R}_B(X)$ the rules (u_1, u_2, u_1, u_2) and (v_1, v_2, v_1, v_2) can also be in $\mathcal{R}_B(X)$. This follows directly from the definition of θ-rules.

(ii) θ-rules can be symmetric. If $(u_1, u_2, u_3, u_4) \in \mathcal{R}_B(X)$ then $(u_3, u_4, u_1, u_2) \in \mathcal{R}_B(X)$. This also follows from the definition of θ-rules.

(iii) One way to obtain a set of θ-rules for X is the following. Let $m = \min\{|x| : x \in X\}$ and $k = \lceil \frac{m}{2} \rceil$.

Note that even if X is strictly θ, $\mathrm{Sub}_k(X)$ may not necessarily be so. We partition $\mathrm{Sub}_k(X)$ into three strictly θ subsets in the following way. Set $U_1 = \{x \mid x \notin \mathrm{Sub}_k(\theta(X))\}$ and $U_2 = \mathrm{Sub}_k(X) \setminus U_1$. Then by definition U_1 is strictly θ and for every $u \in U_2$ we have $\theta(u) \in U_2$. Now we partition U_2 into U_2' and U_2'' such that $u \in U_2'$ and $\theta(u) \in U_2''$. Hence $\mathrm{Sub}_k(X) = U_1 \cup U_2' \cup U_2''$ is a partition of $\mathrm{Sub}_k(X)$ into three strictly θ sets. We form the basis of the words used in the splicing rules from two of these sets.

Let W be either U_2' or U_2'' and let $B = U_1 \cup W$. Then B is a maximal set of subwords of X of length k that is strictly θ. Now we remove from B all words that violate the properties (1) and (2) of Definition 2.4.1. This may leave an empty set of B. In that case we consider the subwords of length $k-1$ and $k+1$ and repeat the procedure for obtaining B. This procedure ends either with a base B for splicing rules or, since X is finite, with no base for splicing rules for X.

(iv) A set K is called a *strong splicing base* if for all $x \in \Sigma^*$ and for all $u \in K$, if xu is a prefix of K^* then $x \in K^*$ and if ux is a suffix of K^* then $x \in K^*$. In [16] the strong splicing base is called simply a splicing base. We have to make a distinction in our case since the base for splicing rules is not necessarily a strong splicing base (see Example 2.4.3). However, the base for splicing rules

B obtained from the construction in 2.4.2(iii) above, with words of constant length, does give us a strong splicing base.

The need for the set of θ-rules $\mathcal{R}$ to be reflexive and symmetric comes naturally from the chemistry of the restriction enzymes. If an enzyme can cut one molecule, the ligase can recombine the same molecule back together, hence a reflexive operation. If two molecules x and y take part in a splicing operation (can be cut by an enzyme) then the same molecules written in the opposite order y and x are part of the same operation. Hence the symmetric operation. In the following we assume that the splicing rules are reflexive and symmetric.

Proposition 2.4.1 *Let $\gamma = (\Sigma, A, \mathcal{R}_B(A))$, be a splicing system with $\mathcal{R}_B(A)$ being the a set of θ-rules. If the axiom set A is strictly θ-infix then the language generated by the system $L(\gamma)$ is strictly θ-infix.*

Proof: Since $L(\gamma) = \cup_{n\geq 0}\sigma^n(A)$ we proceed by induction on n. splicing rules derivation steps. If $n = 0$ then $\sigma^0(A) = A$ and A is θ-infix by assumption. Assume that $\sigma^n(A)$ is θ-infix and suppose $\sigma^{n+1}(A)$ is not θ-infix. Then $\exists x, y \in \sigma^{n+1}(A)$ such that $x = s\theta(y)p$ for some words s and p. Since $\mathcal{R}_B(A)$ is reflexive, we can assume that both x and y have been obtained by splicing words from $\sigma^n(A)$. Let $x = x_1u_1u_2x_4$ and $y = y_1v_1v_4y_4$ for some $(u_1, u_2, u_3, u_4) \in \mathcal{R}_B(A)$ and $(v_1, v_2, v_3, v_4) \in \mathcal{R}_B(A)$. Then $x = x_1u_1u_4x_4 = s\theta(y_1v_1v_4y_4)p$. In what part of x can $\theta(v_1v_4)$ appear as a subword? There are several possibilities:

(i) $\theta(v_1v_4) \in \mathrm{Sub}(x_1u_1)$ or $\theta(v_1v_4) \in \mathrm{Sub}(u_4x_4)$.

(ii) $\theta(v_1v_4) \in \mathrm{Sub}(u_1u_4)$.

(iii) $\theta(v_1v_4) \in \mathrm{Sub}(x_1u_1u_4)$ or $\theta(v_1v_4) \in \mathrm{Sub}(u_1u_4x_4)$ but not case 1 or 2.

(iv) $\theta(v_1v_4) \in \mathrm{Sub}(x)$ but none of the previous cases.

The first case involves longer analysis, but it ends up violating the property 1 or property 2 from definition 2.4.1. The second and the third case violate the property 1 from Definition 2.4.1. The fourth case means that $u_1u_4 \in \mathrm{Sub}(\theta(v_1v_4)$ which again violates condition 1 of Definition 2.4.1. Hence, $\sigma^{n+1}(A)$ has to be θ-infix.

In order to show that $L(\gamma)$ is strictly θ it is sufficient to take the words s and p to be empty and then follow the above argument. $\square$

The following proposition provides conditions under which the language generated by a splicing system is θ-comma-free.

Proposition 2.4.2 *Let $\gamma = (\Sigma, A, \mathcal{R}_B(A))$ be a splicing system where $\mathcal{R}_B(A)$ is a set of θ-rules for A. Assume that the base B for the splicing rules is such that*

$$\{\theta(B^2)\} \cap \mathrm{Sub}(A^2) = \emptyset.$$

If the axiom set A is strictly θ-comma-free, then $L(\gamma)$ is also θ-comma-free.

Proof. The proof is similar to the above proof and is by induction on the number of derivation steps. If $n = 0$ then $\sigma^0(A) = A$ and A is θ-comma-free by assumption. Assume that $\sigma^n(A)$ is θ-comma-free and suppose $\sigma^{n+1}(A)$ is not θ-comma-free. Then there are $x, y, z \in \sigma^{n+1}(A)$ such that $xy = s\theta(z)p$, $s, p \in \Sigma^+$. Let $x = x_1u_1u_4x_4$, $y = y_1v_1v_4y_4$ and $z = z_1w_1w_4z_4$ for some $(u_1, u_2, u_3, u_4), (v_1, v_2, v_3, v_4)$ and $(w_1, w_2, w_3, w_4) \in \mathcal{R}_B(A)$

Let $\theta(y)$ be a subword of $u_4x_4y_1v_1$.

u_4x_4 and $y_1v_1 \in Sub(A)$ violates the property 2 from definition 2.4.1. If $u_4x_4 = r_1t_1...t_lr_2$ and $y_1v_1 = r_3q_1...q_mr_4$ such that $t_lr_2, r_3q_1 \in Sub(A)$, $t_i, q_i \in B$, then either

(i) $\theta(w_1w_4)$ is a subword of t_it_{i+1}, or $\theta(w_1w_4)$ is a subword of q_iq_{i+1}.

(ii) $\theta(w_1w_4)$ is a subword of r_2r_3, then $r_2r_3 \in Sub(A^2)$ which is a contradiction.

(iii) $\theta(w_1w_4)$ is a subword of t_lr_2 or $\theta(w_1w_4)$ is a subword of r_3q_1.

Then the first case violates property 1 of definition 2.4.1 , the second case violates $\{\theta(B^2)\} \cap \mathrm{Sub}(A^2) = \emptyset$ and the third case violates property 2 of definition 2.4.1.

Hence, $\sigma^{n+1}(A)$ is θ-comma-free.$\square$

We would like to point out that in [16] it was proven that if a strong splicing base is strictly θ-comma-free, then so is the language generated by the splicing system, provided that the set of axioms is a subset of the free monoid generated by the splicing base. The Proposition 2.4.2 does not require that the axiom set is a subset of the

free monoid generated by the base for the splicing rules, and therefore, the language generated by the splicing system is not necessarily a subset of this same monoid (as is the case in [16]). The following example shows a base for splicing rules that is not strong splicing base.

Example 2.4.3 Consider the alphabet $\Sigma = \{a, b, c\}$ with the morphic involution θ defined with $a \mapsto c$, $b \mapsto b$ and $c \mapsto a$. Let the axiom set $A = \{abaaba\}$. We choose a base for splicing rules to be $B = \{a, ab\}$. This is clearly a strictly θ-comma-free code, but it is not a strong splicing base. Consider $a \cdot ab \cdot a \in B^+$ with suffix aba. If we pick $u = a$ and $x = ba$ we have that ux is a suffix of a word in B^+ and $u \in B$ but $x \notin B^+$. Now, $\theta(B^2) = \{cc, ccb, cbc, cbcb\}$ and clearly the conditions of the definition 2.4.1 and Proposition 2.4.2 are satisfied. We can take $\mathcal{R} = B^{(4)}$ and the resulting generated splicing language is infinite (contains $aba^+ba \cup (ab)^+a$) and is strictly θ-comma-free.

The following proposition provides conditions under which the language generated by a splicing system is θ-k-code.

Proposition 2.4.3 *Let $\gamma = (\Sigma, A, \mathcal{R}_B(A))$ be a splicing system where $\mathcal{R}_B(A)$ is a set of θ-rules for A. Assume that the base B for the splicing rules is such that*

$$\mathrm{Sub}_k\theta(B^2) \cap \mathrm{Sub}_k(A) = \emptyset$$

If the axiom set A is θ-k-code, then $L(\gamma)$ is also θ-k-code.

Proof: The proof is by induction on the number of derivation steps. If $n = 0$ then $\sigma^0(A) = A$ and A is θ-k-code by assumption. Assume that $\sigma^n(A)$ is θ-k-code, and suppose $\sigma^{n+1}(A)$ is not θ-k-code. Then there are $x, y \in \sigma^{n+1}(A)$ such that $x' \in \mathrm{Sub}_k(x)$ and $y' \in \mathrm{Sub}_k(y)$ and $x' = \theta(y')$. Let $x = x_1u_1u_4x_4$, $y = y_1v_1v_4y_4$ for some (u_1, u_2, u_3, u_4), $(v_1, v_2, v_3, v_4) \in \mathcal{R}_B(A)$. Both $x, y \in \sigma^n(A)$ and the case when $x' \in \mathrm{Sub}_k(x_1u_1)$ and $y' \in \mathrm{Sub}_k(y_1v_1)$ violates that $\sigma^n(A)$ is a θ-k-code. If $x' \in \mathrm{Sub}_k(u_1u_4)$ and $y' \in \mathrm{Sub}_k(y_1v_1)$ then either $y_1v_1 \in A$ which violates the condition $\mathrm{Sub}_k\theta(B^2) \cap \mathrm{Sub}_k(A) = \emptyset$ or $y_1v_1 = rw_1w_2...w_nt$ such that $rw_1 \in A, w_i \in B$.

(i) The case when $y' \in \mathrm{Sub}(rw_1)$ contradicts $\mathrm{Sub}_k(\theta(B^2)) \cap \mathrm{Sub}_k(A) = \emptyset$.

(ii) The case when $y' \in w_iw_{i+1}$ violates property 2 of Definition 2.4.1.

(iii) $y' \in w_n t$ where $w_n t \in A$ or $w_n t \in \mathrm{Sub}(B^2)$ which is again a contradiction.

Hence $\sigma^{n+1}(A)$ is θ-k-code and so is $L(\gamma)$. $\square$

We end this section with an observation about the conditions under which a splicing language is θ-subword-k-code.

Proposition 2.4.4 *Given k, let A be such that $\mathrm{Sub}_k(A) \cap \mathrm{Sub}_k(\theta(A)) = \emptyset$. Let $\mathcal{R}_B(A) \subseteq B^{(4)}$ be a set of rules with words from a set $B \subseteq \mathrm{Sub}_k(A)$ that satisfies $\theta(Suff_i(B)Pref_j(B)) \cap \mathrm{Sub}_k(A) = \emptyset$ and $\theta(Suff_i(B)Pref_j(B)) \cap Suff_r(B)Pref_s(B) = \emptyset$ for all $i + j = k$ and $r + s = k$. Then $L(\gamma)$ is θ-subword-k-code.*

Proof: Clearly A is θ-subword-k-code. Assume that $\sigma^n(A)$ is θ-subword-k-code. Suppose $x = x_1u_1u_4x_4$ for some $(u_1, u_2, u_3, u_4) \in \mathcal{R}_B(A)$ such that $x = suv\theta(u)p$ where $|v| \geq 1, |u| = k$ (i.e.) $suv\theta(u)p = x_1u_1u_4x_4$. Then the two possibilities are that

(i) $u \in Sub_k(A)$ and $\theta(u) \in Suff_i(B)Pref_j(B)$.

(ii) $u \in Suff_i(B)Pref_j(B)$ and $\theta(u) \in Suff_r(B)Pref_s(B)$.

(iii) $u \in Sub_k(A)$ and $\theta(u) \in Sub_k(A)$.

All cases contradicts the hypothesis. Therefore $\sigma^{n+1}(A)$ is θ-subword-k-code and hence $L(\gamma)$ is θ-subword-k-code. $\square$

Example 2.4.4 $X = \{(ab)^n : 1 \leq n \leq 10\}$ is θ-comma-free and θ-subword-2-code for $\Sigma = \{a, b, c, d\}$ with the morphic involution θ defined with $a \mapsto c$ and $b \mapsto d$ and $\theta(X) = \{(cd)^n : 1 \leq n \leq 10\}$ and $U = \{a, b\}$. It is easy to verify that $L(\gamma) \subseteq (\Sigma \setminus \{c, d\})^+$ and hence $L(\gamma)$ is θ-subword-2-code, θ-infix and θ-comma-free.

Chapter 3

Algebraic Characterizations of Involution Codes

In this chapter we concentrate on the algebraic characterizations of these involution codes mainly through their syntactic monoid. Algebraic characterizations of many classes of codes are already known [34, 39, 40, 41]. The characterizations of the syntactic monoid of an infix code was shown in [41]. Similar results were shown for the maximal prefix codes in [40]. The characterizations of the syntactic monoid of a comma-free code X and X^+ are discussed in [39]. The syntactic characterization of strictly locally testable languages are discussed in [34].

In our case we concentrate on θ-infix and θ-comma-free codes. In section 3.1 we observe that the image of a syntactic monoid of a language X under a morphic or antimorphic involution θ is equal to the syntactic monoid of the image of the language X under θ. Then we characterize the syntactic monoid of some involution codes. In section 3.2 we construct involution codes that are strictly locally testable. In section 3.3 through a new operation on monoids called "glue along a set" we characterize the syntactic monoid of X, X being θ-infix or θ-comma-free codes and the syntactic monoid of X^+.

3.1 θ-image of the Syntactic Monoid

Given an involution code with one of the "good" property we look at the properties of these codes such that every non zero element of their syntactic monoid also satisfy the same "good" property. We also observe that the image of a syntactic monoid of a language under an involution map is equal to the syntactic monoid of the image of the language. In other words, the syntactic monoid of a language X is isomorphic to the syntactic monoid of $\theta(X)$ for any involution θ. Before we proceed with the results, note that for any code X and an injective morphism θ on X, $\theta(X)$ is also a code (see [5]).

Lemma 3.1.1 *Let $\theta : \Sigma \to \Sigma$ be (anti) morphic involution such that θ can be extended to a (anti) morphism on Σ^*. Let $X \subseteq \Sigma^*$ be a code then $\theta(X) = X'$ is also a code for Σ^*. Define $\bar{\theta} : M(X) \to M(X')$ by $\bar{\theta}([x]) = [\theta(x)]$, where $M(X)$ is the syntactic monoid of X. Then $\bar{\theta}$ is well defined, and $\bar{\theta}$ is a (anti) morphism when θ is a (anti) morphism.*

Proof:

(i) We show that $\bar{\theta}$ is well defined.
To show if $x \equiv y(P_X)$ then $\theta(x) \equiv \theta(y)(P_{X'})$, i.e, $u\theta(x)v \in \theta(X)$ if and only if $u\theta(y)v \in \theta(X)$. Note that $u\theta(x)v \in \theta(X)$ if and only if $\theta(u\theta(x)v) \in X$ since θ is an involution. Hence $\theta(u)x\theta(v) \in X$ when θ is a morphism, if and only if $\theta(u)y\theta(v) \in X$ since $x \equiv y(P_X)$, which implies $u\theta(y)v \in \theta(X)$. Hence $\bar{\theta}$ is well defined.

(ii) $\bar{\theta}$ is a (anti)morphism when θ is a (anti)morphism.
To show that $\bar{\theta}([x][y]) = \bar{\theta}([x])\bar{\theta}([y])$. Note that $\bar{\theta}([x][y]) = \bar{\theta}([xy]) = [\theta(xy)] = [\theta(x)\theta(y)]$ since θ is a morphism. Hence $\bar{\theta}([x][y]) = [\theta(x)][\theta(y)] = \bar{\theta}([x])\bar{\theta}([y])$. Hence $\bar{\theta}$ is a morphism.□

Lemma 3.1.2 *Let θ, $\bar{\theta}$, X and X' be as defined in lemma 3.1.1. $[x] \in M(X)$ if and only if $\bar{\theta}([x]) \in M(X')$. Moreover $\bar{\theta}$ is an isomorphism.*

Proof:

(i) Let $[x] \in M(X)$. To show that $\bar{\theta}([x]) \in M(X')$ (i.e) to show the following.

(a) For $u, v \in \bar{\theta}([x])$, $aub \in X'$ if and only if $avb \in X'$ for some $x, y \in \Sigma^*$.
If $u, v \in \bar{\theta}([x])$ then there are $u_1, v_1 \in [x]$ such that $u = \theta(u_1), v = \theta(v_1)$. Let $a, b \in \Sigma^*$ and $a_1, b_1 \in \Sigma^*$ such that $aub \in X'$ implies $a\theta(u_1)b \in X'$ and $a = \theta(a_1), b = \theta(b_1) \Rightarrow \theta(a_1)\theta(u_1)\theta(b_1) \in X'$. When θ is morphic $\theta(a_1u_1b_1) \in X' \Rightarrow a_1u_1b_1 \in X$ since $[x] \in M(X)$. Hence $\theta(a_1v_1b_1) = \theta(a_1)\theta(v_1)\theta(b_1) \in \theta(X) = X' \Rightarrow avb \in X'$. Similar method works when θ is antimorphic.

(b) If $u \equiv v(P_{X'})$ with $u \in \bar{\theta}([x])$ then $v \in \bar{\theta}([x])$.

If $u \equiv v(P_{X'})$ with $u \in \bar{\theta}([x])$ then there exists $u_1 \in [x]$ such that $u = \theta(u_1)$. Let $a, b \in \Sigma^*$ such that $aub \in X'$ and $avb \in X'$ where $a = \theta(a_1)$ and $b = \theta(b_1)$ for some $a_1, b_1 \in \Sigma^*$. $aub = \theta(a_1)\theta(u_1)\theta(b_1)$. For θ antimorphic $aub = \theta(b_1u_1a_1) \in X'$ which implies $b_1u_1a_1 \in X$. Since $avb \in X'$ and $a = \theta(a_1), b = \theta(b_1)$ there exists some $r \in X$ such that $avb = \theta(r) = \theta(a_1)\theta(v_1)\theta(b_1)$ for some $v_1 \in \Sigma^*$. Hence $avb = \theta(b_1v_1a_1)$ and $b_1v_1a_1 \in X$. Since $u \in \bar{\theta}([x])$ and $[x] \in M(X)$, $u_1, v_1 \in [x]$ which implies $\theta(v_1) \in \bar{\theta}([x])$ and hence $v \in \bar{\theta}([x])$. Similar method works when θ is morphic.

Converse can be proved in a similar way. $\Box$.

We have the following observations.

Lemma 3.1.3 *Let $X \subseteq \Sigma^*$.*

(i) *X is θ-subword-k-code if and only if for every $[x] \in M(X) \setminus W(X)$, $[x]$ is θ-subword-k-code.*

(ii) *$Sub(X)$ is θ-comma-free if and only if for every $[x], [y] \in M(X) \setminus W(X)$, $[x], [x] \cup [y]$ are θ-comma-free.*

(iii) *Let X be θ-infix, or θ-comma-free or θ-subword-k-code or θ-k-code for some $k > 0$. In all cases $W(X) \neq \emptyset$.*

Proof:

(i) Note that for all $y \in [x]$ such that $[x] \in M(X) \setminus W(X)$, $y \in Sub(X)$. Hence $[x] \in M(X) \setminus W(X)$ is θ-subword-k-code if and only if X is θ-subword-k-code.

(ii) Obvious.

(iii) Let X be θ-infix. Then there exists $y \in \Sigma^*$ such that $\theta(y) \in Sub(X)$ and $y \notin X$. Hence $y \in W(X)$. Similarly we can show for all the other involution codes $W(X) \neq \emptyset$. $\Box$

Lemma 3.1.4 *Let $X \subseteq \Sigma^*$. If X is an involution code with one of the "good" properties defined in Definitions 1.2.24, 1.2.25, 1.2.26, then there exists atleast one $[x] \in M(X)$ such that $[x]$ is an involution code with the same "good" property.*

Proof: Note that for a subset of involution code with one of the "good" property is also an involution code with the same "good" property. Since for any X, X is a union of P_X classes, there exists an equivalence class $[x]$ in P_X which is a subset of X. Since X is an involution code, $[x]$ is also an involution with the "good" property. □

The following propositions observe the properties of the non zero elements of the syntactic monoid of involution codes with one of the "good" property.

Proposition 3.1.1 *X is θ-k-code if and only if for every $[x], [y] \in M(X) \setminus W(X)$, $[x]$ and $[x] \cup [y]$ is a θ-k-code.*

Proof: ($\Rightarrow$) obvious. Conversely, assume that X is not θ-k-code. Then there exists $x, y \in Sub_k(X)$ such that $\theta(x) = y$. But $x \in [x]$ and $y \in [y]$ for some $[x], [y] \in M(X) \setminus W(X)$ which implies $[x] \cup [y]$ is not a θ-k-code which is a contradiction. □

Proposition 3.1.2 *Let $X \subseteq \Sigma^*$ and let $Z_x = \{z \in \mathrm{Sub}(X) : z = a\theta(x)b, a, b \in \Sigma^*\}$ for all $x \in \mathrm{Sub}(X)$.*

X is θ-infix and for all $x \in \mathrm{Sub}(X)$ for all $z \in Z_x$, there exists $u, v \in \Sigma^*$ such that $uxv \in X$ and $uzv \notin X$ if and only if for every $[x] \in M(X)$, $[x]$ is θ-infix and $[s_1] \cup [s_2]$ is θ-infix for all $[s_1], [s_2] \in S$ where S is the image of X in $M(X)$ i.e. $\eta^{-1}(S) = X$.

Proof: ($\Rightarrow$). Suppose there exists $[x] \in M(X) \setminus W(X)$ such that $[x]$ is not θ-infix, then there exists $x, y \in [x]$ such that $x = a\theta(y)b$ for some $a, b \in \Sigma^*$ which implies $x \in Z_y$. Since $x, y \in [x]$, for all $u, v \in \Sigma^*$, if $uyv \in L$ then $uxv \in L$ which is a contradiction since $x \in Z_y$. Hence $[x]$ is θ-infix. Since X is θ-infix and $s_1 \subseteq X$, for all $[s_1] \in S$, and $[s_1] \cup [s_2]$ is θ-infix for all $[s_1], [s_2] \in S$.

Conversely, If X is a P_X class then X is θ-infix. Assume X is not θ-infix then there exists $x, y \in X$ such that $x = a\theta(y)b$. If

(i) $x, y \in [x]$ for some $[x] \in M(X)$. Done.

(ii) $x \in [s_1]$ and $y \in [s_2]$ for some $[s_1], [s_2] \in S$ contradicts that $[s_1] \cup [s_2]$ is θ-infix.

Suppose for all $x \in \mathrm{Sub}(X)$ and for all $z \in Z_x$ and for all $u, v \in \Sigma^*$ such that $uxv \in X \Rightarrow uzv \in X$ then $uxv, ua\theta(x)bv \in X$. Hence $x \in [s_1], a\theta(x)b \in [s_2]$ for some $[s_1], [s_2] \in S$ which contradicts that $[s_1] \cup [s_2]$ is θ-infix. □

3.2 Constructing Strictly Locally Testable Involution Codes

In this section we provide with methods to generate involution codes that are strictly locally testable. Locally testable languages are the best known subclass of star free languages. Regular languages can be described with the help of a strictly locally testable languages. In [37] the authors showed that there are no star free languages, where the commutative closure is regular, but not star-free. Locally testable languages are used in the study of DNA and informational macromolecules in biology.

In [14] Head showed an interesting relationship between splicing languages generated by splicing systems and regular languages. He showed the equivalence relation between a certain type of splicing languages and a subclass of regular languages called strictly locally testable languages (i.e.) a certain type of amino acid sequences can be expressed by a locally testable language.

In [27] a linear time algorithm that learns the deterministic finite state automaton of a given strictly locally testable language was presented. Based on the theoretical result in [14] and the algorithm in [27], the authors described several experimental results (see [27]) to identify the protein α-chain region in amino acid sequences for hemoglobin.

Hence it is of interest to look for involution codes that are strictly locally testable. In this section we construct involution codes with one of the "good" property such that they are strictly locally testable and use the syntactic monoid characterizations of these languages to characterize the involution codes. We start the section by recalling the definition of local and strictly locally testable languages and a characterization of local languages.

Definition 3.2.1 *Let $X \subseteq \Sigma^*$*

(i) *X is said to be <u>local</u> if there is a finite set of words H such that $X = \Sigma^* \setminus \Sigma^* H \Sigma^*$. The set H is the set of forbidden words.*

(ii) *An automaton $\mathcal{A}$ is <u>local</u> if there is a positive integer n such that every word $w \in \Sigma^*$ with length $|w| \geq n$ is a constant function in the transition monoid (See [5, 42]) of $\mathcal{A}$. The integer n is called the <u>order</u> of $\mathcal{A}$.*

(iii) *X is said to be <u>strictly locally testable</u> if there are finite sets $P, S, H \subseteq \Sigma^*$ such*

that $X = (P\Sigma^* \cap \Sigma^* S) \setminus \Sigma^* H \Sigma^*$. *The maximal length of words in* $P \cup S \cup H$ *is the* <u>order</u> *of* X.

(iv) $w \in \Sigma^*$ *is a* <u>constant</u> *for a language* X *if, for all* $x_1, x_2, x_3, x_4 \in \Sigma^*$, *if* $x_1 w x_2 \in X$ *and* $x_3 w x_4 \in X$ *then* $x_1 w x_4 \in X$.

Proposition 3.2.1 *Let* X *be a regular language and* $\mathcal{A}$ *be the minimal deterministic finite automaton that recognize* X. *The following are equivalent.*

(i) X *is strictly locally testable of order* k.

(ii) $\mathcal{A}$ *is local of order* k.

(iii) *All words of* Σ^* *of length* $\geq k$ *are constants for* X

For details on the proof of Proposition 3.2.1 refer [31, 42]. Note that if X is local then it is strictly locally testable.

Example 3.2.2 The language $X = \{a^n b^n c^n : n \geq 1\}$ with alphabet set $\Sigma = \{a,b,c, d,e,f\}$ is not regular and hence not local and hence not strictly locally testable. For morphic involution θ with $\theta(a) = b$ and $\theta(c) = c$, we have $\theta(X) = \{b^n a^n c^n : n \geq 1\}$. It is easy to check that $\Sigma^+ \theta(X) \Sigma^+ \cap X^2 = \emptyset$. Hence X is θ-infix and θ-comma-free. For an antimorphic or morphic involution θ with $\theta(a) = d$, $\theta(b) = e$ and $\theta(c) = f$, X is θ-k-code and θ-subword-k-code for some $k > 0$. Hence involution codes in general are not strictly locally testable.

Given finite code words with certain properties, we would to like construct languages that are strictly locally testable with the same "good" properties.

Proposition 3.2.2 *Let* $X \subseteq \Sigma^+$ *be a finite language. Let* $A = \Sigma^* X \cap X \Sigma^* \setminus \Sigma^* \theta(X) \Sigma^*$. *Then* A *is strictly locally testable. If* X *is* θ*-infix* (*comma-free*) *then* A *is also* θ*-infix* (*comma-free*).

Proof: Let X be θ-infix. Suppose A is not θ-infix, then there exists $x, y \in A$ such that $x = a\theta(y)b$ for $a, b \in \Sigma^*$. Hence there exists a word $u \in \theta(X)$ such that u is a subword of $x \in A$ which is a contradiction. Therefore A is θ-infix.

Let X be θ-comma-free. Suppose A is not θ-comma-free, then there exists $x, y, z \in A$ such that $xy = a\theta(z)b$ for some $a, b \in \Sigma^+$. Let $x = x_1 c_1 x_2$,$y = x_3 c_2 x_4$, $z = x_5 c_3 x_6$

for some $x_i \in X$ and $c_i \in \Sigma^*$. Then $xy = a\theta(z)b$ implies that $\theta(x_5)$ is a subword of x or y, or $\theta(x_5)$ is a subword of x_2x_3. Both cases contradict that X is θ-comma-free. Hence A is θ-comma-free. □

Proposition 3.2.3 *Let $X \subseteq \Sigma^*$ such that for all $x \in X$, $|x| > k$ for some $k > 0$ and X is finite and X^* is θ-subword-k-m-code. Let $Y = \bigcup_{i=2k+1}^{2k+m} \mathrm{Sub}_i(X^2)$ and let $M = \bigcup_{i=2k+1}^{2k+m} \Sigma^i \setminus Y$. Then $A = \Sigma^* \setminus \Sigma^* M \Sigma^*$ is θ-subword-k-m-code and strictly locally testable.*

Proof: Suppose not, then there exists $x \in A$ such that $x = auy\theta(u)b$ with $|uy\theta(u)| = 2k + i$ for $1 \leq i \leq m$. Clearly $uy\theta(u) \notin \mathrm{Sub}(X^n)$ since X^* is θ-subword-k-m-code, which implies $uy\theta(u) \in M \Rightarrow x \notin A$ which is a contradiction to our assumption. Hence A is θ-subword-k-m-code. □

The following proposition gives us a method to construct strictly locally testable language that is also θ-k-code for some k. We have much stronger conditions than Proposition 3.2.3 due to stronger requirements.

Proposition 3.2.4 *Let k be a finite number and Σ be a finite alphabet set and we partition Σ^k into three disjoint subsets such that $\Sigma^k = P \cup Q \cup R$ such that $\theta(P) = Q$ and for all $x \in R$, $\theta(x) = x$. Let $H = P \cup R$ and let $A = \Sigma^* \setminus \Sigma^* H \Sigma^*$. Then A is θ-k-code.*

Proof: Suppose A is not θ-k-code, then there exists $x, y \in \mathrm{Sub}_k(A)$ such that $x = \theta(y)$. $x, y \notin P$ and $x, y \notin R$ since $P, R \subseteq H$. If $x \in Q$ then $\theta(x) = y \in H$ which is a contradiction. Hence A is a θ-k-code. □

We use the following well known (see [34, 35]) characterization of strictly locally testable languages to characterize the involution codes constructed above.

Proposition 3.2.5 *(See [35]) Let X be a languages such that $\tau_c = \{[x] : x$ is a constant for $X\}$. X is strictly locally testable iff $[1] = \{1\}$ and the set of idempotents $E = \{e : e^2 = e, e \neq 1, e \neq 0\}$ is a non-empty subset of the ideal τ_c.*

We give more methods in section 4.2 to construct involution codes that are not necessarily strictly locally testable and have also calculated the informational entropy of such codes.

3.3 Syntactic Monoid of Involution Codes

In theory of codes, two types of syntactic monoids are usually considered, the syntactic monoid of the codes itself and the syntactic monoid of the Kleene* of the code. In this section we concentrate on the characterizations of syntactic monoids of θ-infix and θ-comma-free codes and the syntactic monoid of the + closure of the words over these codes. We characterize the syntactic monoid of an involution code through a new operation on monoids called "glue along a set". The main theorem in this section consists of the characterization of syntactic monoid of X and X^+ when X is θ-infix or θ-comma-free. Necessary and sufficient conditions on a monoid to be the syntactic monoid of a θ-infix or a θ-comma-free codes are also discussed.

Proposition 3.3.1 *Let $X \subseteq \Sigma^*$ be θ-infix. Let $[x] \in M(X)$ be such that $[x] \subseteq X$ with $\bar{\theta}([x]) = [x]$ then the P_X class of 1 is trivial where P_X is the syntactic congruence.*

Proof: Suppose the P_X class of 1 is not trivial, then there exists u such that $C_X(u) = C_X(1)$. Let $v \in [x]$ such that $\theta(v) \in [x]$. Then $v, uv \in [x]$ and $\theta(v), u\theta(v) \in [x]$. Hence $v, u\theta(v) \in [x]$ which implies $u = 1$ since $[x] \subseteq X$ and X is θ-infix. □

Proposition 3.3.2 *Let X be a code and let $L = X^+$. Then P_L class of 1 is trivial i.e. $[1]=\{1\}$.*

Proof: Suppose not, there exists $u \in \Sigma^*$ such that $C_L(u) = C_L(1)$. Note that $u \notin L$ since $1 \notin L$ which implies $u \in \mathrm{PSub}(L) \setminus L$ since $C_L(u) \neq \emptyset$. Note that L is a union of P_L classes and choose $[x]$ such that $[x]$ is a P_L class and $[x] \subseteq L$ and there exists $v \in [x]$ such that $v \in X$. Then $C_L(uv) = C_L(v)$, $C_L(vu) = C_L(v)$ and $C_L(vuv) = C_L(v^2)$ which implies both $uv, vu \in [x]$ and $vuv \in L$ since $v^2 \in L$. Hence vuv has two distinct factorizations which implies X is not a code which is a contradiction. □

Corollary 3.3.1 *If X is a code then $M(X^+) \setminus \{1\}$ is a sub semigroup of $M(X^+)$.*

The converse of the above proposition need not be true. For example, take $\Sigma = \{a, b, c\}$ and let $X = \{a, b, ab\}$ and let $L = X^+$. Clearly L is not a code. Note that $\mathrm{Sub}(L) = L$. Note $C_L(1) = C_L(u)$ for all $u \in \mathrm{Sub}(L)$, since $\mathrm{Sub}(L) = L$ and $1 \notin L$ and hence the P_{X^+} class of 1 is trivial. Also note that for any code X with

$L = X^*$, P_L class of 1 may not be trivial. For example for $X = \{ab, aa\}$, it is easy to check that $C_L(aa) = C_L(1) = \{(1,1), (1,ab), (a,b), (ab,1), (1,aa), (aa,1), (a,a)...\}$ and hence $aa \equiv 1(P_L)$.

Proposition 3.3.3 *Let $X \subseteq \Sigma^*$ and $\eta : \Sigma^* \to \Sigma^*/P_X$ then $\eta(X) = S \subseteq M(X)$ is a disjunctive subset of $M(X)$.*

Proof: Suppose there exists $[x], [y] \in M(X)$ such that $C_S([x]) = C_S([y]) = \{([a],[b]) : [a][x][b] = [axb] \in S\}$ which implies $C_X(x_1) = C_X(y_1) = \{(a,b) : axb \in X\}$ for all $x_1 \in [x]$ and $y_1 \in [y]$. Hence $x_1, y_1 \in [x]$ and $x_1, y_1 \in [y]$ and therefore $[x] = [y]$.□

We concentrate on necessary and sufficient conditions for a monoid to be syntactic monoid of a θ-infix code. The characterization of syntactic monoid of an infix code was done in [41]. We have our results here as a corollary to a theorem stated and proved in [41]. For more details on the proof we refer the reader to [41]. The following theorem was proved in [41].

Theorem 3.3.2 *A monoid M with identity 1 is isomorphic to the syntactic monoid of an infix code X if and only if*

(i) *$M \setminus \{1\}$ is a subsemigroup of M.*

(ii) *M has a zero.*

(iii) *M has a disjunctive element $[s]$ such that $[s] \neq 0, 1$ and $[s] = [x][s][y]$ for some $x, y \in \Sigma^*$ implies $[x] = [y] = 1$.*

Note that for $L = X \cup \theta(X)$, $\theta(L) = \theta(X) \cup X$ and hence $M(L) = M(\theta(L))$. Therefore by Lemma 3.1.2 , $\bar{\theta} : M(L) \to M(L)$.

Corollary 3.3.3 *Let $X \subseteq \Sigma^*$ be such that $L = X \cup \theta(X)$ is θ-infix. Then M is the syntactic monoid of L if and only if*

(i) *$M \setminus \{1\}$ is a subsemigroup of M*

(ii) *M has a zero.*

(iii) *M has a disjunctive element $[s]$ such that $[s] \neq 0, 1$ and if $[x][s][y] = [s]$ or $[x]\bar{\theta}([s])[y] = [s]$ for $[x], [y] \in M$ then $[x] = [y] = 1$.*

Proof: Note that L is θ-infix if and only if L is an infix code. Hence (i),(ii) and part of (iii) are satisfied by Theorem 2 of [41]. It remains to show that if $[x]\bar{\theta}([s])[y] = [s]$ for a disjunctive element $[s] \neq 0, 1$ of the monoid M, then $[x] = [y] = 1$. Take $[s] \in M$ such that $\eta^{-1}([s]) = L$. Suppose there exists $[x], [y] \in M$ such that $[x][\theta(s)][y] = [s]$ then $x\theta(a)y = b$ for some $a, b \in [s]$ (i.e.) $a, b \in L$ which contradicts that L is θ-infix. Since L is infix, converse is true by Theorem 2 of [41]. □

Note that by Proposition 2.2.1, X^+ is θ-infix if and only if X^+ is θ-comma-free. Here we discuss the properties of the syntactic monoid of $X^+ \cup \theta(X^+)$ where X^+ is θ-comma-free.

Lemma 3.3.4 *Let X be a code such that X^+ is strictly θ-infix. Then the P_L class of 1 is trivial where $L = X^+ \cup \theta(X^+)$.*

Proof: Suppose there exists $u \in \Sigma^*$ such that $C_L(u) = C_L(1)$. Note that $u \notin L$ since $1 \notin L$. Hence $u \in \mathrm{PSub}(L) \setminus L$. Choose $[x] \in M$ such that $[x] \subseteq L$ and there exists $y \in [x]$ such that $y \in X$. Hence $C_L(yu) = C_L(x) = C_L(uy)$ which implies $yu, uy, yuy \in L$ and $yuy \notin \theta(X^+)$. If $yuy \in \theta(X^+)$ then either $y \in \theta(X^+)$ or $y \in \mathrm{PSub}(\theta(X^+))$ both contradict that X^+ is strictly θ-infix. Hence $yuy \in X^+$ with $yu, uy \in X^+$ which implies that yuy has two distinct factorizations in X which is a contradiction that X is a code. □

Note that X^+ is strictly θ-infix does necessarily imply that $X^+ \cup \theta(X^+)$ is a code. For example let $X = \{aa, aab, ab\}$ with $\theta(X) = \{bb, bba, ba\}$ for an antimorphic θ. X is θ-comma-free since $\Sigma^+\{bb, bba, ba\}\Sigma^+ \cap X^2 = \emptyset$. Hence by Proposition 2.2.1 X^+ is strictly θ-infix. But $X^+ \cup \theta(X^+)$ is not a code since $aabbba = (aa)(bb)(ba) = (aab)(bba) \in X\theta(X)$.

Corollary 3.3.5 *Let X be a code such that X^+ is strictly θ-infix. Let M be the syntactic monoid of $L = X^+ \cup \theta(X^+)$. Then $M \setminus \{1\}$ is a subsemigroup of M.*

Lemma 3.3.6 *Let X be a code such that X^+ is strictly θ-infix. Let M be the syntactic monoid of $L = X^+ \cup \theta(X^+)$. M has a zero.*

Proof: Let $x \in X$ and $y \in \theta(X)$. Since X^+ is strictly θ-infix, $x \notin \mathrm{Sub}(\theta(X))$ and $y \notin \mathrm{Sub}(X)$. Hence $xy \notin \mathrm{Sub}(L)$ which implies $xy \in W(L)$. □

Consider two monoids M_1 and M_2 with zeros. Let $N \subseteq M_1$ and $N' \subseteq M_2$ such that for an morphic or antimorphic bijection h, $h : N \mapsto N'$. Identify every element $n \in N$ with $n' \in N'$ such that $h(n) = n'$.

Definition 3.3.7 (Glue Along a set)
Define Glue along N *of monoids* M_1 *and* M_2 *to be the monoid* $M = M_1 \cup_N M_2$ *where* $M = (M_1 \setminus N) \cup M_2$ *and the operation*

$$xy = \begin{cases} 0 & : \ x \in M_1 \setminus N, y \in M_2 \setminus N' \\ 0 & : \ x \in M_2 \setminus N', y \in M_1 \setminus N \\ xy & : \ x, y \in M_2 \\ h(x)y & : \ x \in M_1 \setminus N, y \in N' \\ xh(y) & : \ x \in N', y \in M_1 \setminus N \\ h(xy) & : \ x, y \in M_1 \setminus N, xy \in N \\ xy & : \ x, y \in M_1 \setminus N, xy \in M_1 \setminus N \end{cases}$$

We exhibit the operation of gluing of two monoids with the following example of syntactic monoids of a set X and $\theta(X)$.

Example 3.3.8 Let $X = \{aa, aba\}$ and for an antimorphic θ which maps $a \to b$ and $b \to a$, $\theta(X) = \{bb, bab\}$.
Let $\bar{\theta} : M(X) \mapsto M(\theta(X))$ as in Lemma 3.1.1. We can check that $M_1 = M(X) = \{\bar{0}, \bar{1}, \bar{a}, \bar{b}, \bar{ab}, \bar{ba}, \bar{aa}\}$ and $M_2 = M(\theta(X)) = \{\hat{0}, \hat{1}, \hat{b}, \hat{a}, \hat{ab}, \hat{ba}, \hat{bb}\}$. Choose
$N = \{\bar{0}, \bar{1}, \bar{a}, \bar{b}, \bar{ab}, \bar{ba}\}$, then $N' = \{\hat{0}, \hat{1}, \hat{b}, \hat{a}, \hat{ab}, \hat{ba}\}$ and for all $\bar{x} \in N$, $\bar{x}$ is identified with $\hat{y} \in N'$ such that $\hat{y} = \bar{\theta}(\bar{x})$ with the following operation
$x.\hat{0} = \mathbf{0}$ for all $x \in M_1 \setminus N$ and for all $x \in M_2$
$\bar{aa}.\hat{1} = \mathbf{aa}$
$\hat{1}.y = y$ for all $y \in M_2 \setminus \{\hat{0}\}$
$\bar{aa}.x = \bar{\theta}(\bar{aa}).x = \hat{bb}.x = \mathbf{0}$ for all $x \in M_2 \setminus \{\hat{1}\}$
$\hat{a}.\hat{b} = \mathbf{ab}$ and $\hat{b}.\hat{a} = \mathbf{ba}$.
Hence $M_1 \cup_N M_2 = \{$ **0**, **1** , **a** , **b**, **ab**, **ba**, **aa**, **bb** $\}$.

Example 3.3.9 Let $X = \{a^*b\}$ and for an morphic θ which maps $a \to b$ and $b \to a$, $\theta(X) = \{ba^*\}$. Let $\bar{\theta} : M_1 \mapsto M_2$ as in lemma 3.1.1. We can check that $M_1 =$

$M(X) = \{\bar{0}, \bar{1}, \bar{a}, \bar{ab}\}$ and $M_2 = M(\theta(X)) = \{\hat{0}, \hat{1}, \hat{b}, \hat{ba}\}$. Choose $N = \{\bar{0}, \bar{1}, \bar{a}\}$, then $N' = \{\hat{0}, \hat{1}, \hat{b}\}$ and for all $\bar{x} \in N$, $\bar{x}$ is identified with $\hat{y} \in N'$ such that $\hat{y} = \bar{\theta}(\hat{x})$ with the operation

$\bar{ab}.\hat{0} = \mathbf{0}$

$\bar{ab}.\hat{1} = \mathbf{ba}$

$\bar{ab}.x = \mathbf{0}$ for all $x \in M_2 \setminus \{\hat{1}\}$

$\hat{b}.\hat{1} = \mathbf{b}$

$\hat{ba}.\hat{1} = \mathbf{ba}$

and $\bar{ab}.\hat{1} = \bar{\theta}(\bar{ab}).\hat{1} = \mathbf{ba}$.

Hence $M_1 \cup_N M_2 = \{\ \mathbf{0}, \mathbf{1}, \mathbf{b}, \mathbf{ba}, \mathbf{ab}\ \}$.

In the following theorems we give necessary and sufficient conditions for a monoid that is obtained by the "gluing along a set" of the syntactic monoids of X and $\theta(X)$ when X is θ-infix and θ-comma-free.

Let M_1 be the syntactic monoid of X and M_2 be the syntactic monoid of $\theta(X)$. Let $N \subseteq M_1$ such that $M_1 \setminus N = \{[x] : x \in X\}$ and let $N' \subseteq M_2$ such that $M_2 \setminus N' = \{[x] : x \in \theta(X)\}$. Then $\bar{\theta} : M_1 \to M_2$ identifies by Lemma 3.1.2 $[x] \in N$ with $[\theta(x)] \in N'$ for all $[x] \in N$. Let $M = (M_1 \setminus N) \cup_N M_2$.

Theorem 3.3.10 *Let $X \subseteq \Sigma^+$ and M be as above. Then X is strictly θ-infix if and only if there are disjunctive sets $S, S' \in M$ such that $I \cap S = \emptyset$ where I is the ideal $I = MS'M$.*

Proof: Let X be strictly θ-infix and $S = \{[x] : x \in X\}$ and $S' = \{[y] : y \in \theta(X)\}$. It follows from Proposition 3.3.3 that S and S' are disjunctive in M. Suppose $[a][y][b] = [x]$ for some $[a], [b] \in M$ and $[y] \in S'$ and $[x] \in S$. Then there are $x_1 \in [x]$ $y_1 \in [y]$, $a_1 \in [a]$ and $b_1 \in [b]$ such that $a_1 y_1 b_1 = x_1$ with $y_1 \in \theta(X)$ and $x_1 \in X$ which is a contradiction that X is strictly θ-infix.

Let $S \subseteq M$ with $[x] \notin S$ for $x \in X \cup \theta(X)$. Then either $C_S(0) = C_S([y]) = \emptyset$ for $y \in X \cup \theta(X)$, $0 \in S$ but $[y] \neq 0$ or $C_S(0) = C_S([z]) = \{(1,1)\}$ for $0, [z] \in S$ but $[z] \neq 0$. Hence S is not disjunctive. Conversely, suppose X is not strictly θ-infix, then there are $x, y \in X$ such that $a\theta(y)b = x$ for some $a, b \in \Sigma^*$. Then $[a][\theta(y)][b] = [x]$ in M with $[\theta(y)] \in S'$ and $[x] \in S$ which is a contradiction. $\square$

Note by Proposition 2.2.1 X^+ is strictly θ-infix if and only if X^+ is strictly θ-comma-free. Hence we investigate the properties of the monoid obtained by "gluing" $M(X^+)$ and $M(\theta(X^+))$ when X is strictly θ-comma-free.

Let M_1 be the syntactic monoid of X^+ and M_2 be the syntactic monoid of $\theta(X^+)$. Let $M = M_1 \cup_N M_2$ as defined above where $N = M_1 \setminus \{[x] \in M_1 : x \in \mathrm{Sub}(X^+) \setminus X^+\}$.

Theorem 3.3.11 *X^+ is strictly θ-comma-free if and only if there are disjunctive sets $S, S' \in M$ such that $I \cap S = \emptyset$ where I is the ideal $I = MS'M$.*

The proof of this theorem is similar to Theorem 3.3.10.

Note that Theorem 3.3.11 does not hold if M_1 and M_2 are the syntactic monoids of X and $\theta(X)$ respectively instead of X^+ and $\theta(X^+)$. For example let $X = \{baa, aba\}$ and $\theta(X) = \{bba, bab\}$. The syntactic monoids of X and $\theta(X)$ are respectively, $M_1 = \{\bar{0}, \bar{1}, \bar{a}, \bar{b}, \bar{ba}, \bar{ab}, \bar{aa}, \bar{baa}\}$ and $M_2 = \{\hat{0}, \hat{1}, \hat{a}, \hat{b}, \hat{ba}, \hat{ab}, \hat{bb}, \hat{bba}\}$. Choose $N = \{\bar{0}, \bar{1}, \bar{a}, \bar{b}, \bar{ba}, \bar{ab}, \bar{aa}\}$ and hence $N' = \{\hat{0}, \hat{1}, \hat{a}, \hat{b}, \hat{ba}, \hat{ab}, \hat{bb}\}$. Then the glue of M_1 and M_2 along N is given by $M = \{\mathbf{0}, \mathbf{1}, \mathbf{a}, \mathbf{b}, \mathbf{ba}, \mathbf{ab}, \mathbf{aa}, \mathbf{baa}, \mathbf{bba}\}$. Note that the condition $MS'M \cap S = \emptyset$ is satisfied for $S = \{\mathbf{baa}\}$ and $S' = \{\mathbf{bba}\}$, but X is not strictly θ-comma-free.

Chapter 4

Constructing Coded Languages

In the previous chapters we have discussed several properties of the involution codes. In this chapter we provide with methods to generate such involution codes with one or combinations of "good" properties. Section 4.1 gives an algorithm that can generate θ-infix and θ-comma-free codes not necessarily of same length. Section 4.2 provides certain methods to construct the involution codes X and also calculate the information rate of X^*. We designed certain sequences using the methods provided in Section 4.2 and tested them in a laboratory for cross hybridization. The results of the experiments are given in Section 4.3.

4.1 Algorithms

Many authors have addressed the problem of DNA strand design and have proposed various solutions. Most of the approaches have the following properties in common: (i) all DNA strands are of the same length (ii) a DNA sequence should not hybridize with a complement of another sequence, (iii) no two strands should hybridize with each other. We give an algorithm that tests whether a given set of certain DNA sequences that are not necessarily of the same length, prevents potential intermolecular and intramolecular hybridization. In particular we give an algorithm to test whether a given code is θ-infix and/or θ-comma-free codes. Algorithms and programs that test whether a given code is an involution code with one or all of the "good" property are given in [24, 26].

We want the set of DNA sequences that are designed to be codes. This condition is not obvious to verify for a finite set of words in general. It can be accomplished using the Flower automaton as described in [5].

Let X be a finite subset of Σ^+. We define a special automaton for X with $\mathcal{A}(X) = (Q, E, I, T)$ containing a set of states Q, the labeling (transition) $E \subseteq Q \times \Sigma \times Q$, a

set of initial states I and a set of terminal states T such that $I = \{\star\}$ and $T = \{\star'\}$. The set of states is defined to be $Q = \{\star, \star'\} \cup_{w \in X} \{q_i^w | w = a_1...a_n, i = 2, ..., n\}$ and the transitions in $\mathcal{A}(X)$ are:

$E = \{(\star, a_1, q_2^w) | w = a_1...a_n \in X\} \cup$

$\{(q_n^w, a_n, \star') | w = a_1...a_n \in X\} \cup$

$\{(q_i^w, a_i, q_{i+1}^w) | w = a_1...a_n \in X, i = 2, ..., n-1\}$

We call $\mathcal{A}(X)$ a semi Flower automaton for X. If $\star = \star'$, we say that $\mathcal{A}(X)$ is the Flower automaton for X.

With each automaton we associate a labeling function $\lambda : E \to \Sigma$ such that $\lambda(q, a, q') = a$. The labeling is extended to paths in the usual way. We say that an automaton is unambiguous if and only if for all $p \in I$, $q \in T$ and $w \in \Sigma^*$ there is at most one path from p to q with label w. We use the following proposition proved in [5].

Proposition 4.1.1 *A finite set of words X is a code if and only if the Flower automaton $\mathcal{A}(X)$ is unambiguous.*

We use the following defined in [5].

Definition 4.1.1 (Square Automaton) *Let $\mathcal{A}(X) = (Q, E, I, T)$ be an automaton over Σ. The square of $\mathcal{A}$, is the automaton $\mathcal{S}(\mathcal{A}) = (Q', E', I', T')$ where $Q' = Q \times Q$, $I' = I \times I$, $T' = T \times T$ and the transitions are $E' \subseteq Q' \times \Sigma \times Q'$ such that*
$E' = \{((p, q), a, (p', q')) | (p, a, p'), (q, a, q') \in E, a \in \Sigma\}$.

The following proposition follows from the definitions.

Proposition 4.1.2 *An automaton $\mathcal{A} = (Q, E, I, T)$ is unambiguous if and only if there is no path in $\mathcal{S}(\mathcal{A})$ from a state (p, p) to a state (q, q) visiting a state (r, s) with $r \neq s$.*

Using the square of a Flower automaton and Proposition 4.1.2 we can determine whether a finite set X is a code. Based on this a program was constructed (see [19]) that uses the construction of the Flower automaton and its square to determine whether a generated set (or a set of words inputed by the user) is a code.

The program is tested for θ-infix only and the algorithm that is used for comparing $\theta(u)$ with v for all $u, v \in X$ is brute force. It is a rather simple check whether one word is a subword of another that uses standard string library routines of C++ that the use of the brute force algorithm in this case is justified. In the same way we check whether X is strictly θ. However, the algorithm for checking whether it is θ-comma-free is based on the following construction.

Consider the variation of a semi-Flower automaton $\mathcal{A}^2(X)$ for X^2 consisting of the set of states $Q = \{\star, \star', *''\} \cup_{w \in X} \{p_i^w, q_i^w | w = a_1...a_n, i = 2, ..., n\}$. The transitions (edges) of $\mathcal{A}^{\in}(X)$ are as follows:

$$E = \{(\star, a_1, q_2^w), (\star', a_1, p_2^w) | w = a_1...a_n \in X\} \cup$$
$$\{(q_n^w, a_n, \star'), (p_n^w, a_n, *'') | w = a_1...a_n \in X\} \cup$$
$$\{(q_i^w, a_i, q_{i+1}^w), (p_i^w, a_i, p_{i+1}^w | w = a_1...a_n \in X, i = 2, ..., n-1\}.$$

Clearly $\mathcal{A}^2(X)$ recognizes X^2. Since we are interested in the subwords of X^2, we set that all states in $\mathcal{A}^2(X)$ are initial and terminal.

Now consider the semi-Flower automaton $\mathcal{A}(\theta(X))$ in a similar way as we constructed the square automaton. Consider the product of $(A)^2(X) \times (\theta(X))$. The initial states of the product automaton are ordered pairs of initial states, and the terminal states of the product are ordered pairs of terminal states. Denote this automaton with $\mathcal{B}$. The algorithm for θ-comma-free the relies on the following proposition.

Proposition 4.1.3 *The set of code words X is θ-comma-free if and only if $\mathcal{B}$ recognizes the empty set.*

It is easily seen from the definitions that for any morphic or antimorphic involution θ a set of code words that is θ-comma-free is also θ-infix. Hence, it is sufficient for our algorithm to check for θ-comma-free.

4.2 Methods for Constructing Involution codes

With the constructions in this section we show several ways to generate involution codes with "good" properties. Many authors have realized that in the design of DNA strands it is helpful to consider three out of the four bases. This was the case with several successful experiments [4, 8, 32]. It turns out that this, or a variation of this technique, can be generalized in such a way that codes with some of the desired properties can be easily constructed. In this section, we concentrate on providing methods to generate "good" code words X such that X^+ has the same property. For each code X, the entropy of X^+ is computed. The entropy measures the information capacity of the codes, i.e., the efficiency of these codes when used to represent information.

Suppose that we have a source alphabet with p symbols each with probability $s_1, s_2, ...s_p$. If $s_1 = 1$, then there is no information since we know what the message must be. If all the probabilities are different then for a symbol with low probability we get more information than for a symbol with high probability. Hence information is somewhat inversely related to the probability of occurrence. Entropy is the average information over the whole alphabet of symbols.

The standard definition of entropy of a code $X \subseteq \Sigma^+$ uses probability distribution over the symbols of the alphabet of X (see [5]). However, for a p-symbol alphabet, the maximal entropy is obtained when each symbol appears with the same probability $\frac{1}{p}$. In this case the entropy essentially counts the average number of words of a given length as subwords of the code words [25]. From the coding theorem, it follows that $\{0,1\}^+$ can be encoded by X^+ with $\Sigma \mapsto \{0,1\}$ if the entropy of X^+ is at least $\log 2$ ([1], see also Theorem 5.2.5 in [31]). The codes for θ-comma-free, strictly θ-comma-free, and θ-k-codes designed in this section have entropy larger than $\log 2$ when the alphabet has $p = 4$ symbols. Hence, such DNA codes can be used for encoding bit-strings.

We start with the entropy definition as defined in [31].

Definition 4.2.1 *Let X be a code. The entropy of X^+ is defined by*

$$\hbar(X) = lim_{n\to\infty} \frac{1}{n} \log |\mathrm{Sub}_n(X^+)|.$$

If G is a deterministic automaton or an automaton with a delay (see [31]) that recognizes X^+ and A_G is the adjacency matrix of G, then by Perron-Frobenius theory A_G has a maximal positive eigen value $\bar{\mu}$ and the entropy of X^+ is $\log \bar{\mu}$ (see Chapter 4 of [31]). We use this fact in the following computations of the entropies of the designed codes. In [16], Proposition 16, authors designed a set of DNA code words that is strictly θ-comma-free. The following propositions shows that, in a similar way, we can construct codes with additional "good" properties.

In what follows we assume that Σ is a finite alphabet with $|\Sigma| \geq 3$ and $\theta : \Sigma \to \Sigma$ is an involution which is not identity. We denote with p the number of symbols in Σ. We also use the fact that X is (strictly) θ-comma free iff X^+ is (strictly) θ-comma-free (Proposition 2.2.1).

Proposition 4.2.1 *Let $a, b \in \Sigma$ be such that for all $c \in \Sigma \setminus \{a, b\}$, $\theta(c) \notin \{a, b\}$. Let $X = \bigcup_{i=1}^{\infty} a^m(\Sigma \setminus \{a, b\})^i b^m$ for a fixed integer $m \geq 1$. Then X and X^+ are θ-comma-free. The entropy of X^+ is such that $\log(p-2) < \hbar(X^+) < \log(p-1)$.*

Proof. Let $x_1, x_2, y \in X$ such that $x_1 x_2 = s\theta(y)t$ for some $s, t \in \Sigma^+$ and $x_1 = a^m p b^m$, $x_2 = a^m q b^m$ and $y = a^m r b^m$, for $p, q, r \in (\Sigma \setminus \{a, b\})^+$. Since θ is an involution, if $\theta(a) \neq a, b$, then there is $c \in \Sigma \setminus \{a, b\}$ such that $\theta(c) = a$, which is excluded by assumption. Hence, either $\theta(a) = a$ or $\theta(a) = b$. When θ is morphic $\theta(y) = \theta(a^m)\theta(r)\theta(b^m)$, and when θ is antimorphic $\theta(y) = \theta(b^m)\theta(r)\theta(a^m)$. So, $\theta(y) = a^m\theta(r)b^m$ or $\theta(y) = b^m\theta(r)a^m$. Since $x_1x_2 = a^m p b^m a^m q b^m = s b^m \theta(r) a^m t$ or $x_1x_2 = a^m p b^m a^m q b^m = s a^m \theta(r) b^m t$ the only possibilities for r are $\theta(r) = p$ or $\theta(r) = q$. In the first case $s = 1$ and in the second case $t = 1$ which is a contradiction with the definition of θ-comma-free. Hence X is θ-comma-free.

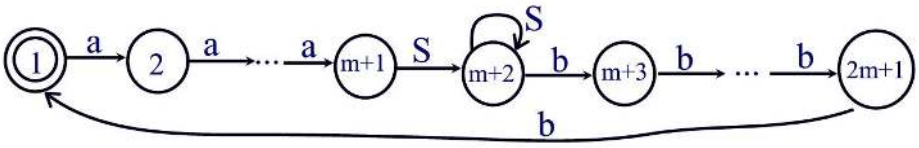

Figure 4.1: Finite state automaton $\mathcal{A}$ that recognizes X^+ where $S = \Sigma \setminus \{a, b\}$.

Let $\mathcal{A} = (V, E, \lambda)$ be the automaton that recognizes X^+ where $V = \{1, ..., 2m+1\}$ is the set of vertices, $E \subseteq V \times \Sigma \times V$ and $\lambda : E \to \Sigma$ (with $(i, s, j) \mapsto s$) is the labeling

function defined in the following way:

$$\lambda(i,s,j) = \begin{cases} a & \text{for } 1 \le i \le m,\ j = i+1, \\ b & \text{for } m+2 \le i \le 2m,\ j = i+1, \text{ and } i = 2m+1, j = 1, \\ s & \text{for } i = m+1, m+2,\ j = m+2,\ s \in \Sigma \setminus \{a,b\} \end{cases}$$

Then the adjacency matrix for $\mathcal{A}$ is a $(2m+1) \times (2m+1)$ matrix with ijth entry equal to the number of edges from vertex i to vertex j. Then the characteristic polynomial can be computed to be $\det(A - \mu I) = (-\mu)^{2m}(p-2-\mu) + (-1)^{2m}(p-2)$. The eigen values are solutions of the equation $\mu^{2m}(p-2) - \mu^{2m+1} + p - 2 = 0$ which gives $p - 2 = \mu - \frac{\mu}{\mu^{2m}+1}$. Hence $0 < \frac{\mu}{\mu^{2m}+1} < 1$, i.e., $p - 2 < \mu < p - 1$.□

In the case of the DNA alphabet, $p = 4$ and for $m = 1$ the above characteristic equation becomes $\mu^3 - 2\mu^2 - 2 = 0$. The largest real value of μ is approximately 2.3593 which means that the entropy of X^+ is greater than $\log 2$.

Example 4.2.2 Consider the DNA alphabet Δ with $\theta = \rho\nu$. Let m=2 and choose A and T such that $X \subseteq \bigcup_{i=1}^{n} A^2\{G,C\}^i T^2$. Then X and so X^+ is θ-comma-free.

Proposition 4.2.2 *Choose distinct $a, b, c \in \Sigma$ such that $\theta(a) \neq b, c$, $\theta(a) \neq a$. Let $X = \bigcup_{i=1}^{\infty} a^m(\Sigma^{m-1}c)^i b^m$ for some $m \ge 2$. Then X, and so X^+ is strictly θ-comma-free. The entropy of X^+ is such that $\log(p^{\frac{m-1}{m}}) < \hbar(X^+) < \log((p^{m-1}+1)^{\frac{1}{m}})$.*

Proof. The proof that X is strictly θ-comma-free is not difficult and we just give a sketch. Suppose there are $x, x_1, x_2 \in X$ such that $s\theta(x)t = x_1x_2$ for some $s, t \in \Sigma^+$. Let $x = a^m s_1 c s_2 c .. s_k c b^m$ then $\theta(x)$ is either $\theta(a^m)\theta(s_1c...s_kc)\theta(b^m)$ or $\theta(b^m)\theta(s_1c...s_kc)\theta(a^m)$ which cannot be a proper subword of x_1x_2 for any $x_1, x_2 \in X$. Hence X is θ-comma-free.

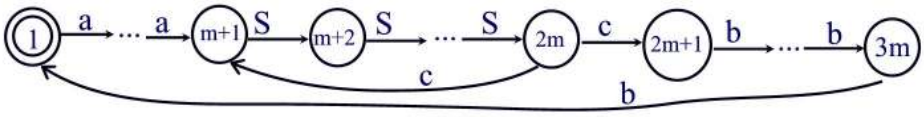

Figure 4.2: Finite state automaton $\mathcal{A}$ that recognizes X^+ where $S = \Sigma$.

Let $\mathcal{A} = (V, E, \lambda)$ be the automaton that recognizes X^+ where $V = \{1, ..., 3m\}$ is the set of vertices, $E \subseteq V \times \Sigma \times V$ is the set of edges and $\lambda : E \to \Sigma$ (with $(i,s,j) \mapsto s$) is the labeling function defined in the following way:

$$\lambda(i,s,j) = \begin{cases} a & \text{for } 1 \le i \le m,\ j = i+1, \\ b & \text{for } 2m \le i \le 3m-1,\ j = i+1, \text{ and } i = 3m, j = 1, \\ c & \text{for } i = 2m,\ j = m+1, \\ s & \text{for } m+1 \le i \le 2m-1,\ j = i+1,\ s \in \Sigma \end{cases}$$

Note that this automaton is not deterministic, but it has a delay 1, hence the entropy of X^+ can be obtained form its adjacency matrix. Let A be the adjacency matrix of this automaton. The characteristic equation for A is $-(\mu)^{3m} + (\mu)^{2m}p^{m-1} + p^{m-1} = 0$. This implies $p^{m-1} = \frac{\mu^{3m}}{\mu^{2m}+1} = \mu^m - \frac{\mu^m}{\mu^{2m}+1}$. Since p is an integer and $0 < \frac{\mu^m}{\mu^{2m}+1} < 1$, $p^{\frac{m-1}{m}} < \mu < p^{m-1}+1)^{\frac{1}{m}}$.□

For the DNA alphabet, $p = 4$, and for $m = 2$, the above characteristic equation becomes $\mu^6 - 4\mu^4 - 4 = 0$. Solving for μ, the largest real value of μ is 2.055278539. Hence the entropy of X^+ is greater than $\log 2$.

Example 4.2.3 Consider Δ and $\theta = \rho\nu$ and let $m = 2$, $a = A, c = C, b = G$. Then $X = \bigcup_{i=1}^{\infty} AA(\Delta C)^i GG$ and X^+ are strictly θ-comma-free.

With the following propositions we consider ways to generate θ-subword-k-code and θ-k-codes.

Proposition 4.2.3 *Let $a, b \in \Sigma$ be such that $\theta(a) = b$, and let $X = \bigcup_{i=1}^{\infty} a^{k-1}((\Sigma \setminus \{a,b\})^{k-2}b)^i$.*

Then X is θ-subword-k-code for $k \ge 3$. Moreover, when θ is morphic, X is a θ-k-code. The entropy of X^+ is such that $\log((p-2)^{\frac{k-2}{k-1}}) < \hbar(X^+) < \log(((p-2)^{k-2}+1)^{k-1})$.

Proof. Suppose that there exists $x \in X$ such that $x = rus\theta(u)t$ for some $r, t \in \Sigma^*$, $u, \theta(u) \in \Sigma^k$ and $s \in \Sigma^m$ for $m \ge 1$. Let $x = a^{k-1}s_1bs_2b...s_nb$ where $s_i \in (\Sigma \setminus a, b)^{k-2}$. Then the following are all possible cases for u:

(i) u is a subword of $a^{k-1}s_1$,

(ii) u is a subword of as_1b,

(iii) u is a subword of s_1bs_2,

(iv) u is a subword of bs_ib for some $i \leq n$.

In all these cases, since $\theta(a) = b$, $\theta(u)$ is not a subword of x. Hence X is θ-subowrd-k-code.

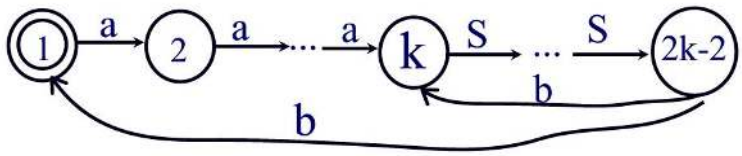

Figure 4.3: Finite state automaton $\mathcal{A}$ that recognizes X^+ where $S = \Sigma \setminus \{a, b\}$.

Let $\mathcal{A} = (V, E, \lambda)$ be the automaton that recognizes X^+ where $V = \{1, ..., 2k-2\}$ is the set of vertices, $E \subseteq V \times \Sigma \times V$ and $\lambda : E \to \Sigma$ (with $(i, s, j) \mapsto s$) is the labeling function defined in the following way:

$$\lambda(i,s,j) = \begin{cases} a & \text{for } 1 \leq i \leq k-1,\ j = i+1, \\ b & \text{for } i = 2k-2,\ j = k, \text{ and } i = 2k-2, j = 1, \\ s & \text{for } k \leq i \leq 2k-3,\ j = i+1,\ s \in \Sigma \setminus \{a, b\} \end{cases}$$

This automaton is with delay 1.

Let A be the adjacency matrix of this automaton. The characteristic equation for A is $[(-\mu)^{2k-2} - (p-2)^{k-2}\mu^{k-1} - (p-2)^{k-2} = 0$. So $(p-2)^{k-2} = \mu^{k-1} - \frac{\mu^{k-1}}{\mu^{k-1}+1}$. Since $0 < \frac{\mu^{k-1}}{\mu^{k-1}+1} < 1$, $(p-2)^{\frac{k-2}{k-1}} < \mu < ((p-2)^{k-2}+1)^{k-1}$.□

For the DNA alphabet, $p = 4$, and for $k = 3$, the above characteristic equation becomes $\mu^4 - 2\mu^2 - 2 = 0$. Solving for μ, the largest real value is 1.6528 which is greater than the golden mean (1.618), but less than 2. The asymptotic value for μ is 2 when k approaches infinity.

Example 4.2.4 Consider Δ with $\theta = \rho\nu$ and choose $k = 3$. Then $X = \bigcup_{i=1}^{\infty} AA(\{G, C\}T)^i$ is θ-subword-3-code.

As other authors have observed, note that it is easy to get θ-k-code if one of the symbols in the alphabet is completely ignored in the construction of the code X.

Proposition 4.2.4 *Assume that $\theta(a) \neq a$ for all symbols $a \in \Sigma$. Let $b, c \in \Sigma$ such that $\theta(b) = c$ and let $X = \bigcup_{i=1}^{\infty} a^{k-1}((\Sigma \setminus \{c\})^{k-2}b)^i$ for $k \geq 3$. Then X and*

X^+ *are a* θ*-*k*-code. The entropy of* X^+ *is such that* $log((p-1)^{\frac{k-2}{k-1}}) < \hbar(X^+) < log(((p-1)^{k-2}+1)^{\frac{1}{k-1}})$.

Proof. The fact that X^+ is a θ-k-code is straight forward, since every subword of $x \in X$ of length k is either power of a or contains the symbol b.

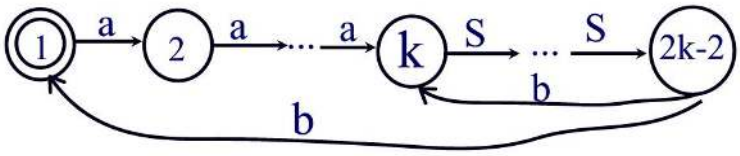

Figure 4.4: Finite state automaton $\mathcal{A}$ that recognizes X^+ where $S = \Sigma \setminus \{c\}$.

Let $\mathcal{A} = (V, E, \lambda)$ be an automaton that recognizes X^+ where $V = \{1, ..., 2k-2\}$ is the set of vertices, $E \subseteq V \times \Sigma \times V$ and $\lambda : E \to \Sigma$ (with $(i, s, j) \mapsto s$) is the labeling function defined in the following way:

$$\lambda(i,s,j) = \begin{cases} a, & \text{for } 1 \leq i \leq k-1,\ j = i+1 \\ b, & \text{for } i = 2k-2,\ j = k, \text{ and } i = 2k-2, j = 1 \\ s, & \text{for } k \leq i \leq 2k-3,\ j = i+1,\ s \in \Sigma \setminus \{c\} \end{cases}$$

This automaton is with delay 1.

Let A be the adjacency matrix of this automaton. Its characteristic equation is $\mu^{2k-2} - \mu^{k-1}(p-1)^{k-2} - (p-1)^{k-2} = 0$. This implies $(p-1)^{k-2} = \mu^{k-1} - \frac{\mu^{k-1}}{\mu^{k-1}+1}$. We are interested in the largest real value for μ. Since $\mu > 0$, we have $0 < \frac{\mu^{k-1}}{\mu^{k-1}+1} < 1$ which implies $(p-1)^{\frac{k-2}{k-1}} < \mu < ((p-1)^{k-2}+1)^{\frac{1}{k-1}}$. $\square$

For the DNA alphabet, $p = 4$ and for $k = 4$ the above estimate says that $\mu > 3^{\frac{2}{3}} > 2$. Hence the entropy of X^+ in this case is greater than $\log 2$.

Proposition 4.2.5 *Assume that* $\theta(a) \neq a$ *for all symbols* $a \in \Sigma$. *Let* $b, c \in \Sigma$ *such that* $\theta(b) = c$ *and let* $X = \bigcup_{i=1}^{\infty}((\Sigma \setminus \{c\})^{k-1}b)^i$ *for* $k \geq 3$. *Then* X *and* X^+ *are a* θ*-*k*-code. The entropy of* X^+ *is such that* $\hbar(X^+) = log((p-1)^{\frac{k-1}{k}})$.

The fact that X^+ is a θ-k-code is straight forward, since every subword of $x \in X$ of length k contains the symbol b.

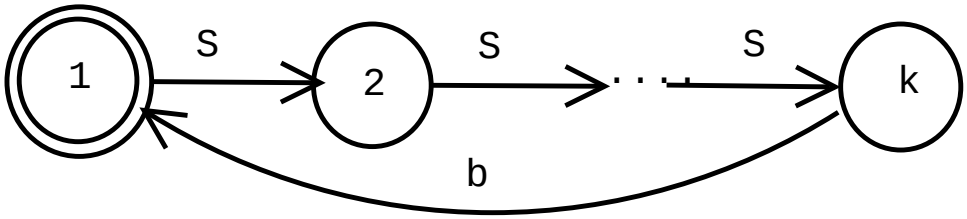

Figure 4.5: Finite state automaton $\mathcal{A}$ that recognizes X^+ where $S = \Sigma \setminus \{c\}$.

Let $\mathcal{A} = (V, E, \lambda)$ be an automaton that recognizes X^+ where $V = \{1, ..., k\}$ is the set of vertices, $E \subseteq V \times \Sigma \times V$ and $\lambda : E \to \Sigma$ (with $(i, s, j) \mapsto s$) is the labeling function defined in the following way:

$$\lambda(i, s, j) = \begin{cases} b, & \text{for } i = k, j = 1 \\ s, & \text{for } 1 \leq i \leq k-1,\ j = i+1,\ s \in \Sigma \setminus \{c\} \end{cases}$$

Let A be the adjacency matrix of this automaton. Its characteristic equation is $\mu^k - (p-1)^{k-1} = 0$. This implies that $\mu = (p-1)^{\frac{k-1}{k}}$. $\square$

For the DNA alphabet, $p = 4$ and for $k = 5$ the above estimate says that $\mu = 3^{\frac{4}{5}}$. Hence the entropy of X^+ in this case is greater than $\log 2$.

We would like to investigate what happens to the "goodness" of these codes under a morphic or antimorphic injective mapping. In other words, what should be the conditions on these codes so that they still satisfy the properties under such a mapping.

Proposition 4.2.6 *Let Σ_1 and Σ_2 be finite alphabet sets and let f be an injective morphism or antimorphism from Σ_1 to Σ_2^*. Let X be a code over Σ_1^*. Then $f(X)$ is a code over Σ_2^*. Let $\theta_1 : \Sigma_1^* \to \Sigma_1^*$ and $\theta_2 : \Sigma_2^* \to \Sigma_2^*$ be both morphisms or antimorphisms respectively such that $f(\theta_1(x)) = \theta_2(f(x))$.*

Let $P = Pref(\theta_2(f(X)))$ and $S = Suff(\theta_2(f(X)))$.

(i) *Let $(A^+P \cup SA^+) \cap f(\Sigma_1^+) = \emptyset$ and $A^+PA^+ \cap f(\Sigma_1) = \emptyset$ where $A = \Sigma_2^* \setminus f(\Sigma_1^*)$. If X is strictly θ_1-infix (comma-free) then $f(X)$ is strictly θ_2-infix (comma-free).*

(ii) *Let f be such that $|f(a)| = r$ for some fixed r and $\Sigma_2^+ \mathrm{Sub}_{(k-1)r}(f(X))\Sigma_2^+ \cap \mathrm{Sub}_{kr}(f(X)) = \emptyset$.*

(a) *If X is θ_1-subword-k-code then $f(X)$ is θ_2-subword-$(k+1)r$-code.*

(b) *If X is θ_1-k-code then $f(X)$ is θ_2-$(k+1)r$-code.*

Proof:

(i) Let X be θ_1-infix. Suppose $f(X)$ is not θ_2-infix. Then there exit $x, y \in f(X)$ such that $x = a\theta_2(y)b$ for some $a, b \in \Sigma_2^*$ and $x_1, y_1 \in X$ such that $x = f(x_1)$ and $y = f(y_1)$. Suppose $a, b \in f(\Sigma_1^*)$ then there exits $a_1, b_1 \in \Sigma_1^*$ such that $f(x_1) = f(a_1)f(\theta_2(y_1))f(b_1)$ which implies $x_1 = a_1\theta_1(y_1)b_1$. This is a contradiction with the second hypothesis in 1. Suppose $a \notin f(\Sigma_1^*)$. Then either there exists $y' \in P$ such that $ay' \in f(\Sigma_1^+)$ or there exists a b' such that $b = b'b''$ with $a\theta_2(y)b' \in f(\Sigma_1)$. This is a contradiction with the first hypothesis in 1. Hence $f(X)$ is θ_2-infix. Similar proof works when X is θ_1-comma-free.

(ii) Suppose $f(X)$ is not a θ_2-$(k+1)r$-code. Then there are $x, y \in Sub_{(k+1)r}(f(X))$ such that $x = \theta_2(y)$. First assume $x, y \in f(\Sigma_1^+)$ and there are $x_1, y_1 \in Sub_{(k+1)}(X)$ such that $f(x_1) = x$ and $f(y_1) = y$. Hence $f(x_1) = \theta_2(f(y_1))$ which implies $x_1 = \theta_1(y_1)$. This contradicts with X being a θ_1-k-code. Otherwise, if $x = cx'd$ such that $x' \in f(\Sigma_1^+)$ and $c, d \in A$, then $|x'| \leq kr$ and $x' = f(x_1)$ for some $x_1 \in Sub_k(X)$. Similarly $y = ey'g$ such that $e, g \in A$ and $y' = f(y_1)$ for some $y_1 \in \mathrm{Sub}_k(X)$. If $x' = \theta_2(y')$ then $f(x_1) = \theta_2(f(y_1))$ which implies $x_1 = \theta_1(y_1)$, a contradiction. Suppose $x' \neq \theta_2(y')$ then x' is a subword of $ey'g$ which contradicts $\Sigma_2^+\mathrm{Sub}_{(k-1)r}(f(X))\Sigma_2^+ \cap \mathrm{Sub}_{kr}(f(X)) = \emptyset$. Hence $f(X)$ is θ-$(k+1)r$ code. Similar proof works for θ_1-subword-k-code.

□

4.3 Experimental Results

Using the methods provided in Section 4.2, a set of 20 sequences of length 20 base pairs was designed and tested experimentally. Among these 10 were designed by methods from θ-5-codes (Proposition 4.2.5), five of them are θ-subword-3-codes (Proposition 4.2.3) and the remaining 5 were θ-comma free codes (Proposition 4.2.2).

Purified oligonucleotides were purchased from Integrated DNA Technologies. In addition to the 20 sequences, Watson -Crick complement of one of the sequences ($K1$) was ordered to act as a control (lane 18). The sequences $K1$-$K10$ are the θ-5-codes, $S1$-$S5$ and $F1$-$F5$ are respectively θ-subword-3-codes and θ-comma free codes. Non-cross-hybridization and secondary structures were measured by running the sequences on TAE polyacrylamide non-denaturing gels (15%) at $4°C$.

Table 1: Sequences

	Sequence		Sequence
K1	aatacatcacatttctaccc	S1	aagtgtctgtctctgtctgt
K2	actactacacacctcttacc	S2	aactgtctctgtctgtctct
K3	atcaccacccatcacactac	S3	aagtctgtgtctctctgtct
K4	ttaccatctctatacatctc	S4	aagtctctgtgtctctgtgt
K5	ctatctattcctctcacatc	S5	aactctgtctgtgtctctct
K6	tacacataactccactcatc	F1	aagcggaatctcggaaacgg
K7	tcccacatcccatactaatc	F2	aatcggaagcgcgcaaacgg
K8	ctaacatacctacacactac	F3	aaccggaatcacgcaatcgg
K9	acctctacacacttcacaac	F4	aaacggaaacacggaagcgg
K10	tatacatacctcaaccactc	F5	aatcggaatctcggaagcgg

The results of the experiment are shown in Figure 4.6.

Table 1: Sequences

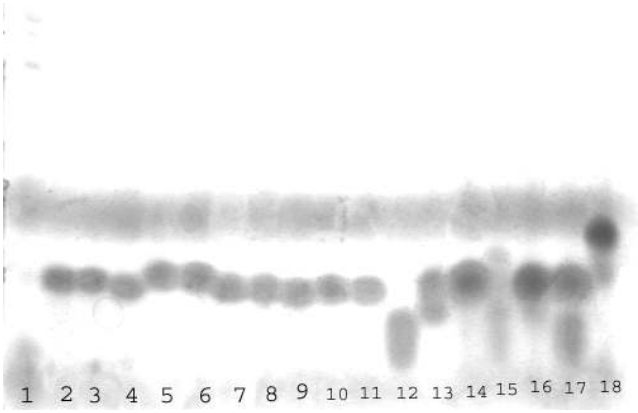

Figure 4.6: A 15% acrylamide non-denaturing gel of the tested DNA codes.

Lane	Content
2-11	K1-K10
12	All θ-comma-free codes
13	All θ-subword-3-codes
14	All θ-5-codes
15	All θ-5-codes with all θ-subword-3-codes
16	All θ-subword-3-codes with all θ-comma-free-codes
17	All θ-comma-free codes with all θ-5-codes
18	Double stranded molecule ($K1$ with $\overleftarrow{K1}$)

Observed results: No duplexes were detected in lane 14 which contains annealed solution of all θ-5-codes. Their speed on the gel coincides with one of the single stranded molecules in lanes 2-11. This shows that no cross-hybridization is observed for these strands. Also no secondary structures were detected in any lanes 2-11. All molecules are at the same level as 20bp mark of the ladder in lane 1. The ladder is not visible clearly due to the staining of the gel for a long time in order to make the DNA's visible. The double stranded molecule on lane 18 runs at a different level than the single stranded DNA molecules ($K1 - K10$) on lanes 2-11. However some cross hybridization was observed in θ-comma-free and θ-subword-3-codes (lanes 12,13,15,16,17).

Conclusion

We have investigated the theoretical properties of languages that consist of DNA based code words. In particular we concentrated on intermolecular and intramolecular cross-hybridizations that can occur as a result that a Watson-Crick complement of a (sub)word of a code word is also a (sub)word of a code word. These conditions are necessary for a design of code words but not sufficient. For example, the algorithms used in the programs developed by Seeman [44], Feldkamp [9] and Ruben [43], all check for uniqueness of k-length subsequences in the code words. Unfortunately none of the properties from Definitions 1.2.24, 1.2.25 and 1.2.26 of the involution codes defined in section 1.2 ensures uniqueness of k-length words. Such code words properties remain to be investigated. We have investigated the closure properties of these involution codes in Section 2.1. The observations in Section 2.2 provide a general way how from a small set of code words with desired property we can obtain, by concatenating the existing words, arbitrarily large sets of codewords with similar properties. We hope that general methods of designing such code words will simplify the search for "good" codes. Better characterizations of good code words that are closed under Kleene* operation may provide even faster ways of designing such code words. The most challenging questions of characterizing and designing good θ-k-codes remains to be developed.

In Section 2.3 we have discussed the properties of various levels of θ-comma-free subcodes of a given code not necessarily θ-comma-free. Such codes are useful when a DNA sequence has to be decoded over a code which is not θ-comma-free. Decoding a DNA sequence over a code that can be completely split into a sequence of θ-comma-free codes are desired. The properties of such codes are yet to be investigated. Section 2.4 investigates the necessary and sufficient conditions for preserving these "good" properties under splicing.

Given a family of sets of code words, we would like to construct an algorithm that takes as input a given set of code words and outputs yes or no depending on whether or not the given set of code words are involution codes with one of the "good" properties. Such decidability issues for θ-infix and θ-comma-free are discussed in [16]. Note that an algorithm to decide whether a given language is regular already exists. We would like to investigate the decidability issues for θ-(subword)-k-codes.

We have also discussed algebraic characterizations of these involution codes through their syntactic monoid. Section 3.2 gives methods to construct involution codes that are strictly locally testable. Necessary and sufficient conditions on a monoid obtained by "gluing along a set" of two monoids are discussed in Section 3.3. General methods that construct involution codes that are regular are given in Section 4.2. The information capacity of these codes designed by the methods from section 4.2 show that in most cases we obtain sets of code words that can encodes binary strings symbols $\mapsto$ symbols. In [2], the authors use binary strings with certain properties to generate DNA codes (assigning bases to bits). It remains to be investigated how powerful (in the sense of information capacity) can be the transition from binary strings to DNA code words. Our experimental results show when θ-k-codes are used, they provide a good base for developing new code words. However, the other properties taken isolated are not shown to provide sufficient distinction between coded DNA strands in laboratory experiment.

Our approach to the question of designing "good" DNA codes has been from the formal language theory aspect. Many issues that are involved in designing such codes have not been considered. These include (and are not limited to) free energy conditions, melting temperature as well as Hamming distance conditions. All these remain to be challenging problems, and a procedure that includes all or majority of these aspects will be desired in practice.

References

[1] R.L. Adler, D. Coppersmith and M. Hassner, Algorithms for sliding block codes - an application of symbolic dynamics to information theory, *IEEE Trans. Inform. Theory* **29** (1983): 5-22.

[2] M. Arita and S. Kobayashi, DNA Sequence Design Using Templates, *New Generation Comput.* **20**(3): 263–277 (2002). (Available as a sample paper at http://www.ohmsha.co.jp/ngc/index.htm.)

[3] E.B. Baum, DNA Sequences useful for computation, unpublished article, available at: http://www.neci.nj.nec.com/homepages/eric/seq.ps (1996).

[4] R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund and L. Adleman, Solution of a 20-variable 3-SAT problem on a DNA computer, *Science* **296** (2002): 499-502.

[5] J. Berstel and D. Perrin, Theory of codes *Academis Press, Inc. Orlando Florida* 1985.

[6] R. Deaton, J. Chen, H. Bi, M. Garzon, H. Rubin and D.F. Wood, A PCR-based protocol for in vitro selection of non-crosshybridizing oligonucleotides *DNA Computing: Proceedings of the 8th International Meeting on DNA Based Computers* (M. Hagiya, A. Ohuchi editors), Springer LNCS **2568** (2003): 196-204.

[7] R. Deaton et. al., A DNA based implementation of an evolutionary search for good encodings for DNA computation *Proc. IEEE Conference on Evolutionary Computation* ICEC-97 (1997): 267-271.

[8] D. Faulhammer, A. R. Cukras, R. J. Lipton and L. F.Landweber, Molecular Computation: RNA solutions to chess problems *Proceedings of the National Academy of Sciences, USA* **97** 4 (2000): 1385-1389.

[9] U. Feldkamp, S. Saghafi and H. Rauhe, DNASequenceGenerator - A program for the construction of DNA sequences *DNA Computing: Proceedings of the 7th International Meeting on DNA Based Computers* (N. Jonoska, N.C. Seeman editors), Springer LNCS **2340** (2002): 23-32.

[10] C. Ferreti and G. Mauri, Remarks on Relativisations and DNA Encodings *Aspects of Molecular Computing*, N.Jonoska, G.Paun, G.Rozenberg editors, Springer LNCS **2950** (2004): 132-138.

[11] M. Garzon, R. Deaton and D. Reanult, Virtual test tubes: a new methodology for computing *Proc. 7th. Int. Symposium on String Processing and Information retrieval, A Coruňa, Spain.* IEEE Computing Society Press (2000): 116-121.

[12] T. Head, Relativized Code Concepts and Multi-Tube DNA Dictionaries *Finite vs Infinite* , C.S. Calude and G. Paun editors, (2000): 175-186.

[13] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors *Bull. Math. Biology* **49** (1987): 737-759.

[14] T. Head, Splicing Representations of Strictly Locally Testable Languages, *Discrete Apllied Mathematics* **87 (1-3)** (1998): 139-147.

[15] T. Head, Gh. Paun and D. Pixton, Language theory and molecular genetics *Handbook of formal languages, Vol.II* (G. Rozenberg, A. Salomaa editors) Springer Verlag (1997): 295-358.

[16] S. Hussini, L. Kari and S. Konstantinidis, Coding properties of DNA languages *Theoretical Computer Science* **290** (2003): 1557-1579.

[17] N. Jonoska, Sofic systems with synchronizing representations, *Theoretical Computer Science*, **158** (1996): No.1-2, 81-115.

[18] N. Jonoska and K. Mahalingam, Languages od DNA based code words *Proceedings of the 9th International Meeting on DNA Based Computers*, J.Chen, J.Reif editors, Springer LNCS **2943**(2004): 61-73 .

[19] N. Jonoska, D. Kephart and K. Mahalingam, Generating DNA code words *Congressus Numernatium* **156** (2002): 99-110.

[20] N.Jonoska and K.Mahalingam, Methods for constructing coded DNA languages *Aspects of Molecular Computing*, N.Jonoska, G.Paun, G.Rozenberg editors, Springer LNCS **2950** (2004): 241-253.

[21] N.Jonoska , K.Mahalingam and J.Chen, Involution Codes: With Application to DNA Coded Languages, To appear in Natural Computing.

[22] L. Kari, S. Konstantinidis, E. Losseva and G. Wozniak, Sticky-free and overhang-free DNA languages, *Acta Informatica* **40** (2003): 119-157.

[23] L.Kari, S.Konstantinidis and P.Sosik, On properties of bond-free DNA languages, *Tech. report 609, University of Western Ontario, Department of Computer Science* (2003).

[24] L.Kari, S.Konstantinidis and P.Sosik, Bonde-free Languages: Formalizations, Maximality and Construction Methods, *Preliminary Proceedings of DNA 10 June 7-10* (2004):16-25.

[25] M.S. Keane, Ergodic theory an subshifts of finite type in *Ergodic theory, symbolic dynamics and hyperbolic spaces* (ed. T. edford, et.al.) Oxford Univ. Press, Oxford (1991): 35-70.

[26] D.Kephart and J.Lefevre, Codegen: The generation and testing of DNA code words, *Proceedings of IEEE Congress on Evolutionary Computation,* June (2004): 1865-1873.

[27] S.Kobayashi and T. Yokomori, Learning Local Languages and their Application to DNA sequence Analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol 20 No. 10, October (1998).

[28] S. Konstantinidis and A. O'Hearn, Error-Detecting Properties of Languages, *Theoretical Computer Science* , **276** (2002): 355-375.

[29] G. Lallement, Semigroups and Combinatorial Dynamics, *Wiley/Inter science, New York* (1995).

[30] Z. Li, Construct DNA code words using backtrack algorithm , preprint.

[31] D. Lind and B. Marcus, An introduction to Symbolic Dynamics and Coding, *Cambridge University Press, Inc. Cambridge United Kingdom* (1999).

[32] Q. Liu et al., DNA computing on surfaces, *Nature* **403** (2000): 175-179.

[33] Y.J. Liu and Z.B. Xu, Monoid Algorithms and Semigroup Properties Related to Dense Regular languages, *Proceedings of the International Conference of Algebra and its Apllication*, Bangkok (2002).

[34] A.D. Luca and A. Restivo, A Characterization of Strictly Locally Testable Languages and Its Application to Subsemigroups of a Free Semigroup, *Information And Control* **44** (1980): 300-319.

[35] N. Jonoska, A Conjugacy Invariant for Reducible Sofic Shifts and its Semigroup Characterizations , *Israel Journal of Mathematics* **106** (1998): 221-312.

[36] A. Marathe, A.E. Condon and R.M. Corn, On combinatorial word design, *Preliminary Preproceedings of the 5th International Meeting on DNA Based Computers, Boston* (1999): 75-88.

[37] A. Muscholl and H. Petersen, A Note on the commutative closure of the Star-free languages, *Information Processing Letters* **57(2)** (1996): 71-74.

[38] Gh. Paun, G. Rozenberg and A. Salomaa, DNA Computing, new computing paradigms, *Springer Verlag* 1998.

[39] M. Petrich and C.M. Reis, The syntactic monoid of the semigroup generated by a comma-free code, *Proceedings of the Royal Society of Edinburgh,* **125A** (1995): 165-179.

[40] M. Petrich, C.M. Reis and G. Thierrin, The syntactic monoid of the semigroup generated by a Maximal Prefix code, *Proceedings of the American Mathematical Society,* **124-3** March (1996): 655-663.

[41] M. Petrich, G. Thierrin, The syntactic monoid of an Infix code, *Proceedings of the American Mathematical Society* **109-4** (1990): 865-873.

[42] J.E. Pin, Varieties fo Formal Languages, *Plenum Press* (1986).

[43] A.J. Ruben, S.J. Freeland and L.F. Landweber, PUNCH: An evolutionary algorithm for optimizing bit set selection,

DNA Computing: Proceedings of the 7th International Meeting on DNA Based Computers (N. Jonoska, N.C. Seeman editors), Springer LNCS **2340** (2002): 150-160.

[44] N.C. Seeman, De Novo design of sequences for nucleic acid structural engineering, *J. of Biomolecular Structure & Dynamics* **8** (3) (1990): 573-581.

[45] A.N. Trahtman, A Polynomial time algorithm for local testability and its level, *International Journal of Algebra and Computer Science*, Vol **9-1** (1998): 31-39.

About the Author

Kalpana Mahalingam received a B.Sc., in Mathematics in 1998 from University of Madras, India and M.sc., in Mathematics in 2000 from Indian Institute of Technology. She completed the Ph.d. program in Mathematics at the University of South Florida in 2004.