Coding Properties of DNA Languages*

Salah Hussini[1], Lila Kari[2], and Stavros Konstantinidis[1]

[1] Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3, Canada
s.konstantinidis@stmarys.ca
[2] Department of Computer Science, University of Western Ontario London, Ontario, N6A 5B7 Canada
lila@csd.uwo.ca
http://www.csd.uwo.ca/∼lila

Abstract. The computation language of a DNA-based system consists of all the words (DNA strands) that can appear in any computation step of the system. In this work we define properties of languages which ensure that the words of such languages will not form undesirable bonds when used in DNA computations. We give several characterizations of the desired properties and provide methods for obtaining languages with such properties. The decidability of these properties is addressed as well. As an application we consider splicing systems whose computation language is free of certain undesirable bonds and is generated by nearly optimal comma-free codes.

1 Introduction

DNA (deoxyribonucleic acid) is found in every cellular organism as the storage medium for genetic information. It is composed of units called nucleotides, distinguished by the chemical group, or base, attached to them. The four bases, are *adenine, guanine, cytosine* and *thymine*, abbreviated as A, G, C, and T. (The names of the bases are also commonly used to refer to the nucleotides that contain them.) Single nucleotides are linked together end–to–end to form DNA strands. A short single-stranded polynucleotide chain, usually less than 30 nucleotides long, is called an *oligonucleotide*. The DNA sequence has a *polarity*: a sequence of DNA is distinct from its reverse. The two distinct ends of a DNA sequence are known under the name of the $5'$ end and the $3'$ end, respectively. Taken as pairs, the nucleotides A and T and the nucleotides C and G are said to be complementary. Two complementary single–stranded DNA sequences with opposite polarity are called *Watson/Crick complements* and will join together to form a double helix in a process called *base-pairing*, or *hybridization* [9].

In most DNA computations, there are three basic stages which have to be performed. The first is encoding the problem using single-stranded or double-stranded DNA. Then the actual computation is performed by employing a suc-

* All correspondence to Lila Kari. Research partially supported by Grants R2824A01 and R220259 of the Natural Sciences and Engineering Research Council of Canada.

N. Jonoska and N.C. Seeman (Eds.): DNA7, LNCS 2340, pp. 57–69, 2002.

cession of *bio-operations* [9]. Finally, the solutions, i.e. the result of the computation are sequenced and decoded.

In many proposed DNA-based algorithms, the initial DNA solution will contain some oligonucleotides which represent single "codewords", and some oligonucleotides which are strings of catenated codewords. This leads to two main types of possible undesirable hybridizations. First, it is undesirable for any DNA strand representing a codeword to form a hairpin structure, which can happen if either end of the strand binds to another section of that same strand. Second, it is undesirable for any DNA strand representing a codeword to bind to either another codeword strand or to the catenation of two codeword strands. If such undesirable hybridizations occur, they will in practice render the involved DNA strands useless for the subsequent computations.

To formalize these problems, let us introduce some notations. An alphabet X is a finite non-empty set of symbols (letters). We denote by $\Delta = \{A, C, G, T\}$ the DNA alphabet. A word u over the alphabet X is a sequence of letters and $|u|$ will denote the length of u, that is, the number of letters in it. By 1 we denote the empty word consisting of zero letters. By convention, a word u over the DNA alphabet Δ will denote the corresponding DNA strand in the $5'$ to $3'$ orientation. We denote by $\overleftarrow{u}$ the Watson-Crick complement of the sequence u, i.e., its reverse mirror image. For example, if $u = 5' - AAAAGG - 3'$ then $\overleftarrow{u} = 5' - CCTTTT - 3'$ will be the Watson-Crick complement that would base-pair to u. By X^* we denote the set of all words over X, and by X^+ the set of all non-empty words over X. A language (over X) is any subset of X^*. For a language K, we denote by K^* the set of words that are obtained by concatenating zero or more words of K; then, $1 \in K^*$. We write K^+ for the subset of all non-empty words of K^*.

Let now $K \subseteq \Delta^*$ be a set of DNA codewords. Several types of undesirable situations can occur with these DNA strands encoding initial data.

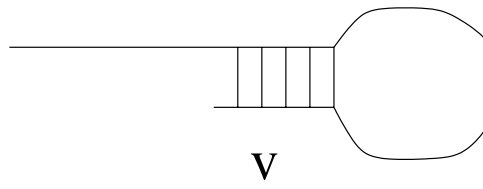

Fig. 1. Intramolecular hybridization I: $uv \in K$, $\overleftarrow{v}$ being a subword of u.

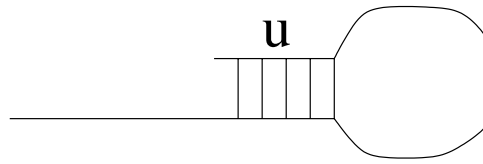

Fig. 2. Intramolecular hybridization II: $uv \in K$, $\overleftarrow{u}$ being a subword of v.

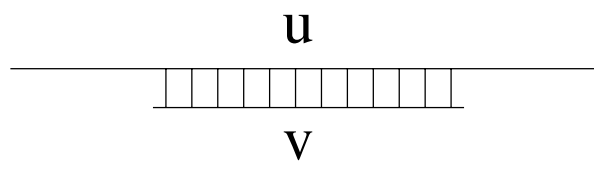

Fig. 3. Intermolecular hybridization I: $u, v \in K$, $\overleftarrow{v}$ being a subword of u.

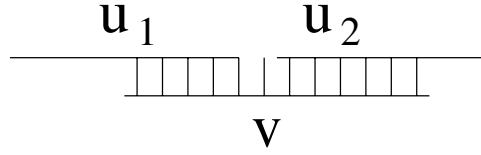

Fig. 4. Intermolecular hybridization II: $u_1, u_2, v \in K$, $\overleftarrow{v}$ being a subword of u_1u_2.

Several attempts have been made to address this issue by trying to find sets of codewords which are unlikely to form undesired bonds in this manner [2], [5], [4]. For example genetic algorithms have been developed which select for sets of DNA sequences that are less likely to form undesirable bonds [3], and combinatorial methods have been used to calculate bounds on the size of a set of uniform codewords (as a function of codeword length) which are less likely to mishybridize [11].

In this paper, we continue the approach in [10] where the notion of θ-compliance has been defined, where θ is an arbitrary morphic or antimorphic involution. In the particular case when θ is the Watson-Crick complement, a language is strictly θ-compliant (respectively strictly prefix θ-compliant, suffix θ-compliant) if situations like the ones depicted in Figure 3 (respectively Figures 1, 2) cannot occur. [10] studied properties of such θ-compliant languages, i.e. languages consisting of words with good coding properties. Here we define the notion of θ-freedom: if θ is the Watson-Crick complement, a language is θ-free iff situations like the ones depicted in Figure 4 cannot occur.

Section 3 studies θ-freedom and its relation to θ-compliance. It turns out that, under some conditions, languages that are θ-free are also θ-compliant.

Section 4 studies the question of whether or not, given a set of codewords, we can constructively decide if the set is θ-compliant or θ-free, i.e., if it has desirable coding properties.

Section 5 studies under what conditions, if we start with an initial set of "good" codewords, θ-compliance and θ-freedom are preserved by all intermediate strands that occur during computations.

With these studies, we attempt to gain a deeper insight into encoding information in DNA, which will hopefully assist solving this otherwise difficult problem.

2 Definitions and Notations

For a set S, we denote by $|S|$ the cardinality of S, that is, the number of elements in S. The set of non-negative integers is denoted by $\mathbf{N}$. Let X^* be the free monoid generated by the finite alphabet X. A mapping $\alpha : X^* \to X^*$ is called a *morphism (anti-morphism)* of X^* if $\alpha(uv) = \alpha(u)\alpha(v)$ (respectively $\alpha(uv) = \alpha(v)\alpha(u)$) for all $u, v \in X^*$. A bijective morphism (anti-morphism) is called an *isomorphism (anti-isomorphism)* of X^*.

An involution $\theta : S \to S$ of S is a mapping such that θ^2 equals the identity mapping, i.e., $\theta(\theta(x)) = x$ for all $x \in S$. It follows then that an involution θ is bijective and $\theta = \theta^{-1}$. The identity mapping is a trivial example of involution.

If Δ^* is the free monoid generated by the DNA-alphabet Δ then two involutions can be defined on Δ^*: the mirror involution μ which is an anti-morphism, and the complement involution γ which is a morphism ([10]).

Indeed, the mapping $\gamma : \Delta \to \Delta$ defined by $\gamma(A) = T, \gamma(T) = A, \gamma(C) = G, \gamma(G) = C$ can be extended in the usual way to a morphism of Δ^* that is also an involution of Δ^*. Obviously, γ is an involution as the complement of the complement of a sequence equals the sequence itself. This involution γ will be called the *complement involution* or simply the *c-involution of* Δ^*.

Let $\mu : \Delta^* \to \Delta^*$ be the mapping $\mu(u) = v$ defined by

$$u = a_1 a_2 \dots a_k, \quad v = a_k \dots a_2 a_1, a_i \in \Delta, 1 \leq i \leq k.$$

The word v is called the mirror image of u. Since μ^2 is the identity mapping, μ is an involution of Δ^* which will be called the *mirror involution* or simply the *m-involution* of Δ^*. The m-involution is an anti-morphism as $\mu(uv) = \mu(v)\mu(u)$ for all $u, v \in \Delta^*$.

It is easy to see that γ and μ commute, i.e. $\gamma\mu = \mu\gamma$, and hence $\gamma\mu$ which will denoted by τ is also an involution of Δ^*. Furthermore τ is an anti-morphism which corresponds to the notion of Watson-Crick complement of a DNA sequence and will therefore sometimes be called the DNA involution.

Instead of γ, μ, τ we sometimes use the alternative notation

$$\gamma(u) = \bar{u},\ \mu(u) = \tilde{u},\ \tau(u) = \gamma\mu(u) = \overleftarrow{\bar{u}}\,.$$

Following [10], if $\theta : X^* \to X^*$ is a morphic or antimorphic involution, a language $L \subseteq X^*$ is said to be *θ-compliant*, (prefix θ-compliant, suffix θ-compliant) iff, for any words $x, y, u \in X^*$,

$$u, x\theta(u)y \in L \ \ (\theta(u)y \in L, x\theta(u) \in L) \ \ \Rightarrow \ \ xy = 1.$$

If, in addition, $L \cap \theta(L) = \emptyset$ then the language L is called strictly θ-compliant (strictly prefix θ-compliant, strictly suffix θ-compliant, respectively).

Note that if the involution θ is the DNA involution, i.e., it represents the Watson-Crick complement, then a language L being strictly θ-compliant (strictly prefix θ-compliant, strictly suffix θ-compliant) amounts to the fact that situations of the type depicted in Figure 3 (respectively Figure 1, Figure 2) do not occur. Such languages with good coding properties have been studied in [10].

We close the section with some terminology on codes [8]. A code K is a subset of X^+ satisfying the property that, for every word w in K^+, there is a unique sequence $(v_1, v_2, \ldots, v_n)$ of words in K such that $w = v_1 v_2 \cdots v_n$. A bifix code K is a prefix and suffix code; that is, $K \cap KX^+ = K \cap X^+K = \emptyset$.

Every bifix code is indeed a code. An infix code, K, has the property that no word of K is properly contained in another word of K, that is, $K \cap (X^+KX^* \cup X^*KX^+) = \emptyset$. Every infix code is a bifix code. A comma-free code K is a language with the property $K^2 \cap X^+KX^+ = \emptyset$. Every comma-free code is an infix code.

3 Involution-Freedom and Involution-Compliance

In this section we define the notion of an involution-free language which formalizes the situation depicted in Figure 4. Moreover, this notion extends the concept of comma-free code, [13], in the same fashion that involution-compliance extends the concept of infix code.

We define the notion of a θ-free language (Definition 1) and establish relations between θ-freedom and θ-compliance (Lemma 1). We also establish some properties of θ-free languages. For example, Proposition 1 states that if a set of codewords K is strictly θ-free then the set consisting of arbitrary catenations of codewords from K will be strictly θ-free as well. Proposition 2 states that if a language is dense (it contains all possible sequences in X^* as subsequences of some of its strands), then it cannot be strictly θ-compliant or θ-free.

To aid the intuition, Propositions 3, 6, 7, 8 are proved for the particular case of the complement involution even though the results hold for the more general case of an arbitrary morphic involution.

After some examples of complement-free languages, Propositions 3, 6 bring together the notions of complement-compliance and complement-freedom. As it turns out, under some conditions, languages that avoid undesirable bindings of the type in Figures 1–3 avoid also situations of the type in Figure 4, and vice-versa.

Definitions 3 and 4 introduce the notions of complement-reflective respectively anti complement-reflective languages, which are generalizations of the notions of reflective and anti-reflective languages, [13], and relates them to complement-freedom.

This paves the way to methods of constructing complement-free languages as the one described in Proposition 7.

Proposition 8 gives a sufficient condition under which the catenation of two languages is complement-free.

Propositions 9, 10 address similar problems but this time for the anti-morphic involution case which is the type that the DNA Watson/Crick involution belongs to. Because of the additional "flipping" complication that such an involution entails, the results are slightly different.

Definition 1. *Let θ be an involution of X^*. A language K is called θ-free if $K^2 \cap X^+\theta(K)X^+ = \emptyset$. If, in addition, $K \cap \theta(K) = \emptyset$ then K is called strictly*

θ-free. For convenience, we agree to call K strictly θ-free when $K \setminus \{1\}$ is strictly θ-free.

As an example, consider the DNA involution τ of Δ^* and the language $ACC\Delta^2$. Then, $ACC\Delta^2 ACC\Delta^2 \cap \Delta^+(\Delta^2 GGT)\Delta^+ = \emptyset$. Hence, $ACC\Delta^2$ is τ-free and, in fact, strictly τ-free. The same language is strictly free for the complement involution as well. On the other hand, $A\Delta^2 A$ is strictly τ-compliant but not τ-free, as $ACTAATAA \in (A\Delta^2 A)^2 \cap \Delta^+\tau(A\Delta^2 A)\Delta^+$.

Lemma 1. *Let θ be a morphic or antimorphic involution and let K be a non-empty subset of X^+.*

(i) If K is θ-free then both K and $\theta(K)$ are θ-compliant.

(ii) If K is strictly θ-free then $K^2 \cap X^*\theta(K)X^* = \emptyset$.

Proposition 1. *Let θ be a morphic or antimorphic involution and let K be a non-empty subset of X^+. If K is strictly θ-free then also K^+ is strictly θ-free.*

It is shown next that the properties of strict θ-compliance and strict θ-freedom impose restrictions on the words of a language in terms of density and completeness. A language L is called dense, [1], if every word is a subword of some word of L; that is, $L \cap X^*wX^* \neq \emptyset$ for every $w \in X^+$. The language L is complete if L^* is dense, [1].

Proposition 2. *Let θ be a morphic or antimorphic involution and let K be a non-empty subset of X^+.*

(i) *If K is dense then K is not strictly θ-compliant.*

(ii) *If K is complete then K is not strictly θ-free.*

To make results more intuitive, we will prove the following four propositions for the particular case where θ is the complement involution. Note that the results can be generalized to refer to any arbitrary morphic involution.

There are many examples of complement-free languages. $L = AC^+A$ is an infinite complement-free language as

$$\{AC^{n_1}AAC^{n_2}A |\ n_1, n_2 > 0\} \cap \Delta^+ TG^+ T\Delta^+ = \emptyset.$$

For any $L_1 \subseteq AC^*$ and $L_2 \subseteq C^+A$, L_1L_2 is a complement-free language. Indeed, $L_1L_2 = AC^+A$ which is complement-free.

Every subset of a complement-free language is itself complement-free.

Let $L \subseteq \Delta^+$ be a finite language and let $m = \max\{|u| \mid u \in L\}$. Then $L' = AC^m LAC^m$ is complement-free. Indeed, assume $L'^2 \cap \Delta^+ \overline{L'} \Delta^+ \neq \emptyset$. Then, $AC^m w_1 AC^m AC^m w_2 AC^m = \alpha_1 TG^m \overline{w_3} TG^m \alpha_2$ for some $w_1, w_2, w_3 \in L$, $\alpha_1, \alpha_2 \in \Delta^+$. This implies TG^m is a subword of w_1 or w_2 – contradiction as $|w_1|, |w_2| \leq m$.

An example of a language L satisfying $L \cap \overline{L} = \emptyset$ is $\Delta^*\{A, C\}$.

For a language $L \subseteq X^+$ denote by

$$L_{pref} = \{x \in X^+ |\ xy \in L \text{ for some } y \in X^+\}$$

$$L_{suff} = \{y \in X^+ |\ xy \in L \text{ for some } x \in X^+\}$$

the language of non-empty proper prefixes of L and the language of non-empty proper suffixes of L respectively.

Proposition 3. *Let $L \subseteq X^+$, $L \neq \emptyset$, be a language. The following statements are equivalent:*

(1) *L is complement-free.*
(2) *L is c-compliant and $\overline{L} \cap L_{suff} L_{pref} = \emptyset$.*
(3) *L is c-compliant and $L^2 \cap L_{pref} \overline{L} L_{suff} = \emptyset$.*
(4) *$\overline{L}$ is complement-free.*

Definition 2. *Let $\theta : X^* \longrightarrow X^*$ be a morphic or antimorphic involution. For a non-empty language $L \subseteq X^+$ define*
$L_s = \{z \in X^+ |\ ux \in L, xz \in \theta(L)$ *for some* $u, x \in X^+\}$,
$L_{is} = \{x \in X^+ |\ wx \in \theta(L), xv \in L$ *for some* $w, v \in X^+\}$,
$L_p = \{x \in X^+ |\ xv \in \theta(L), vy \in L$ *for some* $y, v \in X^+\}$,
$L_{ip} = \{x \in X^+ |\ ux \in L, xv \in \theta(L)$ *for some* $u, v \in X^+\}$.

Note that in the particular case, where we talk about a morphic involution we have that $L_{is} = \theta(L_{ip})$. Indeed $x \in L_{is}$ means $wx \in \theta(L), xv \in L$ for some $w, v \in X^+$ which means that $\theta(w)\theta(x) \in L$, $\theta(x)\theta(v) \in \theta(L)$ which, in turn, means that $\theta(x) \in L_{ip}$, that is, $x \in \theta(L_{ip})$.

Proposition 4. *Let $L \subseteq X^+$ be a non-empty language and $\theta : X^* \longrightarrow X^*$ be an antimorphic involution. Then $\theta(L)_s = \theta(L_p)$ and $\theta(L)_p = \theta(L_s)$.*

Proposition 5. *Let $L \subseteq X^+$ be a non-empty language and $\theta : X^* \longrightarrow X^*$ be a morphic involution. Then $\theta(L)_s = \theta(L_s)$ and $\theta(L)_p = \theta(L_p)$.*

Take the particular case where θ is the complement involution.

Proposition 6. *Let $L \subseteq X^+$ be a non-empty language. Then L is complement-free if and only if L is complement-compliant and one of the following conditions hold:*

(1) $L_s \cap L_{is} = \emptyset$,
(2) $L_p \cap L_{ip} = \emptyset$,
(3) $L \cap \overline{L_p} L_p = \emptyset$,
(4) $L \cap L_s \overline{L_s} = \emptyset$.

Note that Proposition 6 holds also if we replace the complement involution with an arbitrary morphic involution.

Definition 3. *A language $L \subseteq X^+$, $L \neq \emptyset$ is called complement-reflective iff for all $x, y \in X^+$ we have that $xy \in L$ implies $\overline{yx} \in L$.*

Definition 4. *A language $L \subseteq X^+$, $L \neq \emptyset$ is called anti complement-reflective iff for all $x, y \in X^+$ we have that $xy \in L$ implies $\overline{yx} \notin L$.*

Note that every complement-free language is anti complement-reflective. Indeed, if for a complement-free language L we would have $xy \in L$ and $\overline{yx} \in L$ for some $x, y \in X^+$ then

$$(\overline{yx})(\overline{yx}) = \overline{yxyx} = \overline{y}(\overline{xy})\overline{x} \in X^+\overline{L}X^+ \cap L^2$$

which violates the condition of complement-freedom.

However, one can find anti complement-reflective languages that are not complement-free. Indeed, take $L = \{A^nT^{2n} |\ n \geq 1\}$ over the alphabet $\Delta = \{A, C, G, T\}$ with the usual complement function. Then L is anti complement-reflective but not complement-free. Indeed, $xy \in L$ implies $xy = A^nT^{2n}$ for some $n \geq 1$. For $\overline{yx}$ to belong to L, it must be the case that $x \in A^+$ and $y \in T^+$. Then $\overline{yx} = A^{2n}T^n \notin L$ therefore L is anti complement-reflective.

To show that L is not complement-free, take $u_1u_2 \in L$, that is, $u_1u_2 = A^nT^{2n}A^mT^{2m}$ for some $n, m \geq 1$. As it is possible to find some indices n, m, p such that $v \in L$, $v = A^pT^{2p}$ and $\overline{v} = T^pA^{2p}$ a subword of u_1u_2, it results that L is not complement-free.

In general the concatenation of two complement-free languages may not be complement- free. For example, take $L_1, L_2 \subseteq \Delta^+$,

$$L_1 = \{CACA, AC\}, \quad L_2 = \{TGTG, GT\}.$$

Both L_1 and L_2 are complement-free. On the other hand L_1L_2 contains the word $u = CACATGTG$ therefore we have that $uu = CACATGTGCACATGTG \in (L_1L_2)^2$ and the word $v = ACGT \in L_1L_2$ has the property that $\overline{v} = TGCA$ is a subword of uu. This means that L_1L_2 is not complement-free.

Nevertheless, for a given finite language $L \subseteq \Delta^+$ we can always find a word $u \in \Delta^+$ such that uL is complement-free.

Indeed, if $L = \{u_1, u_2, \ldots, u_n\}$ is a finite language and $m = \max\{|u| \mid u \in L\}$ then for the word $u = A^{m+1}C$, $m \geq 1$ the language uL is complement-free.

Take $\alpha_1\alpha_2 \in uLuL$. Then $\alpha_1\alpha_2 = A^{m+1}Cu_iA^{m+1}Cu_j$ for some $1 \leq i, j \leq n$. Consider $\alpha \in uL$. Then $\overline{\alpha} = T^{m+1}G\overline{u_k}$ for some $1 \leq k \leq n$. The only possibility for $\overline{\alpha}$ to be a subword of $\alpha_1\alpha_2$ would be that $T^{m+1}G$ is a subword of u_i or of u_j which is impossible because of the way m has been defined. Consequently, uL is complement-free.

Proposition 7. *For any finite language $L \subseteq \Delta^+$ there exists an infinite language $R \subseteq \Delta^+$ such that RL is a complement-free language.*

The next question to address is: given two languages $A, B \subseteq X^+$, when is AB a complement-free language?

Proposition 8. *Let $A, B \subseteq X^+$ be two non-empty languages. Assume that $A \cap \overline{B} = \emptyset$ and that $A \cup B$ is complement compliant. If $A_s \cap \overline{B_p} = \emptyset$ then AB is a complement-free language.*

Corollary 1. *Let $A, B \subseteq X^+$ be two non-empty languages. If $A \cup B$ is strictly complement compliant and $A_s \cap \overline{B_p} = \emptyset$ then AB is complement free.*

Remark that the preceding proposition and corollary hold also for any arbitrary morphic involution, not only for the complement involution.

The case of antimorphic involutions, i.e., of involutions of the type of the DNA involution, is slightly different.

Proposition 9. *Let $\theta : X^* \longrightarrow X^*$ be an antimorphic involution and $\emptyset \neq L \subseteq X^+$ be a language. Then L is θ-free if and only if L is θ-compliant and one of the following conditions hold:*

(1) $L_s \cap L_{is} = \emptyset$,
(2) $L_p \cap L_{ip} = \emptyset$,
(3) $L \cap \theta(L_s)L_p = \emptyset$,
(4) $L \cap L_s\theta(L_p) = \emptyset$.

Proposition 10. *Let $A, B \subseteq X^+$ be two non-empty languages and $\theta : X^* \longrightarrow X^*$ be an antimorphic involution. Assume, furthermore, that A and B are strictly θ-compliant and $A \cup B$ is θ-compliant. If $B_s \cap \theta(A_s) = A_p \cap \theta(B_p) = \emptyset$ then AB is a θ-free language.*

4 Decidability Issues

Given a family of sets of codewords, we ask if we can construct an effective algorithm that takes as input a description of a set of codewords from the given family and outputs the answer yes or no depending on whether or not the set is θ-free or θ-compliant (has good encoding properties). If such an algorithm exists, the question is deemed "decidable," otherwise it is "undecidable." This section considers the problem of deciding τ-compliance and τ-freedom where τ is the DNA involution. As in the case of infix and comma-free codes — see [8] — it turns out that these properties are decidable for regular languages but undecidable for context-free languages. We use regular expressions to represent regular languages and context-free grammars for context-free languages. If E is a regular expression (context-free grammar), $L(E)$ denotes the language represented by (generated by) E.

Proposition 11. *Let τ be the DNA involution. The following two problems are decidable.*

1. *Input: A regular expression E.*
   Output: YES or NO depending on whether $L(E)$ is τ-free.

2. *Input: A regular expression E.*
   Output: YES or NO depending on whether $L(E)$ is τ-compliant.

Proposition 12. *Let τ be the DNA involution. The following problems are undecidable:*

1. *Input: A context-free grammar F.*
   Output: YES or NO depending on whether or not $L(F)$ is τ-free.
2. *Input: A context-free grammar F.*
   Output: YES or NO depending on whether or not $L(F)$ is τ-compliant.

5 Splicing Systems That Preserve Good Encodings

The preceding sections studied conditions under which sets of DNA codewords have good encoding properties, like DNA compliance and τ-freedom, where τ is the DNA involution. The next question is to characterize initial sets of codewords having the additional feature that the good encoding properties are preserved during *any* computation starting out from the initial set. As a computational model we have chosen the computation by splicing [7,12,9]. Proposition 14 states that if the splicing base (initial set of codewords) is strictly θ-free then all the sequences that may appear along any computation will not violate the property of θ-freedom. Proposition 15 is a stronger result stating that for any computational system based on splicing one can construct an equivalent one with "good," i.e. θ-compliance and θ-freedom properties, being preserved during any computation. Finally, Proposition 16 provides a method of construction of a strictly θ-free code that can serve as splicing base, i.e. initial soup, for a computation. Moreover, it is proved that from the point-of-view of the efficiency of representing information, the constructed code is close to optimal.

A multiset M over an alphabet Σ is a collection of words in Σ^* such that a word can occur in M more than once. More formally, a multiset M is a mapping of Σ^* into $\mathbf{N}$ such that $M(w)$ is the number of copies of the word w in the multiset. For a multiset M, we write $\mathrm{supp}\,(M)$ to denote the set of (distinct) words that occur in M; that is, $\mathrm{supp}\,(M) = \{w \in \Sigma^* \mid M(w) > 0\}$. A splicing system — see [9] — is a quadruple $\gamma = (\Sigma, T, A, R)$ such that Σ is an alphabet, T is a subset of Σ, the set of terminal symbols, A is a multiset over Σ, the initial collection of words, and R is a set of splicing rules of the form $\alpha_1\#\beta_1\$\alpha_2\#\beta_2$ with $\alpha_1, \alpha_2, \beta_1, \beta_2$ being words in Σ^*. For two multisets M and M' we write $M \Longrightarrow_\gamma M'$, if there is a splicing rule $\alpha_1\#\beta_1\$\alpha_2\#\beta_2$ in R and two words in M of the form $x_1\alpha_1\beta_1y_1$ and $x_2\alpha_2\beta_2y_2$ such that M' is obtained from M by replacing those words with $x_1\alpha_1\beta_2y_2$ and $x_2\alpha_2\beta_1y_1$ – see [9]. The language, $L(\gamma)$, *generated* by a splicing system $\gamma = (\Sigma, T, A, R)$ is the set $\{w \in T^* \mid \exists M,\ A \Longrightarrow_\gamma M,\ w \in \mathrm{supp}\,(M)\}$. The *computation language of* γ is the set of words over Σ that can appear in any computation step of γ; that is, the language

$$\{w \mid w \in \mathrm{supp}\,(M) \text{ and } A \Longrightarrow_\gamma^* M, \text{ for some multiset } M\}$$

Note that the language generated by γ is a subset of the computation language of γ.

When the computation language of a splicing system involves only symbols of the alphabet Δ, with the usual functionality of these symbols, one wants to prevent situations like the ones described in the introduction and this requires choosing languages with certain combinatorial properties. In this section, we consider a type of splicing systems in which only symbols of the alphabet Δ are used and require that the language of computation of such systems is strictly θ-free, where θ is a morphic or antimorphic involution. To this end, we utilize Proposition 1 by requiring that the computation language is of the form K^* for some strictly θ-free subset K of Δ^+. Moreover, we require that K is a splicing base, as defined below, to ensure that, after performing a splicing operation between two words in K^*, the resulting words are still in K^*

Definition 5. *A splicing base is a set of words K such that, for all $x_1, x_2, y_1, y_2 \in \Delta^*$ and for all $v_1, u_1, v_2, u_2 \in K^*$ with $|v_1u_1| > 0$ and $|v_2u_2| > 0$, if $x_1v_1u_1y_1$, $x_2v_2u_2y_2 \in K^*$ then $x_1v_1u_2y_2$, $x_2v_2u_1y_1 \in K^*$.*

In the following lemma it is shown that the splicing base condition is equivalent to the following:

(C_K) For all $x \in \Delta^*$ 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 $x \in K^*$.

Remark: A splicing base is not necessarily a code. For example, $K = \{A, A^2\}$ is not a code but it is a splicing base.

Lemma 2.

(i) *Condition (C_K) implies the following: For all $x \in \Delta^*$ and for all $v \in K^+$, if xv is a prefix of K^* then $x \in K^*$, and if vx is a suffix of K^* then $x \in K^*$.*
(ii) *A set of words K is a splicing base if and only if (C_K) holds.*
(iii) *Every splicing base which is a code is a bifix code.*
(iv) *Every comma-free code is a splicing base.*

We define now a special type of splicing systems that involve only symbols of the alphabet Δ.

Definition 6. *Let K be a splicing base. A K-based splicing system is a quadruple $\beta = (K, T, A, R)$ such that $T \subseteq K$, A is a multiset with $\mathrm{supp}\,(A) \subseteq K^*$, and R is a subset of $K^*\#K^*\$K^*\#K^*$ with the property that $v_1\#u_1\$v_2\#u_2 \in R$ implies $|v_1u_1| > 0$ and $|v_2u_2| > 0$.*

For two multisets M and M', the relationship $M \Longrightarrow_\beta M'$ is defined as in the case of ordinary splicing systems – see [9]; that is, M' is obtained from M by applying a splicing rule to two words in $\mathrm{supp}\,(M)$. The language $L(\beta)$ *generated* by a K-based splicing system β is the set $\{w \in T^* \mid \exists M, A \Longrightarrow_\beta^* M$ and $w \in \mathrm{supp}\,(M)\}$. The *computation language* of the system β is the set $\{w \in \Delta^* \mid \exists M, A \Longrightarrow_\beta^* M$ and $w \in \mathrm{supp}\,(M)\}$. As shown next, this language is a subset of K^*.

Proposition 13. *For every splicing base K and for every K-based splicing system β, the computation language of β is a subset of K^*.*

Proposition 14. *Let θ be a morphic or antimorphic involution and let β be a K-based splicing system, where K is a splicing base. If K is strictly θ-free then the computation language of β is strictly θ-free.*

In the rest of the section we consider the question of whether an arbitrary splicing system γ can be translated to a K-based splicing system β over Δ such that the computation language of β is strictly θ-free and the languages generated by β and γ are isomorphic.

Definition 7. *Let θ be a morphic or antimorphic involution of Δ^* and let $\mathbf{K}$ be an infinite family of splicing bases such that every splicing base in $\mathbf{K}$ is a strictly θ-free code. The involution-free splicing class $\mathbf{C}_{\theta,\mathbf{K}}$ is the set of all K-based splicing systems, where $K \subseteq K'$ for some $K' \in \mathbf{K}$.*

Proposition 15. *Let $\mathbf{C}_{\theta,\mathbf{K}}$ be an involution-free splicing class and let Σ be an alphabet. For every splicing system $\gamma = (\Sigma, T, A, R)$ there is a K-based splicing system $\beta = (K, F, B, P)$ in $\mathbf{C}_{\theta,\mathbf{K}}$ and an isomorphism $f : T^* \to F^*$ such that the computation language of β is strictly θ-free and $L(\beta) = f(L(\gamma))$.*

We close the section with a simple construction of an infinite comma-free code, $K_{m,\infty}$, which is strictly free for the DNA involution τ. By Lemma 2 (iv), the code is a splicing base as well. Moreover, we define an infinite family $\{K_{m,n} \mid n \in \mathbf{N}\}$ of finite subsets of $K_{m,\infty}$ whose information rate tends to $(1 - 1/m)$. The involution τ and the family $\{K_{m,n} \mid n \in \mathbf{N}\}$ define a τ-free splicing class which can be used to 'simulate' every splicing system according to Proposition 15. The fact about the information rate of the codes $K_{m,n}$ concerns the efficiency (or redundancy) of these codes when used to represent information. In general, the information rate of a finite code K over some alphabet X is the ratio

$$\frac{\log_{|X|} |K|}{\left(\sum_{v \in K} |v|\right)/|K|}$$

When that rate is close to 1, the code is close to optimal.

Proposition 16. *Let m and n be non-negative integers with $m > 1$, let $K_{m,\infty} = A^m T(\Delta^{m-1}T)^* C^m$, and let $K_{m,n} = \cup_{i=0}^{n} A^m T(\Delta^{m-1}T)^i C^m$. Then,*

(i) *$K_{m,\infty}$ and, therefore, $K_{m,n}$ are comma-free codes and strictly τ-free, where τ is the DNA involution*
(ii) *The information rate of $K_{m,n}$ tends to $(1 - 1/m)$ as $n \to \infty$.*

References

1. J. Berstel and D. Perrin. *Theory of Codes*, Academic Press, Inc., Orlando, Florida, 1985.
2. R. Deaton, R. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens. Good encodings for DNA-based solutions to combinatorial problems. *DNA-based computers II*, in AMS DIMACS Series, vol. 44, L.F. Landweber and E. Baum, Eds., 1998, 247–258.
3. R. Deaton, M. Garzon, R. Murphy, D.R. Franceschetti, and S.E. Stevens. Genetic search of reliable encodings for DNA-based computation, *First Conference on Genetic Programming GP-96*, Stanford U., 1996, 9–15.
4. M. Garzon, R. Deaton, P. Neathery, D.R. Franceschetti, and R.C. Murphy. A new metric for DNA computing. *Proc. 2nd Genetic Programming Conference*, Stanford, CA, 1997, Morgan-Kaufmann, 472–478.
5. M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens, Jr., and M. Wittner. Genome encoding for DNA computing. *Proc. 3rd Genetic Programming Conference*, Madison, WI, 1998, 684–690.
6. T. Harju and J. Karhumäki. Morphisms. In *Handbook of Formal Languages,* vol. 1, G. Rozenberg, A. Salomaa, Eds., Springer-Verlag, Berlin, 1997, 439–510.
7. T. Head. Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors. *Bulletin of Mathematical Biology* 49 (1987), 737–759.
8. H. Jürgensen and S. Konstantinidis. Codes. In *Handbook of Formal Languages*, vol. 1, G. Rozenberg and A. Salomaa, Eds., Springer-Verlag, Berlin, 1997, 511–607.
9. L. Kari. DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer*, vol. 19, no. 2, Spring 1997, 9–22.
10. L. Kari, R. Kitto, and G. Thierrin. Codes, involutions and DNA encoding. Workshop on Coding Theory, London, Ontario, July 2000. To appear.
11. A. Marathe, A. Condon, and R. Corn. On combinatorial DNA word design. *DNA based Computers V*, DIMACS Series, E. Winfree, D. Gifford, Eds., AMS Press, 2000, 75–89.
12. G. Paun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
13. H.J. Shyr, *Free Monoids and Languages*, Hon Min Book Company, Taichung, Taiwan, R.O.C., 1991.