On properties of bond-free DNA languages

Lila Kari[1], Stavros Konstantinidis[2], Petr Sosík[1,3]

[1]Department of Computer Science, The University of Western Ontario, London, ON, Canada, N6A 5B7, {lila,sosik}@csd.uwo.ca

[2]Dept. of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3 Canada, s.konstantinidis@stmarys.ca

[3]Institute of Computer Science, Silesian University, Opava, Czech Republic

November 26, 2003

Abstract

The input data for DNA computing must be encoded into the form of single or double DNA strands. Due to the importance of this step for the success of computations, much attention has been devoted to studying suitable sets of codewords. As complementary parts of single strands can bind together forming a double-stranded DNA sequence, we have to impose some restrictions on these sets of codewords (=languages) to prevent them from interacting in undesirable ways. We recall a list of known properties of DNA languages which improve or guarantee their robustness during computation. Then we introduce a general framework in which we can characterize each of these properties by a solution of a uniform formal language inequation. This characterization allows us to construct (i) a uniform algorithm deciding in polynomial time whether a given DNA language possesses any of the studied properties, and (ii) in many cases also an algorithm to test whether a given DNA language is maximal with respect to the desired property. The possibility of construction of sets of DNA codewords based on this approach is also addressed.

1 Introduction

The main principle of DNA computing (or, more generally, molecular computing), can be summarized as follows: given a problem $P : I \longrightarrow O$, with an input from a set I and an output from a set O, we design an encoding of the input (respectively output) into a starting set (respectively final set) of bio-molecules. Then there must be a set of possible reactions such that for a given input set of molecules, these reactions produce a correct final set with respect to the used encoding. One must be able to construct the input set of molecules for a given input $i \in I$, then to ensure conditions for the desired reactions to run, and finally to detect the (non)presence of the final set of molecules in the reaction products. Unlike conventional computers, molecular computing devices would work in a maximally *parallel* manner, and an input (a set of molecules) of an elementary computing step (i.e. reaction) would be *consumed* during the reaction, producing a set of output molecules.

The most important molecules in DNA computing techniques are the single- and double-stranded *deoxyribonucleic acid* (DNA) molecules. They are composed primarily of *nucleotides* A, C, G, T attached to a sugar-phosphate backbone. The single-stranded DNA molecule can be represented as a linear oriented sequence of these nucleotides. Orientation is defined by convention from the 5' end to the 3' end of the strand. Two single stranded oppositely oriented DNA molecules can bind together under favorable conditions due to the *Watson-Crick complementarity principle:* A is complementary to T and C to G. Conversely, the double-stranded DNA molecule can be broken apart into two complementary single-stranded components. These two operations, called *hybridization (annealing)* and *denaturation (melting),* are fundamental techniques of DNA computing. There are also other bio-operations useful in DNA computing context, and we refer the reader to [2, 19] for further information.

Given this framework, [17] and others distinguish two elementary subproblems of the encoding design:

- *Positive* design problem: we design a set of input molecules such that there exists a way for the sequence of reactions to produce the correct final set.

- *Negative* design problem: the input set of molecules must not give way to the reactions that produce undesired molecules encoding a false output, and/or to consume the molecules in undesired reactions so that the correct final set cannot be produced.

The positive design problem is usually highly related to the specific experiment or computation at hand, and it is reported to be hard to find a general framework for its solution. In contrast, the negative design problem can be solved on a general basis by construction of a *library* of molecules which do not allow undesired mutual reactions. These conclusions have been adopted by DNA computing researchers and there is a significant number of papers devoted to either positive or negative DNA encoding design problem. We refer the reader e.g. to [1, 5, 7, 8, 9, 12, 14, 16] for studies of properties of such a library and for methods of its construction. There are also many subtler questions concerning the design problems. Various strengths of hybridization bonds due to the DNA primary and secondary structure, free energy, melting temperature and other factors are addressed in the literature. General information and further references can be found e.g. in [2, 19].

In this paper we focus on the problem of negative design of sets of DNA codewords which cannot produce undesirable mutual bonds. In Section 2 we give necessary formal language prerequisites and a list of thirteen useful properties of DNA codeword libraries studied by various authors. Section 3 gives insight into binary word operations on trajectories which are extensively used in the remainder of the paper.

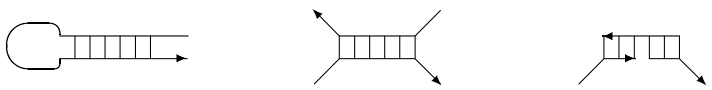
In Section 4 we introduce the key concept of the *bond-free language property*, and show that eight of the previously studied properties are its special cases. Moreover, the bond-free property has an intuitive geometrical interpretation and favorable mathematical features. In particular, one can construct a general quadratic-time algorithm deciding whether a given regular set of codewords satisfies any of the mentioned special cases of the bond-free property.

We then observe that the bond-free property is definable via language inequations. By utilizing and improving recent results in [13] on language inequations, we show in Section 5 that for six instances of the bond-free property the *maximality* problem is decidable. This means that there is an algorithm to decide whether or not a given regular set of codewords can be further extended without the loss of the property or not. For the case of finite sets we construct a polynomial-time algorithm deciding the maximality of the θ-compliant property. Finally, if an extension is possible, we give formulas for obtaining the extended set of codewords.

The same problems are addressed in Sections 6 and 7 for the so-called *strictly bond-free* properties which have the added feature of excluding also exact matching pairs of complementary codewords. This time we show that nine of the properties reported in the literature fit into our general framework, with the same benefits as in the "non-strict case. For eight instances of them we are able to decide also the maximality of regular sets of codewords, while for θ-non-overlapping property we can achieve this in polynomial time. The corresponding formulas for obtaining an extended set of codewords are also given.

2 Undesired bonds in DNA libraries

Two types of unwanted hybridization are usually considered: *intramolecular* (within a molecule) and *intermolecular* (between two or more molecules). The intramolecular hybridization happens when two mutually complementary sequences appearing in the same DNA strand bind together forming a hairpin, see Fig. 1 (a). Intermolecular hybridization may, for example, involve two complementary sequences which are parts of two different strands (b), or one of them is a portion of concatenation of two strands (c).



(a) (b) (c)

Figure 1: Types of undesired (a) intramolecular and (b), (c) intermolecular hybridizations.

In the remainder of this paper we represent the single-stranded DNA molecules by strings over the *DNA alphabet* $\Delta = \{A, C, T, G\}$, and we reduce their mutual reactions to formal manipulation of these strings. Therefore, some formal language prerequisites are necessary.

An *alphabet* is a finite and nonempty set of symbols. In the sequel we shall use a fixed non-singleton alphabet Σ, as a generalization of the natural DNA alphabet Δ.

The set of all words (over Σ) is denoted by Σ^*. This set includes the *empty word* λ. The length of a word w is denoted by $|w|$. $|w|_x$ denotes the number of occurrences of x within u, for $w \in \Sigma^*$, $x \in \Sigma^+$. For a nonnegative integer n and a word w, we use w^n to denote the word that consists of n concatenated copies of w. We denote the mirror image of the word w by w^R. A word v is a *subword* of w if $w = xvy$ for some words x and y. In this case, if $|x| + |y| > 0$ then v is a *proper subword*. By $\mathrm{Sub}(w)$ we denote the set of all subwords of w. For a positive integer k, we use $\mathrm{Sub}_k(w)$ to denote the set of subwords of length k of w. For prefixes we use analogously the notation $\mathrm{Pref}(w)$ and $\mathrm{Pref}_k(w)$, respectively.

A language L is a set of words, or equivalently a subset of Σ^*. A language is said to be λ-free if it does not contain the empty word. If n is a nonnegative integer, we write L^n for the language consisting of all words of the form $w_1 \cdots w_n$ such that each w_i is in L. We also write L^* for the language $L^0 \cup L^1 \cup L^2 \cup \cdots$, and L^+ for the language $L^* - \{\lambda\}$.

The notation L^c represents the complement of the language L; that is, $L^c = \Sigma^* - L$. The mirror image of L is $L^R = \{w^R \,|\, w \in L\}$. By Sub$(L)$ we denote the set of all subwords of L, i.e. $\text{Sub}(L) = \bigcup_{w\in L} \text{Sub}(w)$.

A mapping $\alpha : \Sigma^* \to \Sigma^*$ is called a *morphism* (*anti-morphism)* of Σ^* if $\alpha(uv) = \alpha(u)\alpha(v)$ (respectively $\alpha(uv) = \alpha(v)\alpha(u)$) for all $u, v \in \Sigma^*$. Note that both a morphism and an anti-morphism of Σ^* are completely defined if we define their values on the letters of Σ.

An *involution* $\theta : \Sigma \to \Sigma$ of Σ is a mapping such that θ^2 is equal to the identity mapping, i.e., $\theta(\theta(x)) = x$ for all $x \in \Sigma$. It follows then that an involution θ is bijective and $\theta = \theta^{-1}$. The identity mapping is a trivial example of an involution. In general, if $f : \Sigma \to \Sigma$ is an involution, then Σ can be partitioned into $\Sigma = \Pi \cup \Pi' \cup \Gamma$ where $|\Pi| = |\Pi'|$ and, for every $a \in \Pi$ we have $f(a) = a'$, $f(a') = a$, $a' \in \Pi'$, while $f(b) = b$ for all $b \in \Gamma$. If $\Pi = \Pi' = \emptyset$ then f is the identity on Σ, while if $\Gamma = \emptyset$ f is a sort of complement function on Σ which maps every element of Π into an element of Π' and vice versa.

An involution of Σ can be extended to either a morphism or an antimorphism of Σ^*. For example, if the identity of Σ is extended to a morphism of Σ^*, we obtain the identity involution of Σ^*. However, if we extend the identity of Σ to an antimorphism of Σ^* we obtain instead the mirror-image involution of Σ^* that maps each word u into u^R where

$$u = a_1 a_2 \ldots a_k, \;\; u^R = a_k \ldots a_2 a_1, a_i \in \Sigma, 1 \le i \le k.$$

If we consider the DNA-alphabet Δ, then the mapping $\tau : \Delta \to \Delta$ defined by $\tau(A) = T, \tau(T) = A, \tau(C) = G, \tau(G) = C$ can be extended in the usual way to an antimorphism of Δ^* that is also an involution of Δ^*. This involution formalizes the notion of Watson-Crick complement of a DNA sequence and will therefore be called the *DNA involution*, [12]. By convention, a word $w = a_1 a_2 \ldots a_n$ in Δ^* will signify the DNA single strand $5' - a_1 a_2 \ldots a_n - 3'$. Then single strands $w_1, w_2 \in \Delta^*$ are complementary iff $w_1 = \tau(w_2)$.

Now we are ready to give a list of desirable properties of a DNA language $L \subseteq \Sigma^+$ which have been defined in [7, 12, 14].

(A) **θ-non-overlapping**: $L \cap \theta(L) = \emptyset$.

(B) **θ-compliant**: $\forall w \in L,\; x, y \in \Sigma^*,\; w, x\theta(w)y \in L \Rightarrow xy = \lambda$.

(C) **θ-p-compliant**: $\forall w \in L,\; y \in \Sigma^*,\; w, \theta(w)y \in L \Rightarrow y = \lambda$.

(D) **θ-s-compliant**: $\forall w \in L,\; y \in \Sigma^*,\; w, y\theta(w) \in L \Rightarrow y = \lambda$.

(E) **strictly θ-compliant**: both θ-compliant and θ-non-overlapping.

(F) **θ-free**: $L^2 \cap \Sigma^+\theta(L)\Sigma^+ = \emptyset$.

(G) **θ-sticky-free**: $\forall w \in \Sigma^+,\; x, y \in \Sigma^*,\; wx, y\theta(w) \in L \Rightarrow xy = \lambda$.

(H) **θ-3′-overhang-free**: $\forall w \in \Sigma^+,\; x, y \in \Sigma^*,\; wx, \theta(w)y \in L \Rightarrow xy = \lambda$.

(I) **θ-5′-overhang-free**: $\forall w \in \Sigma^+,\; x, y \in \Sigma^*,\; xw, y\theta(w) \in L \Rightarrow xy = \lambda$.

(J) **θ-overhang-free**: both θ-3′-overhang-free and θ-5′-overhang-free.

For convenience, we agree to say that a language L containing the empty word has one of the above properties if $L \setminus \{\lambda\}$ has that property. Observe that (F) corresponds to Fig. 1 (c), while others are special cases of (b).

In [9], a θ-non-overlapping language is called to be *strictly* θ. Generally, if any other property holds in conjunction with (A), we add the qualifier *strictly*. We have already used this notation for the property (E). Both *strict* and *non-strict* properties turn out to be useful in certain situations.

For example it might be useful to find out whether or not a language L has a non-strict property in a situation such as follows. The usual way to check for the presence of a certain single-stranded molecule is to add to the solution the complement of it and use enzymes to destroy any molecules which are not double stranded (possibly with blunt ends). Let the solution be non-strictly bond-free (exact matches are allowed). Then, the presence of a molecule indicates a perfect hybridization, hence the presence of the desired molecule.

Further properties have been defined in [9] for a language L. Observe that the following property corresponds to Fig. 1 (a):

(K) $\theta(k, m_1, m_2)$**-subword compliant**: $\forall u \in \Sigma^k$, $\Sigma^* u \Sigma^m \theta(u) \Sigma^* \cap L = \emptyset$ for $k > 0$, $m_1 \leq m \leq m_2$.

(L) θ-k-**code**: $\mathrm{Sub}_k(L) \cap \mathrm{Sub}_k(\theta(L)) = \emptyset$, $k > 0$.

In [5] the following property is defined for $\theta = I$, the identity relation. A language L is called

(M) **solid** if:

1. $\forall x, y, u \in \Sigma^*$, $u, xuy \in L \Rightarrow xy = \lambda$, and
2. $\forall x, y \in \Sigma^*, u \in \Sigma^+$, $xu, uy \in L \Rightarrow xy = \lambda$.

L is *solid relative to an* $M \subseteq \Sigma^*$ if 1. and 2. above hold only for $w = pxuyq \in M$. L is called *comma-free* if it is solid relative to L^*. Solid languages are also used in [14] as a tool for constructing error-detecting DNA languages that are invariant under bio-operations.

Figure 2 shows the hierarchy of some of the above language properties. Arrows stand for inclusion relations among language classes corresponding to the properties.

Example 2.1 *Consider the language $L = \{A^n T^n \mid n \geq 1\} \subset \Delta^+$, and the antimorphism τ. Observe that $\tau(L) = L$. We can deduce that L is:*

- *neither τ-non-overlapping, nor τ-k-code for any $k \geq 1$;*
- *not τ-compliant, as for $w = A^n T^n$, $x = A$, $y = T$ we have $w, x\tau(w)y \in L$;*
- *τ-p-compliant, as $w, \theta(w)y \in L$ implies $w = A^n T^n$, $y = \lambda$; similarly, L is τ-s-compliant;*
- *not τ-free, as $A^n T^n A^m T^m$, $n, m > 1$ is both in L^2 and in $\Delta^+ L \Delta^+$;*
- *not τ-sticky-free, as for $w = y = A^n$ $x = T^n$ we have $wx, y\tau(w) \in L$;*
- *τ-3′-overhang-free, as $wx, \tau(w)y \in L$ implies $w = A^n T^m$, $x = T^{n-m}$, $y = T^{m-n}$ and hence $xy = \lambda$; similarly, L is τ-5′-overhang-free and hence τ-overhang-free;*

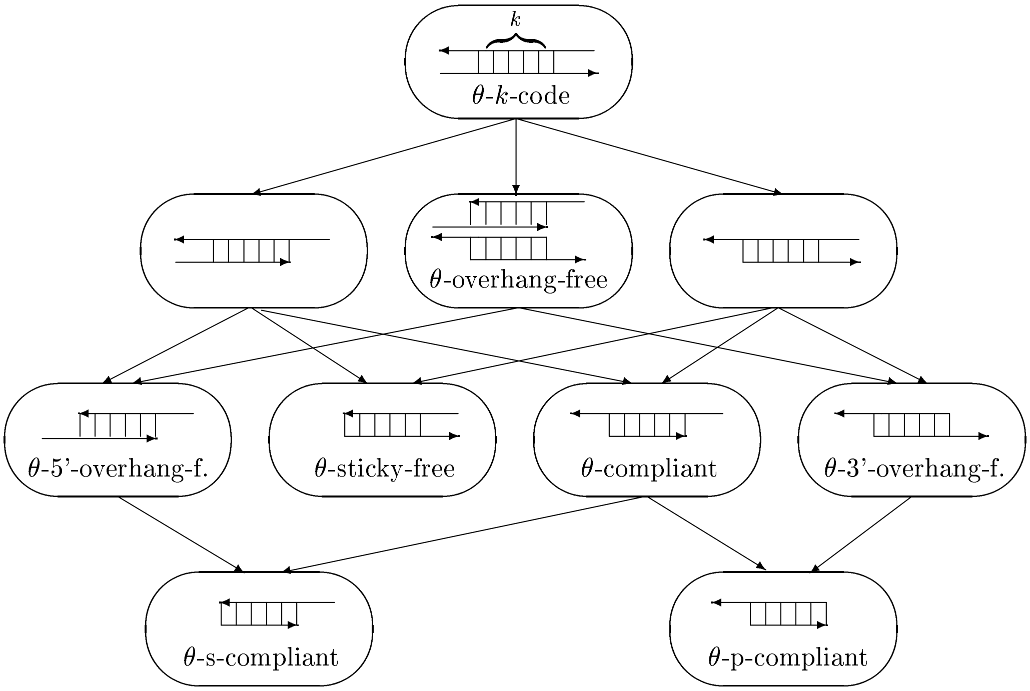


Figure 2: Classes of languages satisfying various DNA language properties.

- *not $\theta(k, m_1, m_2)$-subword compliant for any k, m_1, m_2.*

Besides the inclusion relations in Fig. 2, there are further relations among DNA libraries which are free of various types of (a), (b) or (c) bonds. The following results are shown in [9, 14] or follow easily by the definitions (assume that $\emptyset \neq L \subseteq \Sigma^+$).

(i) Let θ be a morphism. Then L satisfies any of the properties (A)–(M) iff $\theta(L)$ does so.

(ii) Let θ be an antimorphism. Then L satisfies any of the properties (A), (B), (E)–(G), (J)–(L) iff $\theta(L)$ does so. Moreover, L satisfies (C) iff $\theta(L)$ satisfies (D), and similarly for (H) and (I).

(iii) Denote by L_p (L_s) the set of all proper nonempty prefixes (suffixes, respectively) of L. L is θ-sticky-free iff $L_p \cap \theta(L_s) = \emptyset$ and L is both θ-p-compliant and θ-s-compliant.

(iv) Let θ be a morphism, L be θ-compliant and θ-sticky-free. Then L is θ-free.

(v) Let θ be an antimorphism, L be θ-compliant and either θ-3′-overhang-free or θ-5′-overhang-free. Then L is θ-free.

(vi) If L is θ-free, then it is θ-compliant.

(vii) If L is strictly θ and L^2 is $\theta(k, 1, m)$-subword-compliant for all $m \geq 1$, then L is strictly θ-k-code.

(viii) If L is θ-k-code then it is $\theta(k, 1, m)$-subword-compliant for all $m \geq 1$. Furthermore, if $k \leq |x|/2$ for all $x \in L$ then L is θ-free.

3 Binary word operations

Binary word operations are extensively used in the following sections as an important tool for representing interaction of DNA molecules. A *binary word operation* is a mapping $\diamondsuit : \Sigma^* \times \Sigma^* \to 2^{\Sigma^*}$, where 2^{Σ^*} is the set of all subsets of Σ^*. Hence the result of the operation $\diamondsuit$ with operands $u, v \in \Sigma^*$ is generally a language $(u \diamondsuit v) \subseteq \Sigma^*$. In some important particular cases we have $|u \diamondsuit v| = 1$ for $u, v \in \Sigma^*$. If there is no risk of misunderstanding, we may then assume $u \diamondsuit v = w$, $w \in \Sigma^*$, instead of the singleton language $\{w\} \subseteq \Sigma^*$. A typical example is the catenation operation $u \cdot v$.

We extend binary operations to any languages X and Y as follows:

$$X \diamondsuit Y = \bigcup_{u \in X, v \in Y} u \diamondsuit v. \tag{1}$$

Definition 3.1 *([11]) Let $\diamondsuit$ be an operation. The left inverse $\diamondsuit^l$ of $\diamondsuit$ is defined as*

$$w \in (x \diamondsuit v) \textit{ iff } x \in (w \diamondsuit^l v), \textit{ for all } v, x, w \in \Sigma^*,$$

and the right inverse $\diamondsuit^r$ of $\diamondsuit$ is defined as

$$w \in (u \diamondsuit y) \textit{ iff } y \in (u \diamondsuit^r w), \textit{ for all } u, y, w \in \Sigma^*.$$

Let $\diamondsuit$ be a binary word operation. The word operation $\diamondsuit'$ defined by $u \diamondsuit' v = v \diamondsuit u$ is called reversed $\diamondsuit$. Below we list a few binary word operations together with their inverses [10, 13, 18].

Catenation: $u \cdot v = \{uv\}$, with $\cdot^l = \longrightarrow_{\mathrm{rq}}$ and $\cdot^r = \longrightarrow_{\mathrm{lq}}$.

Left quotient: $u \longrightarrow_{\mathrm{lq}} v = \{w\}$ if $u = vw$, with $\longrightarrow_{\mathrm{lq}}^l = \cdot'$ and $\longrightarrow_{\mathrm{lq}}^r = \cdot$.

Right quotient: $u \longrightarrow_{\mathrm{rq}} v = \{w\}$ if $u = wv$, with $\longrightarrow_{\mathrm{rq}}^l = \cdot$ and $\longrightarrow_{\mathrm{rq}}^r = \longrightarrow_{\mathrm{lq}}'$.

Insertion: $u \longleftarrow v = \{u_1 v u_2 \mid u = u_1 u_2\}$, with $\longleftarrow^l = \longrightarrow$ and $\longleftarrow^r = \rightleftharpoons'$.

Deletion: $u \longrightarrow v = \{u_1 u_2 \mid u = u_1 v u_2\}$, with $\longrightarrow^l = \longleftarrow$ and $\longrightarrow^r = \rightleftharpoons$.

Dipolar deletion: $u \rightleftharpoons v = \{w \mid u = v_1 w v_2, v = v_1 v_2\}$, with $\rightleftharpoons^l = \longleftarrow'$ and $\rightleftharpoons^r = \longrightarrow$.

Shuffle (or scattered insertion): $u \sqcup\!\sqcup v = \{u_1 v_1 \cdots u_k v_k u_{k+1} \mid k \geq 1, u = u_1 \cdots u_k u_{k+1}, v = v_1 \cdots v_k\}$, with $\sqcup\!\sqcup^l = \rightsquigarrow$ and $\sqcup\!\sqcup^r = \rightsquigarrow'$.

Scattered deletion: $u \rightsquigarrow v = \{u_1 \cdots u_k u_{k+1} \mid k \geq 1, u = u_1 v_1 \cdots u_k v_k u_{k+1}, v = v_1 \cdots v_k\}$, with $\rightsquigarrow^l = \sqcup\!\sqcup$ and $\rightsquigarrow^r = \rightsquigarrow$.

Balanced literal shuffle: $u \sqcup\!\sqcup_{\mathrm{bl}} v = \{u_1 v_1 \cdots u_k v_k \mid k \geq 0, u = u_1 \cdots u_k, v = v_1 \cdots v_k, u_i, v_i \in \Sigma, 1 \leq i \leq k\}$, with $\sqcup\!\sqcup_{\mathrm{bl}}^l = \rightsquigarrow_{\mathrm{bl}}$. Observe that $u \sqcup\!\sqcup_{\mathrm{bl}} v = \emptyset$ iff $|u| \neq |v|$.

Balanced literal deletion: $u \rightsquigarrow_{\mathrm{bl}} v = \{u_1 \cdots u_k \mid k \geq 0, u = u_1 v_1 \cdots u_k v_k, v = v_1 \cdots v_k, u_i, v_i \in \Sigma, 1 \leq i \leq k\}$, with $\rightsquigarrow_{\mathrm{bl}}^l = \sqcup\!\sqcup_{\mathrm{bl}}$.

If x and y are symbols in $\{l, r, '\}$, the notation $\diamondsuit^{xy}$ represents the operation $(\diamondsuit^x)^y$. The following identities between operations of the form $\diamondsuit^{xy}$ have been established in [13]:

$$\diamondsuit^{ll} = \diamondsuit^{rr} = \diamondsuit'' = \diamondsuit,$$

$$\diamondsuit'^{l} = \diamondsuit^{r'} = \diamondsuit^{lr}.$$

For the composition and inversion of more complicated word operations, the following notations and technical results will be helpful.

Definition 3.2 *Let $x, x_1, x_2, y, y_1, y_2 \in \Sigma^*$ and let $\diamondsuit_1, \diamondsuit_2$ be binary word operations. We define the composed operations $(\diamondsuit_1 ; \diamondsuit_2)$ and $(\diamondsuit_1 : \diamondsuit_2)$ as follows:*

(i) $x(\diamondsuit_1 ; \diamondsuit_2)(y_1 ; y_2) = (x \diamondsuit_1 y_1) \diamondsuit_2 y_2$,

(ii) $(x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)y = x_1 \diamondsuit_1 (x_2 \diamondsuit_2 y)$.

Lemma 3.3 *(i)* $(x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)y = y(\diamondsuit_2' ; \diamondsuit_1')(x_2 ; x_1)$,

(ii) $x(\diamondsuit_1 ; \diamondsuit_2)^l(y_1 ; y_2) = x(\diamondsuit_2^l ; \diamondsuit_1^l)(y_2 ; y_1)$,

(iii) $(x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)^r y = (x_2 : x_1)(\diamondsuit_2^r : \diamondsuit_1^r)y$.

Proof.

(i) $(x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)y = x_1 \diamondsuit_1 (x_2 \diamondsuit_2 y) = (x_2 \diamondsuit_2 y) \diamondsuit_1' x_1 = (y \diamondsuit_2' x_2) \diamondsuit_1' x_1 = y(\diamondsuit_2' ; \diamondsuit_1')(x_2 ; x_1)$.

(ii) The statement has been proven in [13].

(iii)
$$\begin{array}{lclcl}
y \in (x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)^r z & \text{iff} & z \in (x_1 : x_2)(\diamondsuit_1 : \diamondsuit_2)y & \text{iff} \\
z \in y(\diamondsuit_2' ; \diamondsuit_1')(x_2 ; x_1) & \text{iff} & y \in z(\diamondsuit_2' ; \diamondsuit_1')^l (x_2 ; x_1) & \text{iff} \\
y \in z(\diamondsuit_1'^{l} ; \diamondsuit_2'^{l})(x_1 ; x_2) & \text{iff} & y \in z(\diamondsuit_1^{r'} ; \diamondsuit_2^{r'})(x_1 ; x_2) & \text{iff} \\
y \in (z \diamondsuit_1^{r'} x_1) \diamondsuit_2^{r'} x_2 & \text{iff} & y \in x_2 \diamondsuit_2^r (x_1 \diamondsuit_1^r z) & \text{iff} \\
y \in (x_2 : x_1)(\diamondsuit_2^r : \diamondsuit_1^r)z. & & &
\end{array}$$

□

Now we introduce the generalizing concept of word operations on trajectories [4, 15, 18]. Consider an *trajectory alphabet* $V = \{0, 1\}$ and assume $V \cap \Sigma = \emptyset$. We call *trajectory* any string $t \in V^*$. A trajectory is essentially a syntactical condition which specifies how an operation $\diamondsuit$ is applied to the letters of its two operands. Let $t \in V^*$ be a trajectory and let α, β be two words over Σ.

Definition 3.4 *The shuffle of α with β on the trajectory t, denoted by $\alpha \sqcup\!\sqcup_t \beta$, is defined as follows:*

$$\alpha \sqcup\!\sqcup_t \beta = \{\alpha_1\beta_1 \ldots \alpha_k\beta_k \mid \alpha = \alpha_1 \ldots \alpha_k,\ \beta = \beta_1 \ldots \beta_k,\ t = 0^{i_1}1^{j_1} \ldots 0^{i_k}1^{j_k},\ \text{where } |\alpha_m| = i_m \text{ and } |\beta_m| = j_m \text{ for all } m,\ 1 \le m \le k\}.$$

Example 3.5 *Let $\alpha = a_1a_2 \ldots a_8$, $\beta = b_1b_2 \ldots b_5$ and assume that $t = 0^31^20^310101$. The shuffle of α and β on the trajectory t is:*

$$\alpha \sqcup\!\sqcup_t \beta = \{a_1a_2a_3b_1b_2a_4a_5a_6b_3a_7b_4a_8b_5\}.$$

Notice that the above definition implies $\alpha \sqcup\!\sqcup_t \beta = \emptyset$ if $|\alpha| \neq |t|_0$ or $|\beta| \neq |t|_1$. Analogously, deletion on a trajectory is defined.

Definition 3.6 *The deletion of β from α on the trajectory t is the following binary word operation:*

$$\begin{aligned}\alpha \rightsquigarrow_t \beta \quad = \quad & \{\alpha_1 \ldots \alpha_k \mid \alpha = \alpha_1\beta_1 \ldots \alpha_k\beta_k,\ \beta = \beta_1 \ldots \beta_k,\ t = 0^{i_1}1^{j_1} \ldots 0^{i_k}1^{j_k},\ \textit{where} \\ & |\alpha_m| = i_m \textit{ and } |\beta_m| = j_m \textit{ for all } m,\ 1 \leq m \leq k\}.\end{aligned}$$

Example 3.7 *Let $\alpha = babaab$, $\beta = bb$ and assume that $t = 001001$. The deletion of β from α on the trajectory t is:*

$$\alpha \rightsquigarrow_t \beta = \{baaa\}.$$

Notice also that for given α, β, t we have always $|\alpha \sqcup\!\sqcup_t \beta| \leq 1$, $|\alpha \rightsquigarrow_t \beta| \leq 1$.

A *set of trajectories* is any set $T \subseteq V^*$. The *shuffle (deletion) of α with β on the set* T, denoted by $\alpha \sqcup\!\sqcup_T \beta$ ($\alpha \rightsquigarrow_T \beta$), is:

$$\alpha \diamondsuit_T \beta = \bigcup_{t \in T} \alpha \diamondsuit_t \beta, \tag{2}$$

where $\diamondsuit$ stands for $\sqcup\!\sqcup$ or $\rightsquigarrow$, respectively. The operations $\sqcup\!\sqcup_T$ and $\rightsquigarrow_T$ generalize to languages due to the general principle (1). Some basic operations of sequential deletion of words and languages are particular cases of the shuffle or deletion on trajectories.

- Let $T = 0^*1^*$. Then $\sqcup\!\sqcup_T \ = \ \cdot$, the catenation, and $\rightsquigarrow_T = \longrightarrow_{\mathrm{rq}}$, the right quotient.
- Let $T = 1^*0^*$. Then $\sqcup\!\sqcup_T \ = \ \cdot'$, the reversed catenation, and $\rightsquigarrow_T = \longrightarrow_{\mathrm{lq}}$, the left quotient.
- For $T = V^*$ we have $\sqcup\!\sqcup_T \ = \ \sqcup\!\sqcup$ and $\rightsquigarrow_T = \rightsquigarrow$.
- Let $T = (01)^*$. Then $\sqcup\!\sqcup_T \ = \ \sqcup\!\sqcup_{\mathrm{bl}}$ and $\rightsquigarrow_T = \rightsquigarrow_{\mathrm{bl}}$.

The following results are proven in [4, 15, 18] or follow directly by proof techniques used ibidem.

Lemma 3.8 *Let T be a set of trajectories. Then*

(i) $\sqcup\!\sqcup^l_T \ = \rightsquigarrow_T$ *and* $\sqcup\!\sqcup^r_T \ = \rightsquigarrow'_{\widetilde{T}}$,

(ii) $\rightsquigarrow^l_T = \ \sqcup\!\sqcup_T$ *and* $\rightsquigarrow^r_T = \rightsquigarrow_{\widetilde{T}}$,

where $\widetilde{T}$ is the set of trajectories obtained by replacing all 0's for 1's and vice versa in all the trajectories of T.

Lemma 3.9 *Let T be a set of trajectories. The following assertions are equivalent:*

(i) For all regular languages L_1, L_2, $L_1 \sqcup\!\sqcup_T L_2$ is a regular language.

(ii) T is a regular language.

Lemma 3.10 *For all regular languages L_1, L_2, and a regular set of trajectories T, $L_1 \rightsquigarrow_T L_2$ is a regular language.*

4 Bond-Free DNA Languages

Most of the DNA language properties defined in Section 2 are intended to prevent unwanted bonds between two distinct DNA strands. These strands need not be perfect complements of each other, but they may also contain some blunt ends or other slight differences. Word operations on trajectories prove useful when looking for a general approach to various types of properties which differ by type of bonds and free ends, see Fig. 2. Many of such properties can be expressed in the following general form.

Definition 4.1 *A class $\mathcal{P}$ of languages over Σ is called a* bond-free property of degree 2 *if there exist binary word operations $\diamondsuit_{\rm dn}$, $\diamondsuit_{\rm up}$ such that for an arbitrary $L \subseteq \Sigma^*$, $L \in \mathcal{P}$ holds iff*

$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*, (w \diamondsuit_{\rm dn} x \cap L \neq \emptyset,\ w \diamondsuit_{\rm up} y \cap \theta(L) \neq \emptyset) \Rightarrow\ xy = \lambda. \qquad (3)$$

Hence each DNA language property based on bonds of *two* DNA strands, that can be expressed in the form (3), is called a *bond-free property of degree 2.* In the remainder of this paper we write simply *bond-free property* for bond-free property of degree two. In most cases of interest the operations $\diamondsuit_{\rm dn}$, $\diamondsuit_{\rm up}$ take a more specific form. Particularly, in this and the following section we assume that

$$\diamondsuit_{\rm dn} = ⧢_{T_{\rm dn}}, \qquad \diamondsuit_{\rm up} = ⧢_{T_{\rm up}}$$

for some trajectory sets $T_{\rm dn}, T_{\rm up} \subseteq V^*$.

Theorem 4.2 *The language properties (B), (C), (D), (G), (H), (I), (M.1), (M.2) are bond-free properties. Moreover, the associated sets of trajectories $T_{\rm dn}, T_{\rm up}$ are regular.*

Proof. Assume that θ is an antimorphism and define the sets of trajectories $T_{\rm dn}$, $T_{\rm up}$ as follows.

(B) $T_{\rm dn} = 0^+$, $T_{\rm up} = 1^*0^+1^*$.

(C) $T_{\rm dn} = 0^+$, $T_{\rm up} = 1^*0^+$.

(D) $T_{\rm dn} = 0^+$, $T_{\rm up} = 0^+1^*$.

(G) $T_{\rm dn} = 0^+1^*$, $T_{\rm up} = 0^+1^*$.

(H) $T_{\rm dn} = 0^+1^*$, $T_{\rm up} = 1^*0^+$.

(I) $T_{\rm dn} = 1^*0^+$, $T_{\rm up} = 0^+1^*$.

Consider e.g. the property (H), θ-3'-overhang-freedom. Then $w ⧢_{T_{\rm dn}} x = \{wx\}$ and $w ⧢_{T_{\rm up}} y = \{yw\}$. The relations in (3) take the form $wx \in L$, $yw \in \theta(L)$. This is equivalent to $wx \in L$, $\theta(w)\theta(y) \in L$. As $xy = \lambda$ iff $x\theta(y) = \lambda$, (3) corresponds to the definition of (H) in Section 2. The proofs of the other mentioned properties are analogous.

If θ is a morphism, then all the sets of trajectories $T_{\rm up}$ must be replaced by the reversed sets $T_{\rm up}^R$ and we obtain $\theta(wy) = \theta(w)\theta(y)$, the rest of the proof remaining unchanged. As the property (M) is defined only for $\theta = I$, the identity on Σ^*, one can easily verify that, in this case,

(M.1) $T_{\rm dn} = 0^*$, $T_{\rm up} = 1^*0^*1^*$.

(M.2) $T_{\mathrm{dn}} = 1^*0^+$, $T_{\mathrm{up}} = 0^+1^*$.

$\square$

Observe that $T_{\mathrm{dn}}, T_{\mathrm{up}}$ for a certain property corresponds to the "shape" of the bonds prohibited in languages satisfying the property. This correspondence can be even enhanced by the concept of *DNA trajectories* — strings over the alphabet $V_{\mathrm{DNA}} = \{\binom{b}{b}, \binom{f}{f}, \binom{f}{\lambda}, \binom{\lambda}{f}\}$. In this notation b stands for a bonded letter and f for a free letter in a DNA sequence.

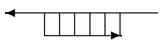
Let $\varphi_{\mathrm{up}}, \varphi_{\mathrm{dn}} : V_{\mathrm{DNA}} \longrightarrow V$ be morphisms defined as follows:

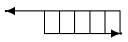
$$\begin{array}{llll} \varphi_{\mathrm{up}}(\binom{b}{b}) = 0, & \varphi_{\mathrm{up}}(\binom{f}{y}) = 1, & \varphi_{\mathrm{up}}(\binom{\lambda}{y}) = \lambda, & y \in \{f, \lambda\}, \\ \varphi_{\mathrm{dn}}(\binom{b}{b}) = 0, & \varphi_{\mathrm{dn}}(\binom{x}{f}) = 1, & \varphi_{\mathrm{dn}}(\binom{x}{\lambda}) = \lambda, & x \in \{f, \lambda\}. \end{array}$$

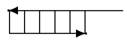
For any bond-free property associated with a pair of sets $T_{\mathrm{dn}}, T_{\mathrm{up}}$ we construct the set of DNA trajectories S as follows:

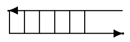
$$S = \{s \in V_{\mathrm{DNA}}^* \mid \varphi_{\mathrm{dn}}(s) \in T_{\mathrm{dn}},\ \varphi_{\mathrm{up}}(s) \in T_{\mathrm{up}}\}.$$

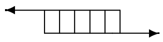
For the properties (B), (C), (D), (G), (H), (I) we obtain the following sets of DNA trajectories (compare with Figure 2):

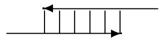
(B) **θ-compliant**: $S_B = \binom{f}{\lambda}^* \binom{b}{b}^+ \binom{f}{\lambda}^*$ 

(C) **θ-p-compliant**: $S_C = \binom{f}{\lambda}^* \binom{b}{b}^+$ 

(D) **θ-s-compliant**: $S_D = \binom{b}{b}^+ \binom{f}{\lambda}^*$ 

(G) **θ-sticky-free**: $S_G = \binom{b}{b}^+ \binom{f}{f}^* (\binom{f}{\lambda}^* \cup \binom{\lambda}{f}^*)$ 

(H) **θ-3′-overhang-free**: $S_H = \binom{f}{\lambda}^* \binom{b}{b}^+ \binom{\lambda}{f}^*$ 

(I) **θ-5′-overhang-free**: $S_I = \binom{\lambda}{f}^* \binom{b}{b}^+ \binom{f}{\lambda}^*$ 

As the following result shows, the DNA trajectories allow us to establish mutual relations between DNA language properties easily. For a set of DNA trajectories S and a language $L \subseteq \Sigma^*$, denote

$$L_S = \{xy \in \Sigma^* \mid w \sqcup\!\sqcup_{\varphi_{\mathrm{dn}}(S)} x \cap L \neq \emptyset,\ w \sqcup\!\sqcup_{\varphi_{\mathrm{up}}(S)} y \cap \theta(L) \neq \emptyset, \text{ for some } w \in \Sigma^+\}. \quad (4)$$

Denote further by $\mathcal{P}_S$ a bond-free property associated with the set of DNA trajectories S. Comparing with Definition 4.1, one can observe that $L \in \mathcal{P}_S$ holds for a language $L \subseteq \Sigma^*$ iff $L_S \subseteq \{\lambda\}$.

Theorem 4.3 *Let S_1, S_2 be sets of DNA trajectories.*

(i) $(S_1 \subseteq S_2) \Rightarrow (\mathcal{P}_{S_1} \supseteq \mathcal{P}_{S_2})$

(ii) $(S \supseteq S_1 \cup S_2) \Rightarrow (\mathcal{P}_S \subseteq \mathcal{P}_{S_1} \cap \mathcal{P}_{S_2})$

(iii) $(S \subseteq S_1 \cap S_2) \Rightarrow (\mathcal{P}_S \supseteq \mathcal{P}_{S_1} \cup \mathcal{P}_{S_2})$

Proof. Observe first that (4) can be rewritten as

$$L_S = \bigcup_{s_1, s_2 \in S} \{xy \in \Sigma^* \mid w \sqcup\!\sqcup_{\varphi_{\mathrm{dn}}(s_1)} x \in L,\ w \sqcup\!\sqcup_{\varphi_{\mathrm{up}}(s_2)} y \in \theta(L),\ w \in \Sigma^+\}. \quad (5)$$

(i) For each $L \subseteq \Sigma^*$, $(S_1 \subseteq S_2) \Rightarrow (L_{S_1} \subseteq L_{S_2}) \Rightarrow (L_{S_2} \subseteq \{\lambda\} \Rightarrow L_{S_1} \subseteq \{\lambda\}) \Rightarrow (L \in \mathcal{P}_{S_2} \Rightarrow L \in \mathcal{P}_{S_1}) \Rightarrow (\mathcal{P}_{S_2} \subseteq \mathcal{P}_{S_1})$.

(ii) By (5), $(S \supseteq S_1 \cup S_2) \Rightarrow (L_S \supseteq L_{S_1} \cup L_{S_2}) \Rightarrow (L_S \subseteq \{\lambda\}$ implies $L_{S_1} \subseteq \{\lambda\}$ and $L_{S_2} \subseteq \{\lambda\}) \Rightarrow (L \in \mathcal{P}_S$ implies $L \in \mathcal{P}_{S_1}$ and $L \in \mathcal{P}_{S_2}) \Rightarrow (\mathcal{P}_S \subseteq \mathcal{P}_{S_1} \cap \mathcal{P}_{S_2})$.

(iii) By (i), $(S \subseteq S_1 \cap S_2) \Rightarrow (S \subseteq S_1$ and $S \subseteq S_2) \Rightarrow (\mathcal{P}_S \supseteq \mathcal{P}_{S_1}$ and $\mathcal{P}_S \supseteq \mathcal{P}_{S_2}) \Rightarrow (\mathcal{P}_S \supseteq \mathcal{P}_{S_1} \cup \mathcal{P}_{S_2})$.

□

Hence, for properties (X) and (Y) associated with the trajectory sets S_X and S_Y, $S_X \subseteq S_Y$ implies (Y) is stronger than (X). For example, if L is θ-compliant, then it is both θ-p-compliant and θ-s-compliant, as $S_C \cup S_D \subseteq S_B$.

The main reason for introducing Definition 4.1 is the characterization of bond-free properties via language inequations. This unified approach allows us to answer important questions regarding these properties, e.g. decidability and maximality questions, as shown below.

Theorem 4.4 *There exist fixed regular sets of trajectories $T_1, T_2 \subseteq V^*$ and regular languages $K_1, K_2 \subseteq (\Sigma \cup V)^*$ such that for a bond-free property $\mathcal{P}$ associated with sets of trajectories T_{dn} and T_{up}, $L \in \mathcal{P}$ holds for an $L \subseteq \Sigma^*$ iff*

$$((L \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}}) \sqcup\!\sqcup_{T_1} (\theta(L) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{T_2} K_1 \subseteq K_2. \tag{6}$$

Proof. Let

$$\begin{aligned} T_1 &= \{00, 11, 0101\}^*, \\ T_2 &= \{00, 1111\}^*, \\ K_1 &= (\bigcup_{a \in \Sigma} \{aa00\})^+, && (7) \\ K_2 &= (\Sigma \cup V)^* 0 (\Sigma \cup V)^* \cup \{\lambda\}. && (8) \end{aligned}$$

(i) Assume that $L \notin \mathcal{P}$ for a language $L \in \Sigma^*$. Then there is $xy, w \in \Sigma^+$ such that $w \sqcup\!\sqcup_{t_{\mathrm{dn}}} x \in L$, $w \sqcup\!\sqcup_{t_{\mathrm{up}}} y \in \theta(L)$ for some $t_{\mathrm{dn}} \in T_{\mathrm{dn}}$, $t_{\mathrm{up}} \in T_{\mathrm{up}}$. Denote

$$\begin{aligned} u_1 &= w \sqcup\!\sqcup_{t_{\mathrm{dn}}} x && (9) \\ u_2 &= w \sqcup\!\sqcup_{t_{\mathrm{up}}} y && (10) \end{aligned}$$

Let $w = a_1 \ldots a_m$, $x = b_1 \ldots b_n$, $y = c_1 \ldots c_p$, for $a_i, b_j, c_k \in \Sigma$, $m > 0$, $n + p > 0$. Then

$$\begin{aligned} u_1 \sqcup\!\sqcup_{\mathrm{bl}} t_{\mathrm{dn}} &= z_1 z_2 \ldots z_{m+n} \mid z_i = a_{j_i} 0 \text{ or } z_i = b_{k_i} 1, \\ &\quad 1 \le i \le m+n,\ j_i = |z_1 \ldots z_i|_0,\ k_i = |z_1 \ldots z_i|_1, && (11) \\ u_2 \sqcup\!\sqcup_{\mathrm{bl}} t_{\mathrm{up}} &= z_1 z_2 \ldots z_{m+p} \mid z_i = a_{j_i} 0 \text{ or } z_i = c_{k_i} 1, \\ &\quad 1 \le i \le m+p,\ j_i = |z_1 \ldots z_i|_0,\ k_i = |z_1 \ldots z_i|_1. && (12) \end{aligned}$$

Denote

$$L' = (L \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}}) \sqcup\!\sqcup_{T_1} (\theta(L) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}}).$$

As $u_1 \in L$ and $u_2 \in \theta(L)$, clearly

$$(u_1 \sqcup\!\sqcup_{\mathrm{bl}} t_{\mathrm{dn}}) \sqcup\!\sqcup_{T_1} (u_2 \sqcup\!\sqcup_{\mathrm{bl}} t_{\mathrm{up}}) \subseteq L'.$$

Then L' must contain a word v of the form

$$\begin{aligned} v \;=\;& z_1 z_2 \ldots z_{m+n+p} \mid z_i = a_{j_i} a_{j_i} 00 \text{ or } z_i = b_{k_i} 1 \text{ or } z_i = c_{l_i} 1, \\ & 1 \le i \le m+n+p,\ 1 \le j_i \le m,\ 1 \le k_i \le n,\ 1 \le l_i \le p. \end{aligned}$$

Consequently, $L' \rightsquigarrow_{T_2} K_1$ must contain a word z of the form

$$z = z_1 z_2 \ldots z_{n+p} \mid z_i = b_{k_i} 1 \text{ or } z_i = c_{l_i} 1,\ 1 \le i \le n+p,\ 1 \le k_i \le n,\ 1 \le l_i \le p.$$

As $n + p > 0$, $z \notin K_2$ and hence (6) does not hold.

(ii) Assume that (6) does not hold, then there is a nonempty word z in $L' \rightsquigarrow_{T_2} K_1$ containing no symbol 0. Notice that all the words in $L' \rightsquigarrow_{T_2} K_1$ are in $(\Sigma \cdot V \cup \Sigma^2 \cdot V^2)^*$. Denote $\Gamma = \Sigma \cdot 1 \cup \Sigma^2 \cdot 1^2$, then we have $z \in \Gamma^+$.

Consequently, there is a $v \in L'$ such that $z \in v \rightsquigarrow_{T_2} K_1$. As $\rightsquigarrow_{T_2}$ is the left inverse of $\sqcup\!\sqcup_{T_2}$, we have $v \in z \sqcup\!\sqcup_{T_2} K_1$, and hence

$$v \in \Gamma^+ \sqcup\!\sqcup_{T_2} K_1. \tag{13}$$

Furthermore, $v \in v_1 \sqcup\!\sqcup_{T_1} v_2$ for some $v_1 \in L \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}}$ and $v_2 \in \theta(L) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}}$. All the parts of v of the form $aa00$ (belonging to K_1 due to (13)) have to be produced from v_1, v_2 via the 0101 parts of a trajectory $t_1 \in T_1$. Then the symbols from Σ immediately preceding 0's in v_1 and v_2 must form two identical strings. In other words, v_1 and v_2 must adopt the form of the right-hand sides of (11) and (12), respectively.

Inevitably, there must exist $w, xy \in \Sigma^+$, $t_{\mathrm{dn}} \in T_{\mathrm{dn}}$, $t_{\mathrm{up}} \in T_{\mathrm{up}}$ such that (9), (10), (11), (12) all hold. It follows that $w \sqcup\!\sqcup_{t_{\mathrm{dn}}} x \in L$, $w \sqcup\!\sqcup_{t_{\mathrm{up}}} y \in \theta(L)$, $xy \neq \lambda$, $xy \in L_S$, and hence $L \notin \mathcal{P}$.

□

Corollary 4.5 *For each bond-free property $\mathcal{P}$ there is a binary word operation $\boxminus_{\mathcal{P}}$ such that $L \in \mathcal{P}$ holds for an $L \subseteq \Sigma^*$ iff*

$$L \boxminus_{\mathcal{P}} L \subseteq K_2.$$

Proof. By Theorem 4.4, we define

$$x \boxminus_{\mathcal{P}} y = ((x \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}}) \sqcup\!\sqcup_{T_1} (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{T_2} K_1. \tag{14}$$

□

The above characterization of bond-free language properties allows us to answer decidability questions "Does $L \in \mathcal{P}$ hold for a given language L and a property $\mathcal{P}$?" To measure the complexity of these decision problems, we need to introduce some further concepts of formal language theory first.

A nondeterministic finite automaton with λ productions (or transitions), a λ-*NFA* for short, is a quintuple $A = (S, \Sigma, s_0, F, P)$ such that S is the finite and nonempty set of states, s_0 is the start state, F is the set of final states, and P is the set of productions of the form $sx \to t$, where s and t are states in S, and x is either a symbol in Σ or the empty word. If there is no production with $x = \lambda$, the automaton is called an *NFA*. If for every two productions of the form $sx_1 \to t_1$ and $sx_2 \to t_2$ of an NFA we have that $x_1 \neq x_2$ then the automaton is called a *DFA* (deterministic finite automaton). The language accepted by the automaton A is denoted by $L(A)$.

Let A_1, A_2 be NFA's and let θ be an involution. Then there are NFA's of the size $\mathcal{O}(|A_1| \cdot |A_2|)$ accepting the languages $L(A_1) \cup L(A_2)$ and $L(A_1) \cap L(A_2)$. The language $\theta(L(A_1))$ can be accepted by an NFA of the size $|A_1|$. Similarly, if A_1, A_2 are DFA's, then there are DFA's of the size $\mathcal{O}(|A_1| \cdot |A_2|)$ accepting the languages $L(A_1) \cup L(A_2)$ and $L(A_1) \cap L(A_2)$. The language $L(A_1)^c$ can be accepted by a DFA of the size $|A_1|$. The *size* $|A|$ of the automaton A is the number $|S| + |P|$. We refer the reader to [21] or [22] for further details on automata and formal languages. The following lemma follows by [4, 15, 18].

Lemma 4.6 *Let L_1, L_2 and T be regular languages accepted by the NFA's A_1, A_2 and A_T, respectively.*

(i) There exists an NFA A accepting $L_1 \sqcup\!\sqcup_T L_2$ of the size $|A| = \mathcal{O}(|A_1| \cdot |A_2| \cdot |A_T|)$, constructible in time $|A|$.

(ii) There exists a λ-NFA A' accepting $L_1 \leadsto_T L_2$ of the size $|A'| = \mathcal{O}(|A_1| \cdot |A_2| \cdot |A_T|)$, constructible in time $|A'|$.

Theorem 4.7 *Let $\mathcal{P}$ be a bond-free property with regular sets of trajectories $T_{\rm dn}, T_{\rm up}$. Then the following problem is decidable in quadratic time:*
Input: an NFA A.
Output: Y/N depending on whether $L(A) \in \mathcal{P}$ holds.

Proof. By (14) and repeated applications of Lemmata 3.9, 3.10, 4.6 we can construct a λ-NFA A' accepting $L(A) \boxminus_{\mathcal{P}} L(A)$ in time $\mathcal{O}(|A|^2)$, as $T_{\rm dn}$, $T_{\rm up}$ and K_1 are fixed. Then due to Corollary 4.5 it is enough to test the emptiness of the language $L(A') \cap K_2^c$. As K_2 is fixed and the size of A' is $\mathcal{O}(|A|^2)$, this requires also $\mathcal{O}(|A|^2)$ steps. □

The decidability problems of some DNA properties was studied in [7], where the decidability of the properties (D) and (F) was shown. In [5] the decidability of (M) in quadratic time is proven. In [8] an algorithm deciding (F) in quadratic time for finite sets of codewords is presented. The following corollary generalizes all these previous results into a uniform quadratic-time decidability procedure for all regular sets of codewords.

Corollary 4.8 *The following problem is decidable in quadratic time w.r.t. $|A|$:*
Input: an NFA A.
Output: Y/N depending on whether $L(A)$ satisfies any of the properties (B), (C), (D), (G), (H), (I), (J) (M).

It is known that for some bond-free properties there is no algorithm which would decide whether a given context-free language L satisfies the property. The corresponding statement has been proven in [7] for the case of the properties (B) and (F), where the alphabet is *fixed* and equal to $\{A, C, G, T\}$.

Corollary 4.9 *The following problem is undecidable.*

Input: *A bond-free property $\mathcal{P}$ associated with regular sets of trajectories $T_{\rm dn}$, $T_{\rm up}$, and a context-free language L.*

Output: *Y/N depending on whether $L \in \mathcal{P}$.*

5 Maximal Bond-Free Languages

In the previous section we introduced the characterization of bond-free properties via language inequations. Now we show that this approach may be applied also to maximality problems ("Is $L \subseteq M$ maximal w.r.t. a bond-free property $\mathcal{P}$?") and to the construction of maximal languages satisfying a given property ("For a given property $\mathcal{P}$ and a set M, find a maximal $L \subseteq M$ such that $L \in \mathcal{P}$ holds."). The set $M \subseteq \Sigma^*$ represents the library of all applicable/constructible DNA strands in a case at hand.

To study these topics in detail, first some more technical results are needed. The following notion of maximal solutions to language inequations and of residue of the solution appears in [13]. Let $L, M \subseteq \Sigma^*$ be two languages and let $\diamondsuit$ be a binary word operation. Consider an inequation of the form

$$X \diamondsuit L \subseteq X^c, \quad X \subseteq M. \tag{15}$$

The language $S_{\max}$ is a *maximal solution* of (15) if $S_{\max}$ is a solution (i.e. (15) holds true for $X = S_{\max}$), and for each $x \in M - S_{\max}$, $S_{\max} \cup \{x\}$ is not a solution.

Let S be a solution of (15). We call the language

$$R = M - (S \cup S \diamondsuit L \cup S \diamondsuit^l L)$$

the *residue* of S. The following theorem is a refinement of Proposition 6.2 in [13]. In the proof we use the fact that S is a solution of (15) if and only if it is a solution of $X \diamondsuit^l L \subseteq X^c$, for $X \subseteq M$.

Theorem 5.1 *Let S be a solution of (15), let R be the residue of S, and let $Q = \{z \in \Sigma^* \mid z \in z \diamondsuit L\}$. Then S is maximal iff $R - Q = \emptyset$.*

Proof.

"$\Rightarrow$" Suppose that S is maximal but there exists a word $z \in R - Q$. Let $T = S \cup \{z\}$. We show that T is a solution of (15), that is, $t \diamondsuit L \subseteq T^c$ for all $t \in T$. As $z \in R - Q$, we have that $z \in M$ and $z \notin S$, $z \notin S \diamondsuit L$, $z \notin S \diamondsuit^l L$, $z \notin z \diamondsuit L$.

- If $t \in S$, then $z \notin t \diamondsuit L$ due to the above, and furthermore $t \diamondsuit L \subseteq S^c$, as S is a solution of (15). Hence, $t \diamondsuit L \subseteq T^c = S^c - \{z\}$.
- If $t = z$, then again $z \notin z \diamondsuit L$ and $z \notin S \diamondsuit^l L$ due to the above, hence $t \diamondsuit L \subseteq T^c$.

Thus, $T \diamondsuit L \subseteq T^c$, hence T is a solution of (15) strictly containing S, a contradiction.

"$\Leftarrow$" Suppose that $R - Q = \emptyset$ but S is not maximal, i.e. $T = S \cup \{z\}$ is a solution of (15) for some $z \in M - S$. Then, $z \diamondsuit L \subseteq T^c$, $S \diamondsuit L \subseteq T^c$ and $S \diamondsuit^l L \subseteq T^c$. This implies $z \notin z \diamondsuit L$ (and hence $z \notin Q$), $z \notin S \diamondsuit L$ and $z \notin S \diamondsuit^l L$. But as $z \in M - S$, we have $z \in M - (S \cup S \diamondsuit L \cup S \diamondsuit^l L) - Q$, hence $z \in R - Q$, a contradiction again.

□

The following result from [13] explains the connection of inequation (15) with the maximality of bond-free languages.

Lemma 5.2 *The inequation* $X \boxminus_{\mathcal{P}} X \subseteq K_2$ *with* $X \subseteq M \subseteq \Sigma^+$ *is equivalent to* $X \boxminus^r_{\mathcal{P}} K_2^c \subseteq X^c$ *with* $X \subseteq M \subseteq \Sigma^+$.

Theorem 5.3 *Let* $\mathcal{P}$ *be a bond-free property and* $M \subseteq \Sigma^+$ *a set of words. For a language* $L \subseteq M$ *such that* $L \in \mathcal{P}$, *denote*

$$\begin{aligned} R &= M - (L \cup L \boxminus^r_{\mathcal{P}} K_2^c \cup K_2^c \boxminus^l_{\mathcal{P}} L), && (16)\\ Q &= \{z \in \Sigma^* \mid z \boxminus_{\mathcal{P}} z \cap K_2^c \neq \emptyset\}, && (17) \end{aligned}$$

where $\boxminus_{\mathcal{P}}$ *is defined by (14) and* K_2 *by (8). Then* L *is a maximal subset of* M *satisfying* $\mathcal{P}$ *iff* $R - Q = \emptyset$.

Proof. Follows by Definition 3.1, Corollary 4.5, Theorem 5.1 and Lemma 5.2. □

To construct an algorithm deciding the maximality of bond-free DNA languages due to the above theorem, we need to calculate the inverses of the operation $\boxminus_{\mathcal{P}}$.

Lemma 5.4 *Let* $\boxminus_{\mathcal{P}}$ *be a word operation defined by (14). Then*

(i) $z \boxminus^l_{\mathcal{P}} y = ((z \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{T_1} (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{\mathrm{bl}} T_{\mathrm{dn}}$,

(ii) $x \boxminus^r_{\mathcal{P}} z = \theta(((z \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{\widetilde{T_1}} (x \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}})) \rightsquigarrow_{\mathrm{bl}} T_{\mathrm{up}})$.

Proof. All the following manipulations are based on repeated application of Definitions 3.1, 3.2 and Lemmata 3.3, 3.8.

$$\begin{array}{llll}
\text{(i)} & z \in x \boxminus_{\mathcal{P}} y \text{ iff} & & \\
& z \in ((x \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{dn}}) \sqcup\!\sqcup_{T_1} (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{T_2} K_1 & \text{iff} \\
& z \in (x \; (\sqcup\!\sqcup_{\mathrm{bl}} ; \sqcup\!\sqcup_{T_1}) \; (T_{\mathrm{dn}} ; \theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{T_2} K_1 & \text{iff} \\
& z \in (x \; (\rightsquigarrow_{T_1} ; \rightsquigarrow_{\mathrm{bl}})^l \; (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}} ; T_{\mathrm{dn}})) \rightsquigarrow_{T_2} K_1 & \text{iff} \\
& z \in x \; ((\rightsquigarrow_{T_1} ; \rightsquigarrow_{\mathrm{bl}})^l ; \rightsquigarrow_{T_2}) \; ((\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}} ; T_{\mathrm{dn}}) ; K_1) & \text{iff} \\
& z \in x \; (\sqcup\!\sqcup_{T_2} ;(\rightsquigarrow_{T_1} ; \rightsquigarrow_{\mathrm{bl}}))^l \; (K_1 ;(\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}} ; T_{\mathrm{dn}})) & \text{iff} \\
& x \in z \; (\sqcup\!\sqcup_{T_2} ;(\rightsquigarrow_{T_1} ; \rightsquigarrow_{\mathrm{bl}})) \; (K_1 ;(\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}} ; T_{\mathrm{dn}})) & \text{iff} \\
& x \in (z \sqcup\!\sqcup_{T_2} K_1) \; (\rightsquigarrow_{T_1} ; \rightsquigarrow_{\mathrm{bl}}) \; (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}} ; T_{\mathrm{dn}}) & \text{iff} \\
& x \in ((z \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{T_1} (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} T_{\mathrm{up}})) \rightsquigarrow_{\mathrm{bl}} T_{\mathrm{dn}} & \text{iff} \\
& x \in z \boxminus^l_{\mathcal{P}} y &
\end{array}$$

$$
\begin{array}{llrcll}
\text{(ii)} & & z & \in & x \boxminus_{\mathcal{P}} y \text{ iff} & \\
& & z & \in & ((x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \sqcup\!\sqcup_{T_1} (\theta(y) \sqcup\!\sqcup_{\text{bl}} T_{\text{up}})) \rightsquigarrow_{T_2} K_1 & \text{iff} \\
& & z & \in & K_1 \rightsquigarrow'_{T_2} ((x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \sqcup\!\sqcup_{T_1} (\theta(y) \sqcup\!\sqcup_{\text{bl}} T_{\text{up}})) & \text{iff} \\
& & z & \in & (K_1 : x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \ (\rightsquigarrow'_{T_2} : \sqcup\!\sqcup_{T_1}) \ (T_{\text{up}} \sqcup\!\sqcup'_{\text{bl}} \theta(y)) & \text{iff} \\
& & z & \in & ((K_1 : x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) : T_{\text{up}}) \ ((\rightsquigarrow'_{T_2} : \sqcup\!\sqcup_{T_1}) : \sqcup\!\sqcup'_{\text{bl}}) \ \theta(y) & \text{iff} \\
& & \theta(y) & \in & ((K_1 : x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) : T_{\text{up}}) \ ((\rightsquigarrow'_{T_2} : \sqcup\!\sqcup_{T_1}) : \sqcup\!\sqcup'_{\text{bl}})^r \ z & \text{iff} \\
& & \theta(y) & \in & (T_{\text{up}} : (K_1 : x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}})) \ (\sqcup\!\sqcup'^r_{\text{bl}} : (\rightsquigarrow'_{T_2} : \sqcup\!\sqcup_{T_1})^r) \ z & \text{iff} \\
& & \theta(y) & \in & T_{\text{up}} \sqcup\!\sqcup'^r_{\text{bl}} ((K_1 : x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \ (\rightsquigarrow'_{T_2} : \sqcup\!\sqcup_{T_1})^r \ z) & \text{iff} \\
& & \theta(y) & \in & T_{\text{up}} \sqcup\!\sqcup^{l\prime}_{\text{bl}} ((x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}} : K_1) \ (\sqcup\!\sqcup^r_{T_1} : \rightsquigarrow^{l\prime}_{T_2}) \ z) & \text{iff} \\
& & \theta(y) & \in & ((x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \sqcup\!\sqcup^r_{T_1} (K_1 \rightsquigarrow^{l\prime}_{T_2} z)) \sqcup\!\sqcup^l_{\text{bl}} T_{\text{up}} & \text{iff} \\
& & \theta(y) & \in & ((x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}}) \sqcup\!\sqcup^r_{T_1} (z \sqcup\!\sqcup_{T_2} K_1)) \rightsquigarrow_{\text{bl}} T_{\text{up}} & \text{iff} \\
& & y & \in & \theta(((z \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{\widetilde{T}_1} (x \sqcup\!\sqcup_{\text{bl}} T_{\text{dn}})) \rightsquigarrow_{\text{bl}} T_{\text{up}}) & \text{iff} \\
& & y & \in & x \boxminus^r_{\mathcal{P}} z &
\end{array}
$$

□

Theorem 5.5 *Let θ be an antimorphism and let $\mathcal{P}$ be one of the properties (B), (C), (D), (G). Let $M \subseteq \Sigma^+$ be a regular set of words, and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.*

Proof. By Theorem 5.3 it suffices to test whether $R - Q = \emptyset$, where R and Q are given by (16) and (17), respectively. Recall that M, L and K_2 are regular languages and $\boxminus_{\mathcal{P}}$ is defined by (14). By Lemmata 3.9, 3.10, 5.4 we can construct an NFA accepting R.

By (17) and Corollary 4.5, Q is the set of all words z such that the language $\{z\}$ does not satisfy the studied property — one of (B), (C), (D), (G). By definition of these properties in Section 2, one can observe that for (B), (C), (D) we have $Q = \emptyset$. In the case of (G), $Q = \bigcup_{a \in \Sigma} (a \Sigma^* \theta(a))$. In all these cases Q is a regular language and hence the question "$R - Q = \emptyset$?" is effectively decidable. □

Theorem 5.6 *Let θ be a morphism and let $\mathcal{P}$ be one of the properties (B), (C), (D), (H), (I). Let $M \subseteq \Sigma^+$ be a regular set of words, and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.*

Proof. As in the above proof, the statement can be reduced to deciding whether $R - Q = \emptyset$, where R is a regular language. For the properties (B), (C), (D) we have again $Q = \emptyset$. Denote $\Gamma = \{a \in \Sigma \mid a = \theta(a)\}$, then in the case of (H) we have $Q = \Gamma\Sigma^+$ and in the case of (I), $Q = \Sigma^+\Gamma$. In all these cases Q is also regular and the problem is decidable. □

Let A be a DFA accepting L. The procedure described in the two above proofs involves applying operations on A that result in an NFA and then taking the complement of that NFA. This process may require an exponential number of steps w.r.t. $|A|$ in the worst case. However, as the following theorem shows, we can obtain a polynomial-time algorithm at least for finite languages.

Lemma 5.7 *Let L and M be two languages such that $L \subseteq M$ and L satisfies property (B). Let R be the set defined in Theorem 5.3. Then,*

$$R = M - (L \cup \Sigma^*\theta(L)\Sigma^+ \cup \Sigma^+\theta(L)\Sigma^* \cup \mathrm{Sub}'(\theta(L))).$$

where $\mathrm{Sub}'(L)$ *is the set of all proper subwords of L.*

Proof. As L satisfies property (B), Corollary 4.5 implies that L is a solution of $L \boxminus_{\mathcal{P}} L \subseteq K_2$, where $x \boxminus_{\mathcal{P}} y$ is defined in (14). Moreover, by the definition of S_B, it follows that $T_{\mathrm{up}} = 1^*0^+1^*$ and $T_{\mathrm{dn}} = 0^+$. By Lemma 5.4 we have that

$$\begin{aligned} K_2^c \boxminus_{\mathcal{P}}^l y &= ((K_2^c \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{T_1} (\theta(y) \sqcup\!\sqcup_{\mathrm{bl}} 1^*0^+1^*)) \rightsquigarrow_{\mathrm{bl}} 0^+ \\ \theta(x \boxminus_{\mathcal{P}}^r K_2^c) &= ((K_2^c \sqcup\!\sqcup_{T_2} K_1) \rightsquigarrow_{\widetilde{T}_1} (x \sqcup\!\sqcup_{\mathrm{bl}} 0^+)) \rightsquigarrow_{\mathrm{bl}} 1^*0^+1^*, \end{aligned}$$

where K_1, T_1 and T_2 are defined in Theorem 4.4. As the language R is a subset of Σ^*, we restrict our attention to the sets $(L \boxminus_{\mathcal{P}}^r K_2^c) \cap \Sigma^*$ and $(K_2^c \boxminus_{\mathcal{P}}^l L) \cap \Sigma^*$. In particular we show next that $\theta(x \boxminus_{\mathcal{P}}^r K_2^c) \cap \Sigma^* = \Sigma^* x \Sigma^+ \cup \Sigma^+ x \Sigma^*$ and $(K_2^c \boxminus_{\mathcal{P}}^l y) \cap \Sigma^* = \mathrm{Sub}'(\theta(y))$.

Let f be a word in $\theta(x \boxminus_{\mathcal{P}}^r K_2^c) \cap \Sigma^*$. First note that $K_2^c \sqcup\!\sqcup_{T_2} K_1$ consists of words w of the form $w_0 v_1 w_1 \cdots v_n w_n$, where each w_i is either empty or in $(\Sigma \cup \{1\})^2$, and each v_j is either empty or of the form $b_j b_j 00$, with $b_j \in \Sigma$. Also, the set $x \sqcup\!\sqcup_{\mathrm{bl}} 0^+$ is equal to $\{u\}$, with u being of the form $a_1 0 \cdots a_m 0$, where the a_i's are the symbols of x. Then $f \in z_2 \rightsquigarrow_{\mathrm{bl}} 1^*0^+1^*$, where z_2 is a word in $w \rightsquigarrow_t u$ with $t \in \widetilde{T}_1$. Thus z_2 must be of the form

$$x_1 1 \cdots x_{i-1} 1 x_i 0 \cdots x_{i+j-1} 0 x_{i+j} 1 \cdots x_k 1$$

with $k \geq 1, 1 \leq i \leq k, j \geq 1$ and each x_l being in Σ. One can verify that for every nonempty subword w_i of w the corresponding subword of the trajectory t must be 00 and, for every nonempty subword v_j of w, the corresponding subword of t must be 1010. Hence, $j = m$ and $x_{i+r} = a_{r+1}$ for all $r = 0, \ldots, m-1$. Moreover, as λ is not in K_2^c, at least one subword w_i of w is nonempty. Hence, z_2 is of the form $u_1 a_1 0 \cdots a_m 0 u_2$ for some words $u_1, u_2 \in \Sigma \cup \{1\}$ with $u_1 u_2 \neq \lambda$, and therefore f must be in $\Sigma^* x \Sigma^+ \cup \Sigma^+ x \Sigma^*$.

Now observe that $x \boxminus_{\mathcal{P}}^r K_2^c$ is equal to $\Sigma_1^+ \longleftarrow \theta(x)$, where $\longleftarrow$ is the insertion operation and $\Sigma_1 = \Sigma \cup \{1\}$. One verifies that $x \in y \boxminus_{\mathcal{P}}^{rl} K_2^c$ if and only if $x \in \theta(y) \rightleftharpoons \Sigma_1^+$ and, therefore, as $\boxminus_{\mathcal{P}}^{rl} = \boxminus_{\mathcal{P}}^{l\prime}$, $(K_2^c \boxminus_{\mathcal{P}}^l y) = \theta(y) \rightleftharpoons \Sigma_1^+$. Moreover, as $y \in \Sigma^*$, we have that $\theta(y) \rightleftharpoons \Sigma_1^+ = \theta(y) \rightleftharpoons \Sigma_1^+ \cap \Sigma^*$, which is equal to $\mathrm{Sub}'(\theta(y))$. □

Theorem 5.8 *The following problem is decidable in time $O(\|L\|^3|A|)$, where $\|L\|$ is the quantity $\sum_{w \in L} |w|$.*

Input: DFA A and a finite language L such that $L \subseteq L(A)$ and L satisfies property (B).

Output: Y/N, depending on whether L is a maximal subset of $L(A)$ satisfying (B).

Proof. Let $M = L(A)$ and let $R_1 = L \cup \Sigma^*\theta(L)\Sigma^+ \cup \Sigma^+\theta(L)\Sigma^* \cup \mathrm{Sub}'(\theta(L))$. By Theorem 5.3 and Lemma 5.7, to decide whether L is maximal, it is sufficient to test whether the language $M \cap R_1^c$ is empty – recall from the previous theorem that the set Q for property (B) is empty. Let $R_2 = L \cup \mathrm{Sub}(\theta(L)) \cup \Sigma^*\theta(L)\Sigma^*$. One verifies that $R_2 = R_1 \cup \theta(L)$, which is equivalent to $R_2^c = R_1^c \cap \theta(L)^c$. This implies that $R_1^c = (R_1^c \cap \theta(L)) \cup R_2^c$. Hence, $M \cap R_1^c$ is empty if and only if $M \cap R_2^c$ is empty and $M \cap R_1^c \cap \theta(L)$ is empty.

The algorithm consists of the following steps.

1 Construct DFAs A_0, A_1 and A_2 accepting the languages $L, \mathrm{Sub}(\theta(L))$ and $\Sigma^*\theta(L)\Sigma^*$, respectively.

2 Use a product construction on A_0, A_1 and A_2 to obtain a complete DFA A_3 accepting R_2, and then consider the DFA A_3^c accepting R_2^c.

3 Use a product construction on A_3^c and A to obtain a DFA A_4 accepting the language $M \cap R_2^c$.

4 Output No and quit, if there is a path from the start state to a final state of A_4.

5 For each $w \in L$, if $\theta(w)$ is in M and $\theta(w)$ is not in L and $\theta(w)$ is not in $\mathrm{Sub}'(\theta(L))$ and $\theta(w)$ is not in $\Sigma^*\theta(L)\Sigma^+ \cup \Sigma^+\theta(L)\Sigma^*$, output No and quit.

6 Output Yes.

The DFA A_0 is the trie corresponding to L and can be constructed in time $O(\|L\|)$ [3]. The DFA A_1 is the factor automaton accepting $\mathrm{Sub}(\theta(L))$ and can be constructed in time $O(\|L\|)$ [3]. The DFA A_2 can be constructed from L in time $O(\|L\|)$ as well, by modifying the construction of the dictionary matching DFA accepting $\Sigma^*\theta(L)$ [13]. It follows now that $|A_3^c| = |A_3| = O(\|L\|^3)$ and $|A_4| = O(\|L\|^3|A|)$. The fourth step of the algorithm requires time proportional to the size of A_4.

For the fifth step, we note the following. For each word w in L, testing whether $\theta(w)$ is not in L can be done in time $O(\|L\|)$, and testing whether $\theta(w)$ is in M can be done in time $O(|w|)$ by running the DFA A on input $\theta(w)$. Hence, for all w, these two tests require time $O(\|L\|^2)$. Now we need to test, for each $w \in L$, whether $\theta(w)$ is not in $\mathrm{Sub}'(\theta(L))$ and $\theta(w)$ is not in $\Sigma^*\theta(L)\Sigma^+ \cup \Sigma^+\theta(L)\Sigma^*$. This is equivalent to testing, for each word $u \in L$ with $u \neq w$, whether the condition ($\theta(w)$ is not a subword of $\theta(u)$ AND $\theta(u)$ is not a subword of $\theta(w)$) is true. The question of whether a word x is a subword of a word y is a pattern matching problem and can be solved in time $O(|x| + |y|)$ [3]. Hence, the overall time for the remaining tests is $O(\|L\|^2)$. □

The language inequation approach can be used also for construction of bond-free DNA languages. The following result is a direct consequence of Proposition 6.3 in [13].

Theorem 5.9 *Let $\mathcal{P}$ be a bond-free property and $M \subseteq \Sigma^+$ a set of words. Let $L \subseteq M$ be a language such that $L \in \mathcal{P}$. Denote*

$$\begin{aligned} L_1 &= M \cap (K_2^c \boxminus_{\mathcal{P}}^l L)^c \cap ((K_2^c \boxminus_{\mathcal{P}}^l L)^c \boxminus_{\mathcal{P}}^r K_2^c)^c, \\ L_2 &= M \cap (L \boxminus_{\mathcal{P}}^r K_2^c)^c \cap (K_2^c \boxminus_{\mathcal{P}}^l (L \boxminus_{\mathcal{P}}^r K_2^c)^c)^c. \end{aligned}$$

Then $L \subseteq L_i \subseteq M$ and $L_i \in \mathcal{P}$, for $i = 1, 2$.

Given a language L satisfying a certain bond-free property, the above theorem allows us to construct "larger" languages satisfying the same property and containing L. However, if A is the NFA accepting L, the procedure may require an exponential number of steps w.r.t. $|A|$ in the worst case.

6 Strictly Bond-Free Languages

In this section we focus mostly on the strict versions of the DNA language properties (B)–(L), i.e. their conjunctions with (A). As we already mentioned in Section 2, the property (E) is equal to strictly (B), hence we do not refer to (E) in the sequel. The motivation for the following general concept of a *strictly bond-free* property is the fact the strict versions of the above properties are its special cases. The property (A) itself is a special case of strictly bond-freedom. For some choices of the operations $\diamondsuit_{\mathrm{dn}}$, $\diamondsuit_{\mathrm{up}}$ below, however, the (A) property need not necessarily hold. This is verified by the fact that (non-strictly) (L) is also a special case of the strictly bond-free property.

Definition 6.1 *A class $\mathcal{P}$ of languages over Σ is called the* strictly bond-free property of degree 2 *if there are binary word operations $\diamondsuit_{\mathrm{dn}}$, $\diamondsuit_{\mathrm{up}}$ and an involution θ such that for an arbitrary $L \subseteq \Sigma^*$, $L \in \mathcal{P}$ holds iff*

$$\forall w, x, y \in \Sigma^* \ (w \diamondsuit_{\mathrm{dn}} x \cap L \neq \emptyset,\ w \diamondsuit_{\mathrm{up}} y \cap \theta(L) \neq \emptyset) \Rightarrow w = \lambda. \tag{18}$$

The formulation *strictly bond-free property of degree 2* is used to stress the fact that the property describes bonds of *two* single DNA strands. In the remainder of this paper we write simply *strictly bond-free property* for the strictly bond-free property of degree two.

Theorem 6.2 *The language properties (A), strictly (B)–(D), strictly (G)–(I), (L), strictly (L) are strictly bond-free properties.*

Proof. Let $\diamondsuit_{\mathrm{dn}} = ⧢_{T_{\mathrm{dn}}}$ and $\diamondsuit_{\mathrm{up}} = ⧢_{T_{\mathrm{up}}}$, where T_{dn} and T_{up} are the sets of trajectories used in the proof of Theorem 4.2. For the properties not studied in Theorem 4.2 we define the sets of trajectories T_{dn}, T_{up} as follows.

(A)	$T_{\mathrm{dn}} = T_{\mathrm{up}} = 0^+$,
(L)	$T_{\mathrm{dn}} = T_{\mathrm{up}} = 1^*0^k1^*$,
strictly (L)	$T_{\mathrm{dn}} = T_{\mathrm{up}} = 1^*0^k1^* \cup 0^+$.

Hence for the above mentioned properties, (18) takes the form

$$\forall w, x, y \in \Sigma^* \ (w ⧢_{T_{\mathrm{dn}}} x \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset) \Rightarrow w = \lambda. \tag{19}$$

Consider first the properties strictly (A), strictly (B)–(D), strictly (G)–(I) for which we have $x ⧢_{T_{\mathrm{dn}}} \lambda = x ⧢_{T_{\mathrm{up}}} \lambda = x$ for each $x \in \Sigma^+$. Then (19) is equivalent to

$$\begin{array}{ll}
\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ \neg(w ⧢_{T_{\mathrm{dn}}} x \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset) & \text{iff} \\
\forall w \in \Sigma^+,\ (\neg(w ⧢_{T_{\mathrm{dn}}} \lambda \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} \lambda \cap \theta(L) \neq \emptyset) & \\
\quad \wedge\ \forall x, y \in \Sigma^* \text{ with } xy \neq \lambda,\ \neg(w ⧢_{T_{\mathrm{dn}}} x \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset)) & \text{iff} \\
\forall w \in \Sigma^+,\ \neg(w \in L, w \in \theta(L)) & \\
\quad \wedge\ \forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ \neg(w ⧢_{T_{\mathrm{dn}}} x \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset,\ xy \neq \lambda) & \text{iff} \\
L \cap \theta(L) = \emptyset & \\
\quad \wedge\ \forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ (w ⧢_{T_{\mathrm{dn}}} x \cap L \neq \emptyset,\ w ⧢_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset) \Rightarrow xy \neq \lambda, &
\end{array}$$

which is further equivalent to strictly bond freedom by Definitions 4.1, 6.1 and Theorem 4.2. The proof for (L) and strictly (L) is similar except that in the case of (L) we do not obtain the condition $L \cap \theta(L) = \emptyset$. $\square$

Our interest now will be to express the strictly bond-free language properties via language inequations as in Section 4.

Theorem 6.3 *Let $\mathcal{P}$ be a strictly bond-free property associated with operations $\diamondsuit_{\rm dn}, \diamondsuit_{\rm up}$. For a language $L \subseteq \Sigma^*$, $L \in \mathcal{P}$ holds iff*

$$(L \diamondsuit_{\rm dn}^{l} \Sigma^*) \rightsquigarrow_{1+} (\theta(L) \diamondsuit_{\rm up}^{l} \Sigma^*) = \emptyset. \tag{20}$$

Proof. Recall that due to Definition 6.1, $L \in \mathcal{P}$ holds iff

$$\begin{array}{ll}
\forall w, x, y \in \Sigma^*,\ (w \diamondsuit_{\rm dn} x \cap L \neq \emptyset,\ w \diamondsuit_{\rm up} y \cap \theta(L) \neq \emptyset) \Rightarrow w = \lambda & \text{iff} \\
\forall w, x, y \in \Sigma^*,\ (\exists \alpha \in L, \alpha \in w \diamondsuit_{\rm dn} x,\ \exists \beta \in \theta(L), \beta \in w \diamondsuit_{\rm up} y) \Rightarrow w = \lambda & \text{iff} \\
\forall w, x, y \in \Sigma^*,\ (\exists \alpha \in L, w \in \alpha \diamondsuit_{\rm dn}^{l} x,\ \exists \beta \in \theta(L), w \in \beta \diamondsuit_{\rm up}^{l} y) \Rightarrow w = \lambda & \text{iff} \\
\forall w, x, y \in \Sigma^*,\ w \in L \diamondsuit_{\rm dn}^{l} x,\ w \in \theta(L) \diamondsuit_{\rm up}^{l} y \Rightarrow w = \lambda & \text{iff} \\
\forall w \in \Sigma^*,\ w \in L \diamondsuit_{\rm dn}^{l} \Sigma^*,\ w \in \theta(L) \diamondsuit_{\rm up}^{l} \Sigma^* \Rightarrow w = \lambda & \text{iff} \\
(L \diamondsuit_{\rm dn}^{l} \Sigma^*) \cap (\theta(L) \diamondsuit_{\rm up}^{l} \Sigma^*) \subseteq \{\lambda\} & \text{iff} \\
(L \diamondsuit_{\rm dn}^{l} \Sigma^*) \rightsquigarrow_{1+} (\theta(L) \diamondsuit_{\rm up}^{l} \Sigma^*) = \emptyset. &
\end{array}$$

□

As in Section 4, we present a general result about effective decidability of the strictly bond-free properties for a given regular language L.

Theorem 6.4 *Let $\mathcal{P}$ be a strictly bond-free property associated with operations $\diamondsuit_{\rm dn} = \sqcup\!\sqcup_{T_{\rm dn}}$, $\diamondsuit_{\rm up} = \sqcup\!\sqcup_{T_{\rm up}}$, with regular sets of trajectories $T_{\rm dn}, T_{\rm up}$. Then the following problem is decidable in quadratic time:*

Input: an NFA A.

Output: Y/N depending on whether $L(A)$ is in $\mathcal{P}$.

Proof. Due to Theorems 4.2, 6.2, 6.3, a described strictly bond-free property can be expressed in the form

$$L' = (L(A) \sqcup\!\sqcup_{T_{\rm dn}}^{l} \Sigma^*) \rightsquigarrow_{1+} (\theta(L(A)) \sqcup\!\sqcup_{T_{\rm up}}^{l} \Sigma^*) = \emptyset.$$

Then by Lemmata 3.8, 3.10, 4.6, the λ-NFA A' accepting L' can be constructed in time $\mathcal{O}(|A|^2)$, and so is its size. Hence, testing whether $L(A') = \emptyset$ or not is limited by the same time bound. □

Corollary 6.5 *Let $\mathcal{P}$ be any of the properties (A), strictly (B) – strictly (D), strictly (G) – strictly (J), (L), strictly (L). The following problem is decidable in quadratic time w.r.t. $|A|$:*

Input: an NFA A.

Output: Y/N depending on whether $L(A)$ is in $\mathcal{P}$.

On the other hand, at least for some strictly bond-free properties there is no algorithm to decide whether a given context-free language satisfies the property. We demonstrate this fact for the θ-non-overlapping property (A).

Theorem 6.6 *For a given context-free language L it is undecidable whether L is θ-non-overlapping or not.*

Proof. Let L_1, L_2 be two context-free languages. Let $\#$, $\theta(\#)$ be symbols not in Σ, then $L = L_1\#\theta(\#)\theta(L_2)$ is also a context-free language. Clearly, $L \cap \theta(L) = \emptyset$ iff $L_1 \cap L_2 = \emptyset$ which is undecidable. □

Corollary 6.7 *The following problem is undecidable.*
Input: A strictly bond-free property $\mathcal{P}$ associated with regular sets of trajectories $T_{\rm dn}$, $T_{\rm up}$, and a context-free language L.
Output: Y/N depending on whether $L \in \mathcal{P}$.

7 Maximal Strictly Bond-Free Languages

In the sequel we concentrate on maximality problems ("Is $L \subseteq \Sigma^*$ maximal with respect to a strictly bond-free property $\mathcal{P}$?"). For this purpose, we reformulate Theorem 6.3 as follows.

Theorem 7.1 *For each strictly bond-free property $\mathcal{P}$ there is a binary word operation $\boxdot_{\mathcal{P}}$ such that for a language $L \subseteq \Sigma^*$, $L \in \mathcal{P}$ holds iff*

$$L \boxdot_{\mathcal{P}} L = \emptyset. \tag{21}$$

Proof. By Theorem 6.3, for $x, y \in \Sigma^*$ we have

$$x \boxdot_{\mathcal{P}} y = (x \diamondsuit^l_{\rm dn} \Sigma^*) \rightsquigarrow_{1+} (\theta(y) \diamondsuit^l_{\rm up} \Sigma^*). \tag{22}$$

□

We apply the techniques from Section 5 for studying maximal bond free languages. It follows that as a first step we need to calculate the left and right inverse of the operation $\boxdot_{\mathcal{P}}$.

Lemma 7.2 *Let $\boxdot_{\mathcal{P}}$ be a word operation defined by (22). Then*

(i) $z \boxdot^l_{\mathcal{P}} y = (z \sqcup\!\sqcup_{1+} (\theta(y) \diamondsuit^l_{\rm up} \Sigma^*)) \diamondsuit_{\rm dn} \Sigma^*$,

(ii) $x \boxdot^r_{\mathcal{P}} z = \theta(((x \diamondsuit^l_{\rm dn} \Sigma^*) \rightsquigarrow_{0+} z) \diamondsuit_{\rm up} \Sigma^*)$.

Proof. All the following manipulations are based on repeated application of Definitions 3.1, 3.2 and Lemmata 3.3, 3.8.

$$\begin{array}{llll}
\text{(i)} & z & \in & x \boxdot_{\mathcal{P}} y \text{ iff} \\
& z & \in & (x \diamondsuit^l_{\rm dn} \Sigma^*) \rightsquigarrow_{1+} (\theta(y) \diamondsuit^l_{\rm up} \Sigma^*) \quad \text{iff} \\
& z & \in & x\ (\diamondsuit^l_{\rm dn} ; \rightsquigarrow_{1+})\ (\Sigma^* ; \theta(y) \diamondsuit^l_{\rm up} \Sigma^*) \quad \text{iff} \\
& x & \in & z\ (\diamondsuit^l_{\rm dn} ; \rightsquigarrow_{1+})^l\ (\Sigma^* ; \theta(y) \diamondsuit^l_{\rm up} \Sigma^*) \quad \text{iff} \\
& x & \in & z\ (\sqcup\!\sqcup_{1+} ; \diamondsuit_{\rm dn})\ (\theta(y) \diamondsuit^l_{\rm up} \Sigma^* ; \Sigma^*) \quad \text{iff} \\
& x & \in & (z \sqcup\!\sqcup_{1+} (\theta(y) \diamondsuit^l_{\rm up} \Sigma^*)) \diamondsuit_{\rm dn} \Sigma^* \quad \text{iff} \\
& x & \in & z \boxdot^l_{\mathcal{P}} y
\end{array}$$

$$\begin{array}{lrcll}
\text{(ii)} & z & \in & x \boxdot_{\mathcal{P}} y \text{ iff} & \\
& z & \in & (x \diamondsuit_{\text{dn}}^{l} \Sigma^*) \rightsquigarrow_{1+} (\Sigma^* \diamondsuit_{\text{up}}^{ll} \theta(y)) & \text{iff} \\
& z & \in & (x \diamondsuit_{\text{dn}}^{l} \Sigma^* : \Sigma^*) \; (\rightsquigarrow_{1+} : \diamondsuit_{\text{up}}^{ll}) \; \theta(y) & \text{iff} \\
& \theta(y) & \in & (x \diamondsuit_{\text{dn}}^{l} \Sigma^* : \Sigma^*) \; (\rightsquigarrow_{1+} : \diamondsuit_{\text{up}}^{ll})^r \; z & \text{iff} \\
& \theta(y) & \in & (x \diamondsuit_{\text{dn}}^{l} \Sigma^* : \Sigma^*) \; (\rightsquigarrow_{1+} : \diamondsuit_{\text{up}}^{lr})^r \; z & \text{iff} \\
& \theta(y) & \in & (\Sigma^* : x \diamondsuit_{\text{dn}}^{l} \Sigma^*) \; (\diamondsuit_{\text{up}}^{l} : \rightsquigarrow_{0+}) \; z & \text{iff} \\
& \theta(y) & \in & \Sigma^* \diamondsuit_{\text{up}}^{l} ((x \diamondsuit_{\text{dn}}^{l} \Sigma^*) \rightsquigarrow_{0+} z) & \text{iff} \\
& y & \in & \theta(((x \diamondsuit_{\text{dn}}^{l} \Sigma^*) \rightsquigarrow_{0+} z) \diamondsuit_{\text{up}} \Sigma^*) & \text{iff} \\
& y & \in & x \boxdot_{\mathcal{P}}^{r} z &
\end{array}$$

□

In the rest of this section we denote $\Gamma = \{a \in \Sigma \mid a = \theta(a)\}$, a subalphabet of Σ such that θ is an identity over Γ. In the case of the DNA involution τ, of course, $\Gamma = \emptyset$.

Theorem 7.3 *The following problem is decidable in time $O((|A| \cdot |A_\theta| \cdot |A_M|)^3)$.*
Input: DFA's A, A_θ and an NFA A_M such that $L(A) = \theta(L(A_\theta)) \subseteq L(A_M)$ and $L(A)$ is θ-non-overlapping.
Output: Y/N, depending on whether $L(A)$ is a maximal θ-non-overlapping subset of $L(A_M)$.

Proof. Similarly as in the proof of Theorem 5.5, our problem can be reduced to testing whether $R - Q = \emptyset$, where

$$\begin{aligned} R &= M - (L \cup L \boxdot_{\mathcal{P}}^{r} \Sigma^* \cup \Sigma^* \boxdot_{\mathcal{P}}^{l} L), && (23)\\ Q &= \{z \in \Sigma^* \mid z \boxdot_{\mathcal{P}} z \neq \emptyset\}, && (24)\end{aligned}$$

and $\boxdot_{\mathcal{P}}$ is defined by (22). For the θ-non-overlapping property we have $\diamondsuit_{\text{dn}} = \diamondsuit_{\text{up}} = \sqcup\!\sqcup_{0+}$ by Theorem 6.2. Substituting into expressions in Lemma 7.2, we obtain after certain conversions

$$L \boxdot_{\mathcal{P}}^{r} \Sigma^* = \Sigma^* \boxdot_{\mathcal{P}}^{l} L = \theta(L).$$

Hence $R = M - (L \cup \theta(L))$ and the NFA accepting R can be constructed in the time $\mathcal{O}(|A| \cdot |A_\theta| \cdot |A_M|)$.

By (24) and Theorem 7.1, Q is the set of all words z such that the language $\{z\}$ does not satisfy the given property $\mathcal{P}$.

Let θ be an antimorphism. By definition of θ-non-overlapping property in Section 2, one can observe that

$$Q = \{z \in \Sigma^+ \mid z = \theta(z)\} = \{wa\theta(w) \mid w \in \Sigma^+, \; a \in \Gamma \cup \{\lambda\}\}.$$

The complement of Q is

$$Q^c = \{xay \mid x, y \in \Sigma^+, \; a \in \Sigma \cup \{\lambda\}, \; |x| = |y|, \; x \neq \theta(y) \text{ or } a \neq \theta(a)\} \cup \{\lambda\}.$$

Both Q and Q^c are context-free languages. Given a fixed PDA accepting Q^c whose size is constant, the PDA accepting $R - Q = R \cap Q^c$ has the size $\mathcal{O}(|A| \cdot |A_\theta| \cdot |A_M|)$. Then, by Theorem 7.1 in [6], we can construct an equivalent context-free grammar G of the size

$|G| = \mathcal{O}((|A| \cdot |A_\theta| \cdot |A_M|)^3)$. Finally, the algorithm described in [6], Sec. 7.4.3, testing whether $L(G) = \emptyset$, requires linear time with respect to $|G|$.

If θ is a morphism, then $Q = \Gamma^+$ and $Q^c = \Sigma^*(\Sigma - \Gamma)\Sigma^* \cup \{\lambda\}$, a fixed regular language. Then the problem is decidable in time $\mathcal{O}(|A| \cdot |A_\theta| \cdot |A_M|) = \mathcal{O}(|A|^2 \cdot |A_M|)$. □

Lemma 7.4 *Let θ be an antimorphism, let k be a positive integer and let S be the set of words u satisfying the condition $\mathrm{Sub}_k(u) \cap \mathrm{Sub}_k(\theta(u)) \neq \emptyset$. Then,*

$$S = \{xwy\theta(w)z \,|\, x, y, z \in \Sigma^*, w \in \Sigma^k\} \cup \{xwa\theta(w)y \,|\, x, y \in \Sigma^*, a \in \Gamma \cup \{\lambda\}, w \in \Sigma^{\lceil (k-|a|)/2 \rceil}\}.$$

Proof. First note that u is in S iff $u = xwz_1 = x_2\theta(w)z$, for some words x, z_1, x_2, z, w with $|w| = k$ and $|x| \le |x_2|$. Then one of the following conditions holds.

- $|xw| \le |x_2|$ and $u = xwy\theta(w)z$, for some word y.
- $|xw| > |x_2|$ and $u = xsvs'z$, for some words s, s', v with $w = sv$, $\theta(w) = vs'$, and $|v| > 0$.

In the first case, u is in $S_1 = \{xwy\theta(w)z \,|\, x, y, z \in \Sigma^*, w \in \Sigma^k\}$. In the second case we show that u must be in $S_2 = \{xga\theta(g)z \mid x, z \in \Sigma^*, a \in \Gamma \cup \{\lambda\}, g \in \Sigma^{\lceil (k-|a|)/2 \rceil}\}$. Indeed, as $\theta(w) = \theta(v)\theta(s) = vs'$, one has that $v = \theta(v)$ and $s' = \theta(s)$, which implies that $v = fa\theta(f)$ for some word f and $a \in \Gamma \cup \{\lambda\}$. As $|sv| = |s| + 2|f| + |a| = k$, it follows that $|sf| \ge \lceil (k - |a|)/2 \rceil$ – otherwise, $|sf| < \lceil (k - |a|)/2 \rceil$ would imply $|f| < \lceil (k - |a|)/2 \rceil$ and $|f| = k - |a| - |sf| > k - |a| - \lceil (k - |a|)/2 \rceil = \lfloor (k - |a|)/2 \rfloor$, which is impossible. Thus u is of the form $xsfa\theta(f)\theta(s)z$ with $|sfa| \ge \lceil k/2 \rceil$, which implies that u must be in S_2. Hence, $S \subseteq S_1 \cup S_2$.

For the converse inclusion, first note that $S_1 \subseteq S$. Moreover, every word $xga\theta(g)z$ in S_2 can be written in the form $xsfa\theta(f)\theta(s)z$ with $|sfa\theta(f)| = k$, by choosing $s = \lambda$ and $f = g$ if $k - |a|$ is even, or $s \in \Sigma$ and $g = sf$ if $k - |a|$ is odd. Hence, $S_2 \subseteq S$ as well. □

Theorem 7.5 *Let θ be an antimorphism. Let $\mathcal{P}$ be any of the properties strictly (B) – strictly (D), strictly (G), (L), strictly (L). Let $M \subseteq \Sigma^+$ be a regular set of words and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.*

Proof. As in the above proof, the decision algorithm must test whether $R - Q = \emptyset$, where R and Q are defined by (23) and (24), respectively. As M and L are regular languages, we can construct an NFA accepting R using Lemmata 3.9, 3.10, 7.2.

- For the properties strictly (B), strictly (C), strictly (D) we have $Q = \{wa\theta(w) \,|\, w \in \Sigma^+, a \in \Gamma \cup \{\lambda\}\}$. As we have shown in the proof of Theorem 7.3, Q^c is a context-free language and hence there is an algorithm to test whether $R - Q = R \cap Q^c = \emptyset$.
- For strictly (G), $Q = \bigcup_{a \in \Sigma}(a\Sigma^*\theta(a))$, a regular language and hence again the question "$R - Q = \emptyset$?" is decidable.
- For (L), $Q = \{z \,|\, \mathrm{Sub}_k(z) \cap \mathrm{Sub}_k(\theta(z)) \neq \emptyset\}$. Denote $Q = Q_L$ for further use. By Lemma 7.4,

  $$\begin{aligned} Q_L &= \{xwy\theta(w)z \,|\, x, y, z \in \Sigma^*, w \in \Sigma^k\} \\ &\cup \{xwa\theta(w)y \,|\, x, y \in \Sigma^*, a \in \Gamma \cup \{\lambda\}, w \in \Sigma^{\lceil k/2 \rceil}\}. \end{aligned}$$

  As Q_L is regular, the problem is decidable.

- For strictly (L),

$$Q = Q_L \cup \{wa\theta(w) \mid |w| < \lceil k/2 \rceil, a \in \Gamma \cup \{\lambda\}\}.$$

Again Q is regular and the problem is decidable.

□

Similar results as above can be obtained in the case of θ being a morphism, but again a technical result analogous to Lemma 7.4 is necesary first.

Lemma 7.6 *Let θ be a morphism, let k be a positive integer and let S be the set of words u satisfying the condition* $\mathrm{Sub}_k(u) \cap \mathrm{Sub}_k(\theta(u)) \neq \emptyset$. *Then,*

$$S = \{xwy\theta(w)z \mid x, y, z \in \Sigma^*, w \in \Sigma^k\} \cup \Sigma^* Z \Sigma^* \cup \Sigma^* \Gamma^k \Sigma^*,$$

where

$$Z = \{w\theta(w)\theta^2(w) \cdots \theta^n(w)x \mid w, x \in \Sigma^*, x \in \mathrm{Pref}(\theta^{n+1}(w)), n \geq 1, |w^n x| = k\}. \tag{25}$$

Proof. Denote $S_1 = \{xwy\theta(w)z \mid x, y, z \in \Sigma^*, w \in \Sigma^k\}$, $S_2 = \Sigma^* \Gamma^k \Sigma^*$ and $S_3 = \Sigma^* Z \Sigma^*$. Then we can express our statement as $S = S_1 \cup S_2 \cup S_3$. The same arguments as in the proof of Lemma 7.4 show that $S_1 \subseteq S$. Obviously also $S_2 \subseteq S$. Consider now a word $u \in S_3$, then

$$u = vw\theta(w)\theta^2(w) \cdots \theta^n(w)xy$$

for some $v, y \in \Sigma^*$, and w, x as in (25). Consequently,

$$\theta(u) = \theta(v)\theta(w)w\theta(w)\theta^2(w) \cdots \theta^{n-1}(w)\theta(x)\theta(y).$$

As $\theta(x) \in \mathrm{Pref}(\theta^n(w))$, we can write

$$w\theta(w)\theta^2(w) \cdots \theta^{n-1}(w)\theta(x) \in \mathrm{Sub}_k(u) \cap \mathrm{Sub}_k(\theta(u)).$$

Hence, for each $u \in S_3$, $\mathrm{Sub}_k(u) \cap \mathrm{Sub}_k(\theta(u)) \neq \emptyset$ and thus $S_1 \cup S_2 \cup S_3 \subseteq S$.

For the converse inclusion, assume that $u \in S$ but $u \notin S_1 \cup S_2$. We show that then $u \in S_3$. We can assume without loss of generality that $u = v_1 z y_1$, $\theta(u) = v_2 z y_2$ such that $|z| = k$ and $|v_1| \leq |v_2|$. Then one can derive that $|v_1| < |v_2| < |v_1 z|$. (If $|v_1| = |v_2|$, then $u \in S_2$; if $|v_2| \geq |v_1 z|$, then $u \in S_1$.) Denote $z = wz'$ such that $|w| = |v_2| - |v_1|$, $z' \in \Sigma^*$. Then

$$\begin{aligned} u &= v_1 w z' y_1, \\ \theta(u) &= \theta(v_1)\theta(w) w z' y_2. \end{aligned}$$

Suppose that $|z'| \geq |w|$, then $z' y_1 = \theta(w z' y_2)$ and hence $z' = \theta(w) z''$ for some $z'' \in \Sigma^*$. Let again $|z''| \geq |w|$, similarly we can deduce that $z'' = \theta^2(w) z'''$ for some $z''' \in \Sigma^*$. By induction, we get that

$$\begin{aligned} u &= v_1 w\theta(w) \cdots \theta^{n-1}(w) x' y_1, \\ \theta(u) &= \theta(v_1)\theta(w) w\theta(w) \cdots \theta^{n-1}(w) x' y_2, \end{aligned}$$

for some $n \geq 1$ and $|x'| < |w|$. As $x' y_1 = \theta(\theta^{n-1}(w) x' y_2) = \theta^n(w)\theta(x')\theta(y_2)$, we get $x' \in \mathrm{Pref}(\theta^n(w))$ and we denote $x = \theta(x') \in \mathrm{Pref}(\theta^{n+1}(w))$. Therefore,

$$u = v_1 w\theta(w) \cdots \theta^n(w) x\theta(y_2)$$

which concludes the proof. □

Theorem 7.7 *Let θ be a morphism. Let $\mathcal{P}$ be any of the properties strictly (B) – strictly (D), strictly (H), strictly (I), (L), strictly (L). Let $M \subseteq \Sigma^+$ be a regular set of words and $L \subseteq M$ a regular language satisfying $\mathcal{P}$. Then there is an algorithm deciding whether L is a maximal subset of M satisfying $\mathcal{P}$.*

Proof. As in the proof of Theorem 7.5, we need to decide whether $R - Q = \emptyset$, where R and Q are defined by (23) and (24), respectively, and R is a regular language.

- Using results of Theorem 7.3, one can easily derive for the properties strictly (B), strictly (C), strictly (D), that $Q = \Gamma^+$ and the problem is decidable.
- For strictly (H) or strictly (I), $Q = \Gamma\Sigma^*$ or $Q = \Sigma^*\Gamma$, respectively, both regular languages, and the problem is decidable.
- For (L), $Q = \{z \mid \mathrm{Sub}_k(z) \cap \mathrm{Sub}_k(\theta(z)) \neq \emptyset\}$. Denote $Q = Q_L$, for further use. By Lemma 7.6,

$$Q_L = \{xwy\theta(w)z \mid x, y, z \in \Sigma^*, w \in \Sigma^k\} \cup \Sigma^* Z \Sigma^* \cup \Sigma^* \Gamma^k \Sigma^*,$$

  where Z is defined by (25). As Z is finite, apparently Q_L is regular and the problem is decidable.
- For strictly (L), $Q = Q_L \cup \bigcup_{1 \leq i < k} \Gamma^i$, a regular language, hence the problem is decidable again.

□

The following theorem is a counterpart of Theorem 5.9 for the case of strictly bond-free properties.

Theorem 7.8 *Let $\mathcal{P}$ be a bond-free property and $M \subseteq \Sigma^+$ a set of words. Let $L \subseteq M$ be a language such that $L \in \mathcal{P}$. Denote*

$$\begin{aligned} L_1 &= M \cap (K_2^c \boxdot_{\mathcal{P}}^l L)^c \cap ((K_2^c \boxdot_{\mathcal{P}}^l L)^c \boxdot_{\mathcal{P}}^r K_2^c)^c, \\ L_2 &= M \cap (L \boxdot_{\mathcal{P}}^r K_2^c)^c \cap (K_2^c \boxdot_{\mathcal{P}}^l (L \boxdot_{\mathcal{P}}^r K_2^c)^c)^c. \end{aligned}$$

Then $L \subseteq L_i \subseteq M$ and $L_i \in \mathcal{P}$, for $i = 1, 2$.

Again, given a language L satisfying a certain strictly bond-free property, the above theorem allows us to construct "larger" languages satisfying the same property and containing L. Its proof follows by Proposition 6.3 in [13].

Acknowledgements

Research was partially supported by the Canada Research Chair Grant to L.K., NSERC Discovery Grants R2824A01 to L.K. and R220259 to S.K., and by the Grant Agency of Czech Republic Grant 201/02/P079 to P.S.

References

[1] M. Arita, S. Kobayashi, DNA sequence design using templates. *New Generation Computing* **20** (2002), 263–277.

[2] C.S. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor & Francis, London, 2001.

[3] M. Crochemore, C. Hancart, Automata for Matching Patterns, In G. Rozenberg, A. Salomaa (eds): *Handbook of Formal Languages*, vol II, 1997, pp. 399–462.

[4] M. Domaratzki, *Deletion Along Trajectories.* Tech. Report 464-2003, School of Computing, Queen's University, 2003, and submitted for publication.

[5] T. Head, Relativised code concepts and multi-tube DNA dictionaries. In C.S. Calude, G. Păun, *Finite Versus Infinite: Contributions to an Eternal Dilemma*, Springer-Verlag, London, 2000, 175–186.

[6] J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Boston, 2001.

[7] S. Hussini, L. Kari, S. Konstantinidis, Coding properties of DNA languages. *Theoretical Computer Science* **290/3** (2002), 1557-1579.

[8] N. Jonoska, D. Kephart, K. Mahalingam, Generating DNA code words. *Congressus Numerantium* **156** (2002), 99–110.

[9] N. Jonoska, K. Mahalingam, Languages of DNA based code words. In J. Chen, J. Reif (eds.), Preproceedings of DNA9, June 1–4, 2003, Madison, Wisconsin, pp. 58–68.

[10] L. Kari, On insertion and deletion in formal languages, *PhD thesis*, University of Turku, Finland, 1991.

[11] L. Kari, On language equations with invertible operations, *Theoretical Computer Science* **132** (1994), 129–150.

[12] L. Kari, R. Kitto, G. Thierrin, Codes, involutions and DNA encoding. In W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa (eds.), *Lecture Notes in Computer Science* **2300** (2002), 376–393.

[13] L. Kari, S. Konstantinidis, Language equations, maximality and error detection. Submitted for publication.

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

[15] L. Kari, P. Sosík, *Language deletion on trajectories.* Dept. of Computer Science Technical Report No. 606, University of Western Ontario, London, 2003.

[16] A. Marathe, A.E. Condon, R.M. Corn, On combinatorial DNA words design. *J. Computational Biology*, **8**:3, 2001.

[17] G. Mauri, C. Ferretti, *Word Design for Molecular Computing: A Survey.* Invited lecture at DNA9, June 1–4, 2003, Madison, Wisconsin, U.S, submitted for publication.

[18] A. Mateescu, G. Rozenberg, A. Salomaa, Shuffle on trajectories: syntactic constraints, TUCS technical report No. 41, Turku Centre for Computer Science, 1996, and *Theoretical Computer Science* **197** (1998), 1–56.

[19] G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms,* Springer-Verlag, Berlin, 1998.

[20] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages,* Springer-Verlag, Berlin, 1997.

[21] A. Salomaa, *Formal Languages,* Academic Press, London, 1973.

[22] S. Yu, Regular languages. In [20], pp. 41–110.