International Journal of Foundations of Computer Science

OPERATIONS ON TRAJECTORIES WITH APPLICATIONS TO CODING AND BIOINFORMATICS

LILA KARI
Department of Computer Science, The University of Western Ontario, London, Ontario, N6A 5B7, Canada, lila@csd.uwo.ca

and

STAVROS KONSTANTINIDIS
Dept. of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3, Canada, s.konstantinidis@smu.ca

and

PETR SOSÍK
Institute of Computer Science, Silesian University, 74601 Opava, Czech Republic, petr.sosik@fpf.slu.cz

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We study binary word operations of the insertion, deletion and substitution type. Many of these operations can be generalized into a unified framework by introducing so-called *trajectory condition*. This generalization has been previously made for insertion and deletion operations. In this paper we naturally extend this approach also to substitution operations. We study closure properties and decision problems of substitutions on trajectories. The obtained results are then applied to model complex noisy channels and a cryptanalysis problem. Another application concerns the design of sets of DNA strands without undesired bonds.

Keywords: formal languages; trajectory; noisy channel; biocomputing

1. Introduction

Binary word operations play an important role in formal language theory. They serve for composition/decomposition of languages and their descriptions (grammars, automata). They are also crucial for forming algebraic structures of formal languages, such as abstract families of languages (AFL) [26], and have numerous other applications. Besides their closure properties, language equations involving these operations have been studied. Various problems from automaton theory [26], coding theory [11], biocomputing [15] etc. have been reduced to finding solutions to language equations involving these operations [11, 19].

In this paper we focus on insertion/deletion/substitution word operations, such as catenation, insertion, quotient, shuffle, deletion, scattered deletion, substitution, etc. These operations differ in the positions where the letters of one operand are inserted/deleted/substituted into/from the other one. It turns out that one can characterize all these positions by a set of binary strings called *trajectories*. *Shuffle on trajectories* has been introduced and investigated in [25], characterizing a class of insertion operations. Also their applications to concurrent processes modelling were considered. Further related problems have been addressed e.g., in [22, 23]. An inverse operation, the *deletion on trajectories*, has been introduced in [3, 13]. Further theoretical results can be found e.g., in [4, 5, 6, 7], while for applications we refer to [14, 15].

As a further natural extension, we introduce in Section 4 the operations of *substitution on trajectories*. Substitution word operations were introduced in [11], where they have been used to model noisy channels. A basic principle is to replace certain letters of one argument by letters of the other argument. The trajectory condition can restrict positions or frequency of these replacements. The idea of substitution on trajectories seems to have interesting applications in coding theory and bioinformatics.

The paper is organized as follows. A basic description of deletion/insertion operations on trajectories is given in Section 3. Then in Section 4 we introduce *substitution on trajectories*. Closure properties of substitution on trajectories are studied in Section 5, and related decision questions in Section 6. In Section 7 we discuss a few applications of the substitution on trajectories in modelling complex noisy channels and a cryptanalysis problem. In the former case, the channels involved permit only substitution errors. This restriction allows us to improve the time complexity of the problem of whether a given regular language is error-detecting with respect to a given channel [18]. Finally, in Section 8 applications to bioinformatics are discussed. We characterize certain types of bonds of single DNA strands by operations on trajectories. This allows for construction of a quadratic-time algorithm testing the presence of such bonds in a given regular set of DNA words.

2. Definitions

An *alphabet* is a finite and nonempty set of symbols. In the sequel we shall use a fixed alphabet Σ which is assumed to be non-singleton, if not stated otherwise. 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 $|w|$, and $|w|_x$ denotes the number of occurrences of x within w, 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. The *Hamming distance* $H(u, v)$ between two words u and v of the same length is the number of corresponding positions in which u and v differ. For example, $H(abba, aaaa) = 2$.

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.

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. For a language L, we write L_λ to denote $L \cup \{\lambda\}$. 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 $L^1 \cup L^2 \cup \cdots$ and L^* for the language $L^+ \cup L^0$. The notation L^c represents the complement of the language L, that is, $L^c = \Sigma^* - L$.

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 $L(A)$. The *size* $|A|$ of the automaton A is the number $|S| + |P|$. We refer the reader to [26] for further details on automata and formal languages.

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^*. The *characteristic relation* of $\diamondsuit$ is

$$C_\diamondsuit = \{(w, u, v) : w \in u \diamondsuit v\}.$$

For any languages X and Y, we define

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

It should be noted that every subset B of $\Sigma^* \times \Sigma^* \times \Sigma^*$ defines a unique binary word operation whose characteristic relation is exactly B. The left inverse $\diamondsuit^l$ and the right inverse $\diamondsuit^r$ of $\diamondsuit$ are defined as

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

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

Moreover, the word operation $\diamondsuit'$ defined by $u \diamondsuit' v = v \diamondsuit u$ is called reversed $\diamondsuit$. It should be clear that, for every binary operation $\diamondsuit$, the triple (w, u, v) is in $C_\diamondsuit$ if and only if (u, w, v) is in $C_{\diamondsuit^l}$ if and only if (v, u, w) is in $C_{\diamondsuit^r}$ if and only if (w, v, u) is in $C_{\diamondsuit'}$. If x and y are symbols in $\{l, r, '\}$, the notation $\diamondsuit^{xy}$ represents the operation $(\diamondsuit^x)^y$. Using the above observations, one can establish identities between operations of the form $\diamondsuit^{xy}$.

Lemma 1 *(i)* $\diamondsuit^{ll} = \diamondsuit^{rr} = \diamondsuit'' = \diamondsuit$,

(ii) $\diamondsuit'^{l} = \diamondsuit^{r'} = \diamondsuit^{lr}$,

(iii) $\diamondsuit'^{r} = \diamondsuit^{l'} = \diamondsuit^{rl}$.

Bellow we list several binary word operations together with their left and right inverses [10].

Catenation[a]*:* $u \cdot v = \{uv\}$, 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 $\longrightarrow_{\mathrm{rq}}^r = \longrightarrow_{\mathrm{lq}}'$.

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 $\rightsquigarrow^r = \rightsquigarrow$.

3. Shuffle and Deletion on Trajectories

The above insertion and deletion operations can be naturally generalized using the concept of *trajectories.* A trajectory defines an order in which the operation is applied to the letters of its arguments. Notice that this restriction is purely syntactical, as the content of the arguments has no influence on this order. Formally, a trajectory is a string over the *trajectory alphabet* $V = \{0, 1\}$. The following definitions are due to [3, 13, 25].

Let Σ be an alphabet and let t be a trajectory, $t \in V^*$. Let α, β be two words over Σ.

Definition 1 *The shuffle of α with β on trajectory t, denoted by $\alpha \sqcup\!\sqcup_t \beta$, is defined as follows:*

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

Observe that due to the above definition, if $|\alpha| \neq |t|_0$ or $|\beta| \neq |t|_1$, then $\alpha \sqcup\!\sqcup_t \beta = \emptyset$.

Definition 2 *The deletion of β from α on trajectory t is the following binary word operation:*

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

Similarly as in the previous definition, if $|\alpha| \neq |t|$ or $|\beta| \neq |t|_1$, then $\alpha \rightsquigarrow_t \beta = \emptyset$.

A *set of trajectories* is any set $T \subseteq V^*$. We extend the shuffle and deletion to sets of trajectories as follows:

$$\alpha \sqcup\!\sqcup_T \beta = \bigcup_{t \in T} \alpha \sqcup\!\sqcup_t \beta, \qquad \alpha \rightsquigarrow_T \beta = \bigcup_{t \in T} \alpha \rightsquigarrow_t \beta. \tag{2}$$

The operations $\sqcup\!\sqcup_T$ and $\rightsquigarrow_T$ generalize to languages by (1).

[a] We shall also write uv for $u \cdot v$.

Example. The following binary word operations can be expressed via shuffle on certain sets of trajectories.

(i) Let $T = 0^*1^*$, then $\sqcup\!\sqcup_T \; = \; \cdot$, the catenation operation, and $\rightsquigarrow_T \; = \; \longrightarrow_{\mathrm{rq}}$, the right quotient.

(ii) For $T = 1^*0^*$ we have $\sqcup\!\sqcup_T \; = \; \cdot'$, the anti-catenation, and $\rightsquigarrow_T \; = \; \longrightarrow_{\mathrm{lq}}$, the left quotient.

(iii) Let $T = \{0,1\}^*$, then $\rightsquigarrow_T \; = \; \sqcup\!\sqcup$, the shuffle, and $\rightsquigarrow_T \; = \; \rightsquigarrow$, the scattered deletion.

Lemma 2 *Let T be a set of trajectories. Then $\sqcup\!\sqcup_T^l \; = \rightsquigarrow_T$ and $\rightsquigarrow_T^l \; = \; \sqcup\!\sqcup_T$.*

Lemma 3 *For all regular languages L_1, L_2, and a regular set of trajectories T, both $L_1 \sqcup\!\sqcup_T L_2$ and $L_1 \rightsquigarrow_T L_2$ are regular languages.*

Furthermore, let A_1, A_2 and A_T be NFA's accepting L_1, L_2, and T, respectively. Then there exists an NFA (λ-NFA for $\rightsquigarrow_T$) of the size $\mathcal{O}(|A_1| \cdot |A_2| \cdot |A_T|)$ accepting the language $L_1 \sqcup\!\sqcup_T L_2$ ($L_1 \rightsquigarrow_T L_2$, respectively).

4. Substitution on Trajectories

Based on the previously studied concepts of the insertion and deletion on trajectories, we consider a generalization of three natural binary word operations which are used to model certain noisy channels [11]. Generally, a *channel* [18] is a binary relation $\gamma \subseteq \Sigma^* \times \Sigma^*$ such that (u, u) is in γ for every word u in the input domain of γ – this domain is the set $\{u \mid (u, v) \in \gamma$ for some word $v\}$. The fact that (u, v) is in γ means that the word v can be received from u via the channel γ.

In [11], certain channels with insertion, deletion and substitution errors are characterized via word operations. For instance, the channel with exactly m insertion errors is the set of all pairs (u, v) such that $v \in u \sqcup\!\sqcup \Sigma^m$, and analogously for deletion errors. The following operations allow to characterize channels with substitution errors.

Definition 3 *For a trajectory $t \in V^*$ and $u, v \in \Sigma^*$ we define the* substitution in u by v *on trajectory t as*

$$\begin{aligned} u \bowtie_t v \;\; = \;\; & \{u_1 v_1 u_2 v_2 \ldots u_k v_k u_{k+1} \mid k \geq 0,\; u = u_1 a_1 \ldots u_k a_k u_{k+1},\; v = v_1 \ldots v_k, \\ & t = 0^{j_1} 1 0^{j_2} 1 \ldots 0^{j_k} 1 0^{j_{k+1}},\; a_i, v_i \in \Sigma, 1 \leq i \leq k,\; a_i \neq v_i, \forall i, 1 \leq i \leq k, \\ & j_i = |u_i|, 1 \leq i \leq k+1\}. \end{aligned}$$

Definition 4 *For a trajectory $t \in V^*$ and $u, v \in \Sigma^*$ we define the* substitution in u of v *on trajectory t as*

$$\begin{aligned} u \,\triangle_t\, v \;\; = \;\; & \{u_1 a_1 u_2 a_2 \ldots u_k a_k u_{k+1} \mid k \geq 0,\; u = u_1 v_1 \ldots u_k v_k u_{k+1},\; v = v_1 \ldots v_k, \\ & t = 0^{j_1} 1 0^{j_2} 1 \ldots 0^{j_k} 1 0^{j_{k+1}},\; a_i, v_i \in \Sigma, 1 \leq i \leq k,\; a_i \neq v_i, \forall i, 1 \leq i \leq k, \\ & j_i = |u_i|, 1 \leq i \leq k+1\}. \end{aligned}$$

Definition 5 *For a trajectory $t \in V^*$ and $u, v \in \Sigma^*$ we define the* right difference of u and v *on trajectory t as*

$$u \rhd_t v \;\; = \;\; \{v_1 v_2 \ldots v_k \mid k \geq 0,\; u = u_1 a_1 \ldots u_k a_k u_{k+1},\; v = u_1 v_1 \ldots u_k v_k u_{k+1},$$

$t = 0^{j_1}10^{j_2}1\ldots 0^{j_k}10^{j_{k+1}},\ a_i, v_i \in \Sigma, 1 \le i \le k,\ a_i \neq v_i, \forall i, 1 \le i \le k,$
$j_i = |u_i|, 1 \le i \le k+1\}.$

Example. Consider an alphabet $\Sigma = \{a, b, c\}$. Then

(i) $aaaa \bowtie_{0110} bc = \{abca\}$,
(ii) $aaaa \bigtriangleup_{0110} aa = \{abba, abca, acba, acca\}$,
(iii) $aaaa \rhd_{0110} abca = \{bc\}$.

We note that, analogously to Definitions 1 and 2, if $|u| \neq |t|$ or $|v| \neq |t|_1$, then $u \bowtie_t v = u \bigtriangleup_t v = \emptyset$. Similarly, if the condition $|u| = |v| = |t|$ is not met, then $u \rhd_t v = \emptyset$. Observe also that for an arbitrary $u, v \in \Sigma^*$, $t \in V^*$ one gets $|u \bowtie_t v| \le 1$, $|u \rhd_t v| \le 1$, but $|u \bigtriangleup_t v| \le (|\Sigma| - 1)^{|t|_1}$.

These operations can be generalized to sets of trajectories in the natural way:

$$u \bowtie_T v = \bigcup_{t \in T} u \bowtie_t v, \quad u \bigtriangleup_T v = \bigcup_{t \in T} u \bigtriangleup_t v \ \text{ and } \ u \rhd_T v = \bigcup_{t \in T} u \rhd_t v.$$

We give notice of the fact that the notation $\bowtie_T$ was used also by A. Mateescu in [21] and some other papers to denote splicing on routes.

Example. For positive integers m and l, with $m < l$, consider the SID (Substitution, Insertion, Deletion) channel [17] that permits at most m substitution errors in any l (or less) consecutive symbols of any input message. Using the operation $\bowtie_T$, this channel is defined as the set of pairs of words (u, v) such that $u \in v \bowtie_T \Sigma^*$, where T is the set of all trajectories t such that, for any subword s of t, if $|s| \le l$ then $|s|_1 \le m$.

One can observe that similarly as in [11], substitution on trajectories can characterize channels where errors occur in certain parts of words only, or with a certain frequency. If we replace the language Σ^* in the above example by a more specific one, we can also model channels where errors depend on the content of the message.

Lemma 4 *For a set of trajectories T and words $u, v \in \Sigma^*$, the following holds:*

(i) $\bowtie_T^l = \bigtriangleup_T$ *and* $\bowtie_T^r = \rhd_T$,
(ii) $\bigtriangleup_T^l = \bowtie_T$ *and* $\bigtriangleup_T^r = \rhd_T'$,
(iii) $\rhd_T^l = \bigtriangleup_T'$ *and* $\rhd_T^r = \bowtie_T$.

Proof.

(i) Recall the characteristic relation $C_{\bowtie_t}$ of the operation $\bowtie_t$. The statements $\bowtie_t^l = \bigtriangleup_t$ and $\bowtie_t^r = \rhd_t$, $t \in T$, follow directly by careful reading of the definitions of $\bowtie_t$, $\bigtriangleup_t$ and $\rhd_t$. Now observe that

$$u \bowtie_T^l v = \bigcup_{t \in T} u \bowtie_t^l v = \bigcup_{t \in T} u \bigtriangleup_t v = u \bigtriangleup_T v.$$

The proof for $\bowtie_T^r$ is analogous.

(ii) Due to Lemma 1, $\bowtie_T^l = \bigtriangleup_T$ implies $\bigtriangleup_T^l = \bowtie_T$ and $\bowtie_T^r = \rhd_T$ implies $\bigtriangleup_T^r = \bowtie_T^{lr} = \bowtie_T^{r'} = \rhd_T'$.

(iii) Similarly, $\bowtie_T^r = \rhd_T$ implies $\rhd_T^r = \bowtie_T$, and consequently $\rhd_T^l = \bowtie_T^{rl} = \bowtie_T^{l'} = \bigtriangleup_T'$.

□

5. Closure Properties

Before addressing the closure properties of substitution, we show first that any (not necessarily recursively enumerable) language over a two letter alphabet can be obtained as a result of substitution.

Lemma 5 *For an arbitrary language $L \subseteq \{a,b\}^*$ there exists a set of trajectories T such that*

(i) $L = a^* \bowtie_T b^*$,

(ii) $L = a^* \bigtriangleup_T a^*$.

Proof. Let $T = \phi(L)$, $\phi : \{a,b\}^* \longrightarrow V^*$ being a coding morphism such that $\phi(a) = 0$, $\phi(b) = 1$. The statements follow easily by definition. □

Similarly as in the case of shuffle and deletion on trajectories [3, 13, 25], the substitution on trajectories can be characterized by simpler language operations.

Lemma 6 *Let $\diamondsuit_T$ be any of the operations $\bowtie_T$, $\bigtriangleup_T$, $\rhd_T$. Then there exists a finite substitution h_1, morphisms h_2, g and a regular language R such that for all languages $L_1, L_2 \subseteq \Sigma^*$, and for all sets of trajectories $T \subseteq V^*$,*

$$L_1 \diamondsuit_T L_2 = g((h_1(L_1) \sqcup\!\sqcup h_2(L_2) \sqcup\!\sqcup T) \cap R). \tag{3}$$

Proof. Let $\Sigma_i = \{a_i \mid a \in \Sigma\}$, for $i = 1,2,3$, be copies of Σ such that Σ, Σ_1, Σ_2, Σ_3 and V are pairwise disjoint alphabets. For a letter $a \in \Sigma$, we denote by a_i the corresponding letter from Σ_i, $i = 1,2,3$.

Let further $h_1 : \Sigma \longrightarrow (\Sigma_1 \cup \Sigma_3)$ be a finite substitution and let $h_2 : \Sigma \longrightarrow \Sigma_2$ and $g : (\Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup V) \longrightarrow \Sigma$ be morphisms.

(i) If $\diamondsuit_T = \bowtie_T$, then define $h_1(a) = \{a_1, a_3\}$, $h_2(a) = a_2$ for each $a \in \Sigma$. Let

$$R = (\Sigma_1 \cdot \{0\} \cup \{a_3 b_2 1 \mid a, b \in \Sigma, a \neq b\})^*.$$

Let further $g(a_1) = a$, $g(a_2) = a$ for all $a_1 \in \Sigma_1$, $a_2 \in \Sigma_2$, and $g(x) = \lambda$ for all $x \in \Sigma_3 \cup V$. Then one can easily verify that (3) holds true.

(ii) If $\diamondsuit_T = \bigtriangleup_T$, then let $h_1(a) = \{a_1\} \cup \{a_3\} \cdot \Sigma_1$, $h_2(a) = a_2$ for each $a \in \Sigma$. Let further

$$R = (\Sigma_1 \cdot \{0\} \cup \{a_3 a_2 b_1 1 \mid a, b \in \Sigma, a \neq b\})^*,$$

and $g(a_1) = a$ for all $a_1 \in \Sigma_1$, $g(x) = \lambda$ for all $x \in \Sigma_2 \cup \Sigma_3 \cup V$.

(iii) If $\diamondsuit_T = \rhd_T$, then define $h_1(a) = a_1$, $h_2(a) = \{a_2, a_3\}$ for each $a \in \Sigma$. Let

$$R = (\{a_1 a_2 0 \mid a \in \Sigma\} \cup \{a_1 b_3 1 \mid a, b \in \Sigma, a \neq b\})^*,$$

and $g(a_3) = a$ for all $a_3 \in \Sigma_3$, $g(x) = \lambda$ for all $x \in \Sigma_1 \cup \Sigma_2 \cup V$.

□

Theorem 1 *Let $\diamondsuit_T$ be any of the operations $\bowtie_T$, $\bigtriangleup_T$, $\rhd_T$. Let L_1, $L_2 \subseteq \Sigma^*$ be regular languages and $T \subseteq V^*$ a regular set of trajectories. Let A_1, A_2 and A_T be NFA's accepting L_1, L_2 and T, respectively.*

Then there exists an NFA (a λ-NFA if $\diamondsuit_T = \rhd_T$) A of the size $|A| = \mathcal{O}(|A_1| \cdot |A_T| \cdot |A_2|)$ accepting $L_1 \diamondsuit_T L_2$.

Proof. Denote $A_T = (Q_T, V, s_T, F_T, p_T)$ and $A_i = (Q_i, \Sigma, s_i, F_i, p_i)$ for $i = 1, 2$. We show the construction of an NFA A accepting $L_1 \bowtie_T L_2$, the remaining cases are analogous. Informally, given a word $x \in \Sigma^*$, A chooses nondeterministically a trajectory $t \in T$, words $x_1 \in L_1$ and $x_2 \in L_2$ and tests whether $\{x\} = x_1 \bowtie_t x_2$. Denote $A = (Q, \Sigma, s, F, p)$. Let $Q = Q_1 \times Q_T \times Q_2$, $s = (s_1, s_T, s_2)$, $F = F_1 \times F_T \times F_2$, and p be defined as follows. For all $q_1 \in Q_1$, $q_T \in Q_T$, $q_2 \in Q_2$, $a \in \Sigma$,

$$
\begin{array}{ll}
(q_1, q_T, q_2)\, a \to (q_1', q_T', q_2) & \text{for all } q_1' \in Q_1,\ q_T' \in Q_T \text{ such that } q_1 a \to q_1', \\
& q_T 0 \to q_T'; \\
(q_1, q_T, q_2)\, a \to (q_1', q_T', q_2') & \text{for all } q_1' \in Q_1,\ q_T' \in Q_T,\ q_2' \in Q_2 \text{ such that } q_1 b \to q_1', \\
& q_T 1 \to q_T',\ q_2 a \to q_2',\ b \in \Sigma,\ a \neq b.
\end{array}
$$

The reader can easily verify that $L(A) = L_1 \bowtie_T L_2$. □

Theorem 2 *Let $\diamondsuit_T$ be any of the operations $\bowtie_T$, $\triangle_T$, $\triangleright_T$.*

- *(i) Let any two of the languages L_1, L_2, T be regular and the third one be context-free. Then $L_1 \diamondsuit_T L_2$ is a context-free language.*
- *(ii) Let any two of the languages L_1, L_2, T be context-free and the third one be regular. Then $L_1 \diamondsuit_T L_2$ is a non-context-free language for some triples (L_1, L_2, T).*

Proof.

- (i) Follows by Lemma 6, and by closure of the class of context-free languages with respect to finite substitution, shuffle, morphisms and intersection with regular languages.
- (ii) Consider the alphabet $\Sigma = \{a, b, c, d\}$.
  - (a) Let $\diamondsuit_T = \bowtie_T$.
    - (1) Consider $L_1 = \{a^n d b^{2n} \mid n > 0\}$, $L_2 = \{a^m c^m \mid m > 0\}$ and $T = V^*$, then $(L_1 \bowtie_T L_2) \cap a^* d a^* c^* = \{a^n d a^n c^n \mid n > 0\}$.
    - (2) Consider $L_1 = \{a^n b^{2n} \mid n > 0\}$, $L_2 = c^+$ and $T = \{0^{2m} 1^m \mid m > 0\}$, then $L_1 \bowtie_T L_2 = \{a^n b^n c^n \mid n > 0\}$.
    - (3) Consider $L_1 = a^+$, $L_2 = \{b^n c^n \mid n > 0\}$ and $T = \{0^m 1^{2m} \mid m > 0\}$, then $L_1 \bowtie_T L_2 = \{a^n b^n c^n \mid n > 0\}$.
  - (b) Let $\diamondsuit_T = \triangle_T$. Consider:
    - (1) $L_1 = \{a^n b a^k b a^l \mid k + l + 1 = 2n > 0\}$, $L_2 = \{a^m b a^{m+1} \mid m > 0\}$ and $T = 0^+ 1^+$,
    - (2) $L_1 = \{a^n b^n a^m \mid n > 0, m \geq 0\}$, $L_2 = a^+$ and $T = \{0^{2m+1} 1^m \mid m > 0\}$,
    - (3) $L_1 = a^+ b a^+$, $L_2 = \{a^n b a^n \mid n > 0\}$ and $T = \{0^m 1^{2m+1} \mid m > 0\}$,

    then in all three cases $(L_1 \triangle_T L_2) \cap a^* b^* a b^* = \{a^n b^n a b^n \mid n > 0\}$.
  - (c) Let $\diamondsuit_T = \triangleright_T$. Consider:
    - (1) $L_1 = \{c^{2m} d c^m \mid m > 0\}$, $L_2 = \{a^n b^n d a^m \mid n, m > 0\}$ and $T = V^+$,
    - (2) $L_1 = \{b^n a^n d b^m \mid n, m > 0\}$, $L_2 = a^+ b^+ d a^+$ and $T = \{1^{2m} 0 1^m \mid m > 0\}$,
    - (3) $L_1 = c^+ d c^+$, $L_2 = \{a^n b^n d a^m \mid n, m > 0\}$ and $T = \{1^{2m} 0 1^m \mid m > 0\}$,

then in all three cases $(L_1 \rhd_T L_2) \cap \{a,b\}^* = \{a^n b^n a^n \mid n > 0\}$.

In all the above cases we have shown that $L_1 \diamondsuit_T L_2$ is a non-context-free language.

$\square$

6. Decision Problems

In this section we study three elementary types of decision problems for language equations of the form $L_1 \diamondsuit_T L_2 = R$, where $\diamondsuit_T$ is one of the operations $\bowtie_T$, $\triangle_T$, $\rhd_T$. These problems, studied already for various binary word operations in [3, 9, 10, 13] and others, are stated as follows. First, given L_1, L_2 and R, one asks whether the above equation holds true. Second, the existence of a solution L_1 to the equation is questioned, when L_1 is unknown (the left operand problem). Third, the same problem is stated for the right operand L_2. All these problems have their variants when one of L_1, L_2 (the unknown language in the case of the operand problems) consists of a single word.

We focus now on the case when L_1, L_2 and T are all regular languages. Then $L_1 \diamondsuit_T L_2$ is also a regular language by Theorem 1, $\diamondsuit_T$ being any of the operations $\bowtie_T$, $\triangle_T$, $\rhd_T$. Immediately we obtain the following result.

Theorem 3 *The following problems are both decidable if the operation $\diamondsuit_T$ is one of $\bowtie_T$, $\triangle_T$, $\rhd_T$, T being a regular set of trajectories:*

(i) For given regular languages L_1, L_2, R, is $L_1 \diamondsuit_T L_2 = R$?

(ii) For given regular languages L_1, R and a word $w \in \Sigma^*$, is $L_1 \diamondsuit_T w = R$?

Also the decidability of the left and the right operand problems for languages is a straightforward consequence of the results in Section 5 and some previously known facts about language equations [10].

Theorem 4 *Let $\diamondsuit_T$ be one of the operations $\bowtie_T$, $\triangle_T$, $\rhd_T$. The problem "Does there exist a solution X to the equation $X \diamondsuit_T L = R$?" (left-operand problem) is decidable for regular languages L, R and a regular set of trajectories T.*

Proof. Due to [10], if a solution to the equation $X \diamondsuit_T L = R$ exists, then also $X_{\max} = (R^c \diamondsuit_T^l L)^c$ is also a solution, $\diamondsuit_T$ being an invertible binary word operation. In fact, $X_{\max}$ is the maximum (with respect to the subset relation) of all the sets X such that $X \diamondsuit_T L \subseteq R$. We can conclude that a solution X exists *iff*

$$(R^c \diamondsuit_T^l L)^c \diamondsuit_T L = R. \tag{4}$$

holds. Observe that if $\diamondsuit_T$ is one of $\bowtie_T$, $\triangle_T$, $\rhd_T$, then $\diamondsuit_T^l$ is $\triangle_T$, $\bowtie_T$ or $\triangle'_T$, respectively, by Lemma 4. Hence the left side of the equation (4) represents an effectively constructible regular language by Theorem 1. Consequently, the equality of (4) is decidable and moreover the maximal solution $X_{\max} = (R^c \diamondsuit_T^l L)^c$ can be effectively found if one exists. $\square$

Theorem 5 *Let $\diamondsuit_T$ be one of the operations $\bowtie_T$, $\triangle_T$, $\rhd_T$. The problem "Does there exist a solution X to the equation $L \diamondsuit_T X = R$?" (right-operand problem) is decidable for regular languages L, R and a regular set of trajectories T.*

Proof. Similarly as in the proof of Theorem 4, a maximal solution to the equation $L \diamondsuit_T X = R$ is $X_{\max} = (L \diamondsuit_T^r R^c)^c$ for a binary word operation $\diamondsuit_T$, see [10]. Hence a solution X exists *iff*

$$L \diamondsuit_T (L \diamondsuit_T^r R^c)^c = R \tag{5}$$

By Lemma 4, if $\diamondsuit_T$ is one of $\ltimes_T$, $\triangle_T$, $\rhd_T$, then $\diamondsuit_T^r$ is $\rhd_T$, $\rhd_T'$ or $\ltimes_T$, respectively. Again the equality of (5) is effectively decidable by Theorem 1, and, moreover, an eventual maximal solution $X_{\max} = (L \diamondsuit_T^r R^c)^c$ can be effectively found. □

The situation is a bit different in the case when the existence of a singleton solution to the left or the right operand problem is questioned. Another proof technique takes place.

Theorem 6 *Let $\diamondsuit_T$ be one of the operations $\ltimes_T$, $\triangle_T$, $\rhd_T$. The problem "Does there exist a word w such that $w \diamondsuit_T L = R$?" is decidable for regular languages L, R and a regular set of trajectories T.*

Proof. Assume that $\diamondsuit_T$ is one of $\ltimes_T$, $\triangle_T$, $\rhd_T$. Observe first that if $y \in w \diamondsuit_T x$ for some $w, x, y \in \Sigma^*$, then $|y| \leq |w|$. Therefore, if R is infinite, then there cannot exist a solution w of a finite length satisfying $w \diamondsuit_T L = R$. Hence for an infinite R the problem is trivial.

Assume now that R is finite. As shown in [10], the regular set $X_{\max} = (R^c \diamondsuit_T^l L)^c$ is the maximal set with the property $X \diamondsuit_T L \subseteq R$. Hence w is a solution of $w \diamondsuit_T L = R$ *iff*

(i) $w \diamondsuit_T L \subseteq R$, i.e. $w \in X_{\max}$, and
(ii) $w \diamondsuit_T L \not\subset R$.

Moreover, (ii) is satisfied *iff* $w \diamondsuit_T L \not\subseteq R_1$ for all $R_1 \subset R$, and hence $w \notin (R_1^c \diamondsuit_T^l L)^c$. Hence we can conclude that the set S of all singleton solutions to the equation $w \diamondsuit_T L = R$ can be expressed as

$$S = (R^c \diamondsuit_T^l L)^c - \bigcup_{R_1 \subset R} (R_1^c \diamondsuit_T^l L)^c.$$

Since we assume that R is finite, the set S is regular and effectively constructible by Lemma 4, Theorem 1 and the closure of the class of regular languages under finite union and under complement. Hence it is also decidable whether S is empty or not, and eventually all its elements can be effectively listed. □

Theorem 7 *Let $\diamondsuit_T$ be one of the operations $\ltimes_T$, $\triangle_T$, $\rhd_T$. The problem "Does there exist a word w such that $L \diamondsuit_T w = R$?" is decidable for regular languages L, R and a regular set of trajectories T.*

Proof. Assume first that $\diamondsuit_T$ is one of $\ltimes_T$, $\triangle_T$. Observe that if $y \in x \diamondsuit_T w$ for some $w, x, y \in \Sigma^*$, then $|y| \geq |w|$. Therefore, if a solution w to the equation $L \diamondsuit_T w = R$ exists, then $|w| \leq k$, where $k = \min\{|y| \mid y \in R\}$. Hence, to verify whether a solution exists or not, it suffices to test all the words from $\Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^k$.

Focus now on the operation $\rhd_T$. Analogously to the case of Theorem 6, we can deduce that there is no word w satisfying $L \rhd_T w = R$, if R is infinite. Furthermore, the set $X_{\max} = (L \rhd_T^r R^c)^c = (L \ltimes_T R^c)^c$ is the maximal set with the property

$L \triangleright_T X \subseteq R$. The same arguments as in the proof of Theorem 6 allow one to express the set of all singleton solutions as

$$S = (L \bowtie_T R^c)^c - \bigcup_{R_1 \subset R} (L \bowtie_T R_1^c)^c.$$

For a finite R, the set S is regular and effectively constructible, hence we can decide whether it contains at least one solution. □

We add that in the above cases of the left and the right operand problems, if there exists a solution, then at least one can be effectively found. Moreover, in the case of their singleton variants, *all* the singleton solutions can be effectively enumerated.

7. Applications to Coding

In this section we discuss a few applications of the substitution-on-trajectories operation in modelling certain noisy channels and a cryptanalysis problem. In the former case, we revisit a decidability question involving the property of error-detection.

Recall the example of a noisy channel characterized by the substitution on trajectories in Section 4. In general, following the notation of [11], for any trajectory set T we shall denote by $[\bowtie_T \Sigma^*]$ the channel $\{(u, v) \mid u \in \Sigma^*, v \in u \bowtie_T \Sigma^*\}$. In the context of noisy channels, the concept of error-detection is fundamental [18]. A language L is called *error-detecting for* a channel γ, if γ cannot transform a word in L_λ to another word in L_λ; that is, if $u, v \in L_\lambda$ and $(u, v) \in \gamma$ then $u = v$. Here L_λ is the language $L \cup \{\lambda\}$. The empty word in this definition is needed in case the channel permits symbols to be inserted into, or deleted from, messages – see [18] for details. In our case, where only substitution errors are permitted, the above definition remains valid if we replace L_λ with L.

In [18] it is shown that, given a rational relation γ and a regular language L, we can decide in polynomial time whether L is error-detecting for γ. Here we take advantage of the fact that the channels $[\bowtie_T \Sigma^*]$ permit only substitution errors and improve the time complexity of the above result.

Theorem 8 *The following problem is decidable in time $O(|A|^2|T|)$.*

Input: NFA A over Σ and NFA A_T over $\{0, 1\}$, such that $L(A_T) = T$.

Output: Yes/No, depending on whether $L(A)$ is error-detecting for $[\bowtie_T \Sigma^*]$.

Proof. In [12] it is shown that given an NFA A, one can construct the NFA A^σ, in time $O(|A|^2)$, such that the alphabet of A^σ is $E = \Sigma \times \Sigma$ and the language accepted by A^σ consists of all the words of the form $(x_1, y_1) \cdots (x_n, y_n)$, with each $(x_i, y_i) \in E$, such that $x_1 \cdots x_n \neq y_1 \cdots y_n$ and the words $x_1 \cdots x_n$ and $y_1 \cdots y_n$ are in $L(A)$. Let ϕ be the morphism of E into $\{0, 1\}$ such that $\phi(x, y) = 0$ iff $x = y$. One can verify that $L(A)$ is error-detecting for $[\bowtie_T \Sigma^*]$ iff the language $\phi(L(A^\sigma)) \cap T$ is empty. Using this observation, the required algorithm consists of the following steps: (i) Construct the NFA A^σ from A. (ii) Construct the NFA $\phi(A^\sigma)$ by simply replacing each transition $s(x, y) \to t$ of A^σ with $s\phi(x, y) \to t$. (iii) Use a product construction on $\phi(A^\sigma)$ and A_T to obtain an NFA B accepting $\phi(L(A^\sigma)) \cap T$. (iv)

Perform a depth first search algorithm on the graph of B to test whether there is a path from the start state to a final state. $\square$

We close this section with a cryptanalysis application of the operation $\bowtie_T$. Let M be a set of candidate binary messages (words over $\{0,1\}$) and let K be a set of possible binary keys. An unknown message v in M is encrypted as $v \oplus t$, where t is an unknown key in K, and $\oplus$ is the exclusive-OR logic operation. Let e be an observed encrypted message and let T be a set of possible guesses for t, with $T \subseteq K$. We want to find the subset X of M for which $X \oplus T = e$, that is, the possible original messages that can be encrypted as e using the keys we have guessed in T. In general T can be infinite and given, for instance, by a regular expression describing the possible pattern of the key. We can model this problem using the following observation whose proof is based on the definitions of the operations $\bowtie_T$ and $\oplus$, and is left to the reader.

Lemma 7 *For every word $v \in \{0,1\}*$ and trajectory t, $v \bowtie_t \{0,1\}* = \{v \oplus t\}$.*

By the above lemma, we have that the equation $X \oplus T = e$ is equivalent to $X \bowtie_T \Sigma^* = e$. By Theorem 4, we can decide whether there is a solution for this equation and, in this case, find the maximal solution $X_{\max}$. In particular, $X_{\max} = (e^c \Delta_T \Sigma^*)^c$. Hence, one needs to compute the set $M \cap X_{\max}$. Most likely, for a general T, this problem is intractable. On the other hand, this method provides an alternate way to approach the problem.

8. Applications to Bioinformatics

During many laboratory protocols involving manipulation of single DNA strands, the following problem arises: one designs an experiment, assuming certain bonds between these strands. Simultaneously, it is necessary to prevent any other undesired types of bonds. Therefore one has to design carefully the set of single DNA strands to prevent undesired bonds. A typical example is the design of primers for a site-specific PCR reaction. Another case is the design of coding for DNA computing processes, as in the famous Adleman's experiment [1].

A significant number of research papers have been devoted to the problem of DNA strands design. Due to space limitations we only cite a few [2, 8, 20, 24]. Many of these papers, such as [14, 15], rely on computational methods where the shuffle and deletion on trajectories are used to characterize undesired bonds. In this section we propose a new formalization of undesired bonds of DNA strands with irregularities (bulges). We show how the operations on trajectories can be effectively used to characterize such bonds and to solve some elementary problems of the DNA strand design.

In the remainder of this section we represent the single-stranded DNA molecules by strings over the *DNA alphabet* $\Delta = \{A, C, T, G\}$. Therefore, some more formal language prerequisites are necessary.

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. An involution of Σ can be extended to either a morphism or an antimorphism

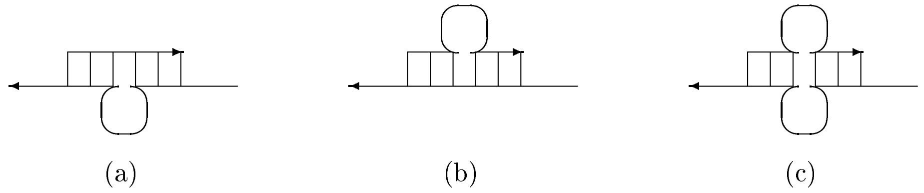

Figure 1: Three types of undesired bonds corresponding to Definition 7. Horizontal lines and bulges represent DNA strands. Vertical lines represent bonds between complementary nucleotides.

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 $a_1a_2\ldots a_k$ onto $a_k\ldots a_2a_1$, where $a_i \in \Sigma$, $1 \le i \le k$.

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*. By convention, a word $w = a_1a_2\ldots a_n$ in Δ^* will signify the DNA single strand $5' - a_1a_2\ldots a_n - 3'$. According to this convention, single strands $w_1, w_2 \in \Delta^*$ are complementary and can stick together via hydrogen bonds *iff* $w_1 = \tau(w_2)$. In the following definitions, however, we allow for an arbitrary alphabet Σ and an arbitrary involution θ over Σ.

Definition 6 *We define the following functions* $\Sigma^* \longrightarrow 2^{\Sigma^*}$:

$$\begin{aligned}
\mathrm{Ins}(u) &= \{u_1vu_2 \mid v \in \Sigma^*,\ u_1, u_2 \in \Sigma^*,\ u = u_1u_2\};\\
\mathrm{Del}(u) &= \{u_1u_3 \mid u = u_1u_2u_3,\ u_i \in \Sigma^*,\ 1 \le i \le 3\};\\
\mathrm{Subs}(u) &= \{u_1u_2'u_3 \mid u = u_1u_2u_3,\ u_1, u_2, u_3 \in \Sigma^*,\ |u_2| = |u_2'| = H(u_2, u_2')\}.
\end{aligned}$$

We note that in [16] and some other papers we have used a similar notation ins, del and Sub. However, the mappings corresponding to this notation differ from the above functions Ins, Del and Subs.

Definition 7 *A language* $L \subseteq \Sigma^+$ *is called*

$$\begin{aligned}
&\theta\textit{-ins-compliant} \quad \textit{iff} \quad \forall w \in L,\ x, y \in \Sigma^*,\ xzy \in L,\ z \in \mathrm{Ins}(\theta(w)) \Rightarrow xy = \lambda;\\
&\theta\textit{-del-compliant} \quad \textit{iff} \quad \forall w \in L,\ x, y \in \Sigma^*,\ xzy \in L,\ z \in \mathrm{Del}(\theta(w)) \Rightarrow xy = \lambda;\\
&\theta\textit{-sub-compliant} \quad \textit{iff} \quad \forall w \in L,\ x, y \in \Sigma^*,\ xzy \in L,\ z \in \mathrm{Subs}(\theta(w)) \Rightarrow xy = \lambda.
\end{aligned}$$

Intuitively, if a language L of single DNA strands is θ-ins-compliant (θ-del-compliant, θ-sub-compliant), then the strands in L cannot create bonds like those in Fig. 1 (a) (or (b), (c), respectively). The above definition is motivated as follows: the molecules depicted in Fig. 1 have "sticky" ends which can potentially react with other molecules, producing undesired bonds. If, however, the condition $xy = \lambda$ is satisfied, no sticky ends are present.

Below we characterize the compliance properties via operations on trajectories. Some technical lemmata will be useful.

Lemma 8 *(i)* $\mathrm{Ins}(u) = u \sqcup\!\sqcup_{0^*1^*0^*} \Sigma^*$;

(ii) $\mathrm{Del}(u) = u \rightsquigarrow_{0^*1^*0^*} \Sigma^*$;

(iii) $\mathrm{Subs}(u) = u \bowtie_{0^*1^*0^*} \Sigma^*$.

Lemma 9 *For arbitrary* $x, y \in \Sigma^*$,

(i) $x \in \mathrm{Ins}(y)$ *iff* $y \in \mathrm{Del}(x)$;

(ii) $x \in \mathrm{Subs}(y)$ *iff* $y \in \mathrm{Subs}(x)$.

Proof. Follows by Lemmata 2, 4 and 8. □

In [14, 15], a general framework of *bond-free language property* has been presented. Within this framework we have characterized a sequence of various types of undesired bonds between single DNA strands. We recall the definition.

Definition 8 *A language property* $\mathcal{P}$ *is called a* bond-free property of degree 2 *if there exist sets of trajectories* $T_{\mathrm{lo}}, T_{\mathrm{up}}$ *and an involution* θ *such that for an arbitrary* $L \subseteq \Sigma^*$, L *satisfies* $\mathcal{P}$ *iff*

$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*, (w \sqcup\!\sqcup_{T_{\mathrm{lo}}} x \cap L \neq \emptyset,\ w \sqcup\!\sqcup_{T_{\mathrm{up}}} y \cap \theta(L) \neq \emptyset) \Rightarrow\ xy = \lambda.$$

Intuitively, w and $\theta(w)$ are complementary parts of two DNA strands. The operations $w \sqcup\!\sqcup_{T_{\mathrm{lo}}} x$ and $w \sqcup\!\sqcup_{T_{\mathrm{up}}} x$ characterize the lower and the upper strand, respectively. We show now how to generalize the concept of the bond-free property to cover also the bonds described in Fig. 1.

Theorem 9 *Consider the sets of trajectories* $T_{\mathrm{lo}} = 0^*$ *and* $T_{\mathrm{up}} = 0^*1^*0^*$. *A language* $L \subseteq \Sigma^+$ *is* θ*-ins-compliant (*θ*-del-compliant,* θ*-sub-compliant, respectively) iff*

$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*, (w \sqcup\!\sqcup_{T_{\mathrm{lo}}} x \cap L \neq \emptyset,\ w \sqcup\!\sqcup_{T_{\mathrm{up}}} y \cap \theta(\psi(L)) \neq \emptyset) \Rightarrow xy = \lambda,$$

where the mapping ψ=Del *for* θ*-ins-compliance,* ψ=Ins *for* θ*-del-compliance and* ψ=Subs *for* θ*-sub-compliance.*

Proof. Consider the property of θ-ins-compliance. By Definition 7, the fact that a language L is θ-ins-compliant is equivalent with each of the following statements:

$$\forall w \in \Sigma^+,\ (\exists x, y, z \in \Sigma^*,\ xy \neq \lambda,\ xzy \in L,\ z \in \mathrm{Ins}(\theta(w))) \Rightarrow w \notin L$$
$$\forall w \in \Sigma^+,\ (\exists x, y \in \Sigma^*,\ xy \neq \lambda,\ \{x\}\mathrm{Ins}(\theta(w))\{y\} \cap L \neq \emptyset) \Rightarrow w \notin L$$
$$\forall w \in \Sigma^+,\ ((\Sigma^*\mathrm{Ins}(\theta(w))\Sigma^+ \cup \Sigma^+\mathrm{Ins}(\theta(w))\Sigma^*) \cap L \neq \emptyset) \Rightarrow w \notin L$$

Now observe that $\Sigma^*\mathrm{Ins}(\theta(w))\Sigma^+ = \mathrm{Ins}(\theta(\Sigma^*\{w\}\Sigma^+))$ and similarly $\Sigma^+\mathrm{Ins}(\theta(w))\Sigma^* = \mathrm{Ins}(\theta(\Sigma^+\{w\}\Sigma^*))$. Therefore, L is θ-ins-compliant *iff*

$$\forall w \in \Sigma^+,\ ((\mathrm{Ins}(\theta(\Sigma^*\{w\}\Sigma^+)) \cup \mathrm{Ins}(\theta(\Sigma^+\{w\}\Sigma^*))) \cap L \neq \emptyset) \Rightarrow w \notin L$$
$$\forall w \in \Sigma^+,\ (\mathrm{Ins}(\theta(\Sigma^*\{w\}\Sigma^+ \cup \Sigma^+\{w\}\Sigma^*)) \cap L \neq \emptyset) \Rightarrow w \notin L$$
$$\forall w \in \Sigma^+,\ ((\Sigma^*\{w\}\Sigma^+ \cup \Sigma^+\{w\}\Sigma^*) \cap \theta(\mathrm{Del}(L)) \neq \emptyset) \Rightarrow w \notin L \quad \text{(by Lemma 9)}$$
$$\forall w \in \Sigma^+,\ (\exists x, y \in \Sigma^*,\ xy \neq \lambda,\ \{xwy\} \cap \theta(\mathrm{Del}(L)) \neq \emptyset) \Rightarrow w \notin L$$
$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*,\ (w \in L,\ \{xwy\} \cap \theta(\mathrm{Del}(L)) \neq \emptyset) \Rightarrow xy = \lambda$$
$$\forall w \in \Sigma^+,\ x, y \in \Sigma^*, (w \sqcup\!\sqcup_{T_{\mathrm{lo}}} x \cap L \neq \emptyset,\ w \sqcup\!\sqcup_{T_{\mathrm{up}}} y \cap \theta(\mathrm{Ins}(L)) \neq \emptyset) \Rightarrow xy = \lambda$$

The proof for the case of θ-del-compliance and θ-sub-compliance is analogous. $\square$

The expressions in Definition 8 and Theorem 9 are identical except that $\theta(L)$ is replaced by $\theta(\psi(L))$. This allows us to apply techniques from [15] in the case of θ-ins-compliant, θ-del-compliant or θ-sub-compliant languages. Particularly, we can decide in a quadratic time whether a given regular set of DNA strands satisfies these θ-compliance conditions.

Theorem 10 *The following problem is decidable in time $\mathcal{O}(|A|^2)$:*

Input: *an NFA A.*

Output: *Yes/No depending on whether $L(A)$ is θ-ins-compliant, θ-del-compliant or θ-sub-compliant, respectively.*

Proof. Given an NFA A, by Lemmata 3, 6 and Theorem 1 we can construct a (λ-) NFA A' of the size $\mathcal{O}(|A|)$, accepting $\mathrm{Ins}(L(A))$, $\mathrm{Del}(L(A))$ or $\mathrm{Subs}(L(A))$, respectively. The rest of the proof follows directly by Theorems 4.4, 4.7 and Corollary 4.5 in [15]. It is necessary only to replace all the occurrences of $\theta(L)$ by $\theta(\mathrm{Del}(L))$, $\theta(\mathrm{Ins}(L))$ or $\theta(\mathrm{Subs}(L))$, respectively, in Theorems 4.4 and 4.7. In Corollary 4.5, we obtain $L \boxminus_{\mathcal{P}} \psi(L)$ instead of $L \boxminus_{\mathcal{P}} L$, where ψ=Del, Ins or Subs, respectively. $\square$

Acknowledgements

This research was 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 No. 201/02/P079 to P.S.

References

1. L. Adleman, Molecular computation of solutions to combinatorial problems. *Science* **266** (1994), 1021–1024.
2. M. Arita, S. Kobayashi, DNA sequence design using templates. *New Generation Computing* **20** (2002), 263–277.
3. M. Domaratzki, Deletion along trajectories. *Theoretical Computer Science* **320** (2004), 293–313.
4. M. Domaratzki, Trajectory-based codes. *Acta Informatica* **40 (6–7)** (2004), 491–527.
5. M. Domaratzki, Trajectory-based embedding orders. *Fundamenta Informaticae* **59 (4)** (2004), 349–363.
6. M. Domaratzki, K. Salomaa, Decidability of trajectory-based equations. In J. Fiala, V. Koubek and J. Kratochvl (Eds.), *Mathematical Foundations of Computer Science 2004: 29th International Symposium.* Berlin: LNCS **3153**, 2004, pp. 723–734.
7. M. Domaratzki, A. Mateescu, K. Salomaa, S. Yu, Deletion on Trajectories and Commutative Closure. In T. Harju and J. Karhumaki (Eds.), *WORDS'03: 4th International Conference on Combinatorics on Words.* TUCS General Publication No. 27, Aug. 2003, pp. 309–319.
8. 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.
9. M. Ito, L. Kari, G. Thierrin, Shuffle and scattered deletion closure of languages. *Theoretical Computer Science* **245** (2000), 115–133.

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

11. L. Kari, S. Konstantinidis, Language equations, maximality and error detection. To appear in *Journal of Computer and System Sciences.*

12. L. Kari, S. Konstantinidis, S. Perron, G. Wozniak, J. Xu, *Finite-state error/edit-systems and difference measures for languages and words.* Dept. of Math. and Computing Sci. Tech. Report No. 2003-01, Saint Mary's University, Canada, 2003.

13. L. Kari, P. Sosík, Aspects of shuffle and deletion on trajectories. To appear in *Theoretical Computer Science.*

14. L. Kari, S. Konstantinidis, P. Sosík, Preventing undesirable bonds between DNA codewords. In C. Feretti, G. Mauri, C. Zandron (Eds.), *DNA 10, Tenth International Meeting on DNA Computing.* University of Milano–Bicocca, 2004, pp. 375–384.

15. L. Kari, S. Konstantinidis, P. Sosík, On properties of bond-free DNA languages. To appear in *Theoretical Computer Science.*

16. L. Kari, G. Thierrin, Languages and monoids with disjunctive identity. *Collect. Math.* **46** (1995), 97–107.

17. S. Konstantinidis, An algebra of discrete channels that involve combinations of three basic error types. *Information and Computation* **167** (2001), 120–131.

18. S. Konstantinidis, Transducers and the properties of error detection, error correction and finite-delay decodability. *J. Universal Comp. Science* **8** (2002), 278–291.

19. E. L. Leiss, *Language Equations.* Springer-Verlag, New York, 1999.

20. A. Marathe, A.E. Condon, R.M. Corn, On combinatorial DNA words design. *J. Computational Biology* **8:3** (2001), 201-220.

21. A. Mateescu, Splicing on routes: a framework of DNA computation. In C. Calude, J. Casti, and M. Dinneen (Eds.), *Unconvential Models of Computation*, Berlin: Springer-Verlag, 1998, pp. 273–285.

22. A. Mateescu, A. Salomaa, Nondeterministic trajectories. In W. Brauer, H. Ehrig, J. Karhumäki and A. Salomaa (Eds.), *Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg*, LNCS **2300** (2002), pp. 96–106.

23. A. Mateescu, K. Salomaa, S. Yu, On fairness of many−dimensional trajectories. *J. Automata, Languages and Combinatorics* **5** (2000), 145–157.

24. G. Mauri, C. Ferretti, Word design for molecular computing: a survey. In J. Chen and J.H. Reif (Eds.), *DNA Computing, 9th International Workshop on DNA Based Computers*, LNCS 2943 (2004), pp. 37–46.

25. A. Mateescu, G. Rozenberg, A. Salomaa, Shuffle on trajectories: syntactic constraints. *Theoretical Computer Science* **197** (1998), 1–56.

26. G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages,* Springer-Verlag, Berlin, 1997.