Transducer Descriptions of DNA Code Properties and Undecidability of Antimorphic Problems[☆]

Lila Kari[a,b], Stavros Konstantinidis[c,*], Steffen Kopecki[b,c]

[a]*School of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada*
[b]*Dept. Computer Science, The University of Western Ontario, London, ON, N6A 5B7, Canada*
[c]*Dept. Math. & Computing Sci., Saint Mary's University, 923 Robbie St, Halifax, NS, B3H 3C3, Canada*

Abstract

This work concerns formal descriptions of *DNA* code properties and related (un)decidability questions. This line of research allows us to give a property as input to an algorithm, in addition to any regular language, which can then answer questions about the language and the property. Here we define DNA code properties via transducers and show that this method is strictly more expressive than that of regular trajectories, without sacrificing the efficiency of deciding the satisfaction question. We also show that the maximality question can be undecidable. Our undecidability results hold not only for the fixed DNA involution but also for any fixed antimorphic permutation. Moreover, we also show the undecidability of the antimorphic version of the Post Correspondence Problem, for any fixed antimorphic permutation.

Keywords: codes, DNA properties, trajectories, transducers, undecidability

1. Introduction

The study of *formal* methods for describing independent language properties (widely known as code properties) provides tools that allow one to give a property as input to an algorithm and answer questions about this property. Examples of such properties include *classic* ones [2–5] like prefix codes, bifix codes, and various error-detecting languages, as well as DNA code properties [6–17] like θ-nonoverlapping and θ-compliant languages. A formal description method should be expressive enough to allow one to describe many desirable properties. Examples of formal methods for describing *classic* code properties

[☆]This paper is an expanded version of [1]. Research supported by: NSERC Discovery Grants R2824A01 (L. Kari) and 220259 (S. Konstantinidis), and University of Western Ontario Internal Grant.

[*]Corresponding author

Email addresses: lila@csd.uwo.ca (Lila Kari), s.konstantinidis@smu.ca (Stavros Konstantinidis), steffen@csd.uwo.ca (Steffen Kopecki) are the implicational conditions method of [18], the trajectories method of [19], and the transducer methods of [20]. The latter two have been implemented to some extent in the Python package FAdo [21]. A formal method for describing DNA code properties is the method of trajectory DNA code properties [12, 22].

Typical questions about properties are the following:

Satisfaction problem: given the description of a property and the description of a regular language, decide whether the language satisfies the property.

Maximality problem: given the description of a property and the description of a regular language that satisfies the property, decide whether the language is maximal with respect to the given property.

Construction problem: given the description of a property and a positive integer n, find a language of n words (if possible) satisfying the given property.

In the above problems regular languages are described via (nondeterministic) finite automata (NFA). Depending on the context, properties are described via trajectory regular expressions or transducer expressions. The satisfaction problem is the most basic one and can be answered usually in efficient polynomial time. The maximality problem as stated above can be decidable, in which case it is normally PSPACE-hard. For existing transducer and trajectory properties, both problems can be answered using the online (formal) language server LaSer [23], which relies on FAdo. LaSer allows users to enter the desired property and language, and returns either the answer in real time (online mode), or it returns a Python program that computes the desired answer if executed at the user's site (program generation mode). For the construction problem a simple statistical algorithm is included in FAdo, but we think that this problem is far from being well-understood.

When it comes to DNA code properties in the context of formal languages, there have been a few algorithms and implementations concerning specific such properties—for example [11, 17, 24, 25]. These provide valuable insights and contribute to the maturity of the research on DNA code properties. Most of the implementations concern the construction problem for sets of words of fixed length. In [17], the authors consider the efficient implementation of two DNA code properties as well as the UD code (Unique Decipherability code) property. The topic of DNA code properties is active and relevant, as there are laboratory experiments involving computations on DNA molecules [26].

The *general objective* of this research is to develop methods for formally describing DNA code properties that would allow one to express various combinations of such properties and be able to get answers to questions about these properties in an actual implementation. While the satisfaction and construction questions are important from both the theoretical and practical viewpoints, the maximality question is at least of theoretical interest and a classic problem in the theory of codes. The contributions of this work are as follows:

1. The definition of a new simple formal method for describing many DNA code properties, called *θ-transducer properties*, some of which *cannot* be

described by the existing transducer and trajectory methods for classic code properties; see Sect. 3. These methods are closed under intersection of code properties. This means that if two properties can be described within the method then also the combined property can be described within the method. This outcome is important as in practice it is desirable that languages satisfy more than one property.

2. The demonstration that the new method of transducer DNA code properties is properly more expressive than the method of trajectories; see Sect. 4. Also the demonstration that the satisfaction problem is decidable for all θ-transducer properties (Sect. 5) in such a way that when these properties are trajectory DNA code properties the efficiency of the satisfaction algorithm is asymptotically the same (Remark 22).

3. The demonstration that the maximality problem can be decidable for some of these properties but undecidable for some others; see Sect. 6.

4. The demonstration that some classic undecidable problems (like *PCP*, the Post Correspondence Problem) remain undecidable when rephrased in terms of *any fixed* (anti-)morphic permutation θ of the alphabet, with the case $\theta = \mathrm{id}$ corresponding to these classic problems, where id is the *(morphic) identity*; see Sect. 7.

5. The presentation of a natural hierarchy of DNA properties which are all θ-transducer properties; see Sect. 8. This hierarchy generalizes the concept of bond-free properties in [7–9].

Even though our main motivation is the description of DNA-related properties, we follow the more general approach which considers properties described by transducers involving a fixed (anti-)morphic permutation θ; again, the classical transducer properties are obtained by letting $\theta = \mathrm{id}$. As it turns out, in the case when θ is morphic all questions that we consider in this paper can be answered analogously to the solutions for the classical case of $\theta = \mathrm{id}$. Therefore, we focus on the transducer properties involving antimorphic permutations in this paper.

2. Basic Notions and Background Information

We assume the reader to be familiar with the fundamental concepts of language theory; see e. g., [27, 28]. In Sect. 2.1–2.3, we lay down our notation for formal languages, (anti-)morphic permutations and involutions, automata and transducers, and trajectories and related word operations. In Sect. 2.4 we recall the method of transducers for describing classic code properties, and in Sect. 2.5 we recall the method of trajectories for describing DNA code properties. In Sect. 2.6, we describe what technical tools are used to establish the main results of the paper.

For a set S, we write $|S|$ to denote its cardinality. The symbols $\mathbb{N}$ and $\mathbb{N}_+$ denote, respectively, the set of nonnegative integers and the set of positive integers.

2.1. Formal Languages and (Anti-)morphic Permutations

An *alphabet* A is a finite set of *letters*; A^* is the set of all words or strings over A; ε denotes the *empty word*; and $A^+ = A^* \setminus \{\varepsilon\}$. A *language* L over A is a subset $L \subseteq A^*$; the *complement* L^c of L is the language $A^* \setminus L$. For an integer $m \in \mathbb{N}$ we let $A^{\leq m}$ denote the set of words whose length is at most m; i. e., $A^{\leq m} = \bigcup_{i \leq m} A^i$. The *DNA alphabet* is $\Delta = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Often it is convenient to consider the *generic alphabet* $A_k = \{0, 1, \ldots, k-1\}$ *of size* k rather than a general alphabet; note that $A_2 \subseteq A_3 \subseteq A_4 \subseteq \cdots$. Throughout this paper we only consider alphabets with at least two letters because our investigations would become trivial over unary alphabets.

Let $w \in A^*$ be a word. Unless confusion arises, by w we also denote the singleton language $\{w\}$, e. g., $L \cup w$ means $L \cup \{w\}$. If $w = xyz$ for some $x, y, z \in A^*$, then x, y, and z are called *prefix*, *infix* (or *factor*), and *suffix* of w, respectively. For a language $L \subseteq A^*$, the set $\operatorname{Pref}(L) = \{x \in A^* \mid \exists y \in A^* \colon xy \in L\}$ denotes the language containing all prefixes of words in L. If $w = a_1 a_2 \cdots a_n$ for letters $a_1, a_2, \ldots, a_n \in A$, then $|w| = n$ is the *length of* w; for $b \in A$, $|w|_b = |\{i \mid a_i = b, 1 \leq i \leq n\}|$ is the tally of b occurring in w; the i-th letter of w is $w_{[i]} = a_i$ for $1 \leq i \leq n$; the infix of w from the i-th letter to the j-th letter is $w_{[i;j]} = a_i a_{i+1} \cdots a_j$ for $1 \leq i \leq j \leq n$; and the *reverse* of w is $w^R = a_n a_{n-1} \cdots a_1$.

Consider a generic alphabet A_k with $k \geq 2$. The *identity* function on A_k is denoted by id_k; when the alphabet is clear from the context, the index k is omitted. For a *permutation* (or bijection) $\theta \colon A_k \to A_k$, the permutation θ^{-1} is the *inverse* of θ as usual; i. e., $\theta \circ \theta^{-1} = \mathrm{id}_k$ ("$\circ$" is the composition of two functions $(g \circ h)(x) = g(h(x))$ for all x). For $i \in \mathbb{Z}$, the permutation θ^i is the *i-fold composition* of θ; i. e., $\theta^0 = \mathrm{id}_k$, $\theta^i = \theta \circ \theta^{i-1}$, and $\theta^{-i} = (\theta^i)^{-1} = (\theta^{-1})^i$ for $i > 0$. There exists a number n, called the *order* of θ, such that $\theta^n = \mathrm{id}_k$. An *involution* θ is a permutation of order 2; i. e., $\theta = \theta^{-1}$.

A permutation θ over A_k can naturally be extended to operate on words in A_k^* as (a) *morphic permutation:* $\theta(uv) = \theta(u)\theta(v)$, or (b) *antimorphic permutation:* $\theta(uv) = \theta(v)\theta(u)$, for $u, v \in A_k^*$. As before, the inverse θ^{-1} of the (anti-)morphic permutation θ over A_k^* is the (anti-)morphic extension of the permutation θ^{-1} over A_k^*. Note that the composition of two antimorphic or two morphic permutations yields a morphic permutation, whereas the composition of a morphic and an antimorphic permutation yields an antimorphic permutation. Therefore, if θ is an antimorphic permutation, then θ^i is morphic if and only if i is even. The identity id_k always denotes the morphic extension of id_k while the antimorphic extension of id_k, called the *mirror image* or reverse, is usually denoted by the exponent ^{R}.

Example 1. The *DNA involution*, denoted as δ, is an antimorphic involution on $\Delta = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ such that $\delta(\mathtt{A}) = \mathtt{T}$ and $\delta(\mathtt{C}) = \mathtt{G}$, which implies $\delta(\mathtt{T}) = \mathtt{A}$ and

$\delta(\mathtt{G}) = \mathtt{C}$.

2.2. Automata and Transducers

A nondeterministic finite automaton with empty transitions, *ε-NFA* for short, or just *automaton*, is a quintuple $\mathbf{a} = (Q, A, E, I, F)$ such that Q is the set of states, A is the alphabet, $I \subseteq Q$ is the set of start (or initial) states, $F \subseteq Q$ is the set of final states, and $E \subseteq Q \times (A \cup \varepsilon) \times Q$ is the finite set of edges (or transitions). Let (p, x, q) be an edge of $\mathbf{a}$. Then x is called the *label* of the edge, and we say that p has an *outgoing* edge (with label x). We also use the notation

$$p \xrightarrow{x} q$$

for the edge (p, x, q). The ε-NFA $\mathbf{a}$ is called an *NFA*, if no edge label is empty, that is, $E \subseteq Q \times A \times Q$. If in addition $\mathbf{a}$ is such that $|I| = 1$ and there is no state having two outgoing edges with the same label, then $\mathbf{a}$ is called a deterministic finite automaton, *DFA* for short. A *path* of $\mathbf{a}$ is a finite sequence of edges of the form

$$(p_0, x_1, p_1), (p_1, x_2, p_2), \ldots, (p_{\ell-1}, x_\ell, p_\ell),$$

for some nonnegative integer ℓ. The word $x_1 \cdots x_\ell$ is called the *label* of the path. We write $p_0 \xrightarrow{x}{}^{*} p_\ell$ to indicate that there is a path with label x from p_0 to p_ℓ. A path as above is called *accepting* if p_0 is the start state and p_ℓ is a final state. The *language accepted* by $\mathbf{a}$, denoted as $L(\mathbf{a})$, is the set of labels of all the accepting paths of $\mathbf{a}$. The automaton $\mathbf{a}$ is called *trim*, if every state appears in some accepting path of $\mathbf{a}$.

A *transducer* $\mathbf{t}$ is a non-deterministic finite state automaton with output; see e. g., [29, 30]. In general, a transducer can have an output alphabet B which is different from its input alphabet A. In this paper however, we only consider transducers where the input alphabet coincides with the output alphabet, $A = B$, which leads to the following simplified definition: a transducer is a quintuple $\mathbf{t} = (Q, A, E, I, F)$, where A is the input and output alphabet, Q is a finite set of states, E is a set of edges between states from Q which are labeled by word pairs $(u, v) \in A^* \times A^*$, I is the set of initial states, and F the set of final states. For an edge label (u, v), the word u is called *input*, while the word v is called *output*. The transducer $\mathbf{t}$ *realizes* the pair $(x, y) \in A^* \times A^*$, if x is formed by concatenating the inputs, and y is formed by concatenating the outputs of the labels in a path of $\mathbf{t}$ from an initial to a final state. The transducer $\mathbf{t}$ *realizes* the relation $R(\mathbf{t})$, which is the set of all pairs $(x, y) \in A^* \times A^*$ realized by $\mathbf{t}$. If $\mathbf{t}$ realizes (x, y) then we write $y \in \mathbf{t}(x)$. The set $\mathbf{t}(x)$ consists of all possible outputs of $\mathbf{t}$ on input x. The *domain* of $\mathbf{t}$ is the set of all words x such that $\mathbf{t}(x) \neq \emptyset$. For a language L we naturally extend the notation $\mathbf{t}(x)$ such that

$$\mathbf{t}(L) = \cup_{x \in L} \mathbf{t}(x). \tag{1}$$

The *size* $|\mathbf{m}|$ of a finite-state machine (automaton or transducer) $\mathbf{m}$ is the sum of: the number of states, the number of edges, and the sum of the lengths of all labels on the edges.

It is well known that for two regular languages R_1, R_2 there exists a transducer $\mathbf{t}$ that realizes the relation $R_1 \times R_2$; i.e., $\mathbf{t}$ realizes (x, y) if and only if $x \in R_1$ and $y \in R_2$. The transducer $\mathbf{t}^{-1}$ is the *inverse of* $\mathbf{t}$; that is,$x \in \mathbf{t}^{-1}(y)$ if and only if $y \in \mathbf{t}(x)$ for all words x, y. Note that $\mathbf{t}^{-1}$ is of size $(|\mathbf{t}|)$ and obtained from $\mathbf{t}$ by simply swapping the input with the output word on each edge in $\mathbf{t}$. Let $\mathbf{t}$ and $\mathbf{s}$ be transducers and $\mathbf{a}$ be an automaton. Then:

- there is a transducer $(\mathbf{t} \vee \mathbf{s})$ of size $O(|\mathbf{s}| + |\mathbf{t}|)$ realizing the relation $R(\mathbf{t}) \cup R(\mathbf{s})$ such that $(\mathbf{t} \vee \mathbf{s})(x) = \mathbf{t}(x) \cup \mathbf{s}(x)$, for all words x;
- there is a transducer $(\mathbf{t} \circ \mathbf{s})$ of size $O(|\mathbf{s}| \cdot |\mathbf{t}|)$ realizing all word pairs (x, z) for which there is a word y such that $y \in \mathbf{s}(x)$ and $z \in \mathbf{t}(y)$;
- there is a transducer $(\mathbf{t} \uparrow \mathbf{a})$ of size $O(|\mathbf{t}| \cdot |\mathbf{a}|)$ such that $y \in (\mathbf{t} \uparrow \mathbf{a})(x)$ if and only if $y \in \mathbf{t}(x) \cap L(\mathbf{a})$, for all words x, y [31];
- there is a transducer $(\mathbf{t} \downarrow \mathbf{a})$ of size $O(|\mathbf{t}| \cdot |\mathbf{a}|)$ such that $y \in (\mathbf{t} \downarrow \mathbf{a})(x)$ if and only if $y \in \mathbf{t}(x)$ and $x \in L(\mathbf{a})$, for all words x, y.

A transducer $\mathbf{t}$ is called *functional* if $|\mathbf{t}(x)| \leq 1$ for all $x \in A^*$. It is called *input-altering* if for all words x we have $x \notin \mathbf{t}(x)$. It is called *input-preserving* if for all words x in the domain of $\mathbf{t}$, we have that $x \in \mathbf{t}(x)$.

2.3. Language Operators, Trajectories and Related Word Operations

A *language operator* is any mapping $\mathrm{Op}\colon 2^{A^*} \to 2^{A^*}$. The prefix function Pref defined earlier is an example of a language operator. A transducer can be viewed as a language operator—see Eq. (1). Any (anti-)morphic permutation, as well as any other function, $h\colon A^* \to A^*$ over words is extended to a language operator such that for $L \subseteq A^*$

$$h(L) = \cup_{x \in L}\{h(x)\}.$$

If Op_1 and Op_2 are language operators, then $(\mathrm{Op}_1 \vee \mathrm{Op}_2)$ is the language operator such that $(\mathrm{Op}_1 \vee \mathrm{Op}_2)(X) = \mathrm{Op}_1(X) \cup \mathrm{Op}_2(X)$, for all languages X.

A *trajectory regular expression* is a regular expression over the alphabet $\{0,\ 1\}$. We shall use symbols with bars, like $\bar{a}$, to denote trajectory regular expressions. Then, $L(\bar{a})$ denotes the *language described* by the expression $\bar{a}$. A *trajectory* t is any element of $L(\bar{a})$.

Next we define the word operations $\sqcup\!\sqcup_t$ and $\rightsquigarrow_t$, called *shuffle* (or scattered insertion) and *scattered deletion*, respectively, on the trajectory t [22, 32].

Definition 2. Let x and w be words and t be a trajectory.

1. The shuffle of w and x on t, denoted by $x \sqcup\!\sqcup_t w$, is the set

$$\begin{aligned} x \sqcup\!\sqcup_t w = \quad & \{x_1 w_1 \cdots x_k w_k \mid x = x_1 \cdots x_k, w = w_1 \cdots w_k, t = 0^{i_1} 1^{j_1} \cdots 0^{i_k} 1^{j_k}, \\ & \qquad \text{where } |x_r| = i_r \text{ and } |w_r| = j_r, \text{ for all } r \text{ with } 1 \leq r \leq k\}. \end{aligned}$$

2. The deletion of w from x on t, denoted by $x \rightsquigarrow_t w$, is the set

$$x \rightsquigarrow_t w = \{x_1 \cdots x_k \mid x = x_1 w_1 \cdots x_k w_k, w = w_1 \cdots w_k, t = 0^{i_1} 1^{j_1} \cdots 0^{i_k} 1^{j_k}, \text{ where } |x_r| = i_r \text{ and } |w_r| = j_r, \text{ for all } r \text{ with } 1 \le r \le k\}.$$

In the above definition, the set $x \sqcup\!\sqcup_t w$ consists of exactly one word if $|t|_0 = |x|$ and $|t|_1 = |w|$, and it is empty otherwise. For example, $1122 \sqcup\!\sqcup_{001010} 34 = 113242$. The set $x \rightsquigarrow_t w$ is either of the form $\{y\}$ such that the word y is of length $|t|_0 = |x| - |w|$ and satisfies $\{x\} = y \sqcup\!\sqcup_t w$, or the empty set otherwise. For example, $113242 \rightsquigarrow_{001010} 34 = 1122$. For any languages X, W and trajectory expression $\bar{a}$, we have that

$$X \sqcup\!\sqcup_{\bar{a}} W = \bigcup_{x \in X, w \in W, t \in L(\bar{a})} x \sqcup\!\sqcup_t w$$

$$X \rightsquigarrow_{\bar{a}} W = \bigcup_{x \in X, w \in W, t \in L(\bar{a})} x \rightsquigarrow_t w.$$

2.4. Describing Classic Code Properties by Transducers and Trajectories

A *(language) property* $\mathcal{P}$ is any set of languages. A language L *satisfies* $\mathcal{P}$, or *has* $\mathcal{P}$, if $L \in \mathcal{P}$. Here by a property $\mathcal{P}$ we mean an *(n-)independence* in the sense of [4]: there exists $n \in \mathbb{N}_+ \cup \{\aleph_0\}$ such that a language L satisfies $\mathcal{P}$ if and only if all nonempty subsets $L' \subseteq L$ of cardinality less than n satisfy $\mathcal{P}$. A language L satisfying $\mathcal{P}$ is *maximal* (with respect to $\mathcal{P}$) if for every word $w \in L^c$ we have $L \cup w$ does not satisfy $\mathcal{P}$—note that, for any independence $\mathcal{P}$, every language in $\mathcal{P}$ is a subset of a maximal language in $\mathcal{P}$ [4]. To our knowledge all code related properties in the literature, including DNA code properties, are independence properties. As we shall see further below the focus of this work is on 3-independence properties. These can also be viewed as independent with respect to a binary relation in the sense of [2].

Definition 3 (Existing transducer properties [20])**.** An input-altering transducer $\mathbf{t}$ *describes the property* that consists of all languages L such that

$$\mathbf{t}(L) \cap L = \emptyset. \tag{2}$$

An input-preserving transducer $\mathbf{t}$ *describes the property* that consists of all languages L such that

$$w \notin \mathbf{t}(L \setminus w), \quad \text{for all } w \in L. \tag{3}$$

A property is called an *input-altering* (resp. *input-preserving*) *transducer property*, if it is described by an input-altering (resp. input-preserving) transducer.

Note that every input-altering transducer property is also an input-preserving transducer property. Input-altering transducers can be used to describe properties like prefix codes, bifix codes, and hypercodes. Input-preserving transducers are intended for error-detecting properties, where in fact the transducer plays the role of the communication channel. Figure 1 shows a couple of examples.

A language L is a prefix code, if $v, vx \in L$ implies $x = \varepsilon$; it is 1-substitution error-detecting, if $v, w \in L$ implies that $|v| \neq |w|$ or $v_{[i]} \neq w_{[i]}$ for some index i, that is, v cannot be converted to w by substituting a symbol $v_{[i]}$ of v with a different one.

Convention about Figures: in this and the following transducer figures, an arrow with label (a, a) represents a set of edges with labels (a, a) for all $a \in A$; and similarly for an arrow with label (a, ε). An arrow with label (a, b) represents a set of edges with labels (a, b) for all $a, b \in A$ with $a \neq b$.

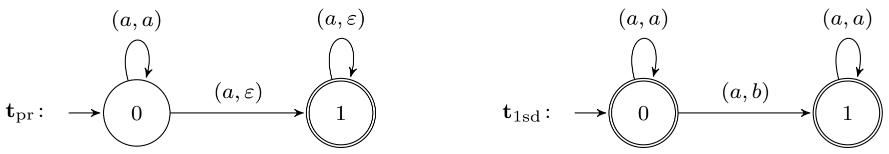


Figure 1: The left transducer is input-altering and describes the prefix codes: on input x it outputs any proper prefix of x. The right transducer is input-preserving and describes the 1-substitution error-detecting languages: on input x it outputs either x or any word differing from x in exactly one position.

Using FAdo format the transducer $\mathbf{t}_{\mathrm{pr}}$ can be specified by the following string, assuming alphabet {`a`, `b`}

```
@Transducer 1 * 0\n0 a a 0\n0 b b 0\n
0 a @epsilon 1\n0 b @epsilon 1\n
1 a @epsilon 1\n1 b @epsilon 1\n
```

Above, `\n` is the new-line character, so the string consists of 7 lines. The first line specifies the FAdo object being described (a transducer), the final states (1 here), and the initial states after the special character *. Each of the next 6 lines specifies an edge. For example, the second line specifies the edge from state 0 to state 0 with label (`a`, `a`), and the last line specifies the edge from state 1 to state 1 with label (`b`, ε)—see [21] for more details.

Many input-altering transducer properties can be described in a simpler manner by trajectory regular expressions [19]. For example, the expression 0^*1^* describes prefix codes and the expression $1^*0^*1^*$ describes infix codes—L is an infix code if $v, xvy \in L$ implies $x = y = \varepsilon$. On the other hand, there are natural input-altering transducer properties that cannot be described by trajectory regular expressions [20].

2.5. Describing DNA-related Properties by Trajectories

In [6–17] the authors consider numerous properties of languages inspired by reliability issues in DNA computing. We state three of these properties below. In Sect. 8 we present a hierarchy of DNA properties which generalizes some of the DNA code properties presented in [7–9]. Let θ be an antimorphic permutation over A_k^*. Recall that, in the DNA setting, $\theta = \delta$ is an involution, and therefore, we have $\theta^2 = \mathrm{id}$.

(A) A language L is *θ-nonoverlapping* if $L \cap \theta(L) = \emptyset$.

(B) L is *θ-compliant* if $\forall w \in \theta(L), x, y \in A_k^*\colon xwy \in L \implies xy = \varepsilon$.

(C) L is *strictly θ-compliant* if it is θ-nonoverlapping and θ-compliant.

Many of the existing DNA code properties can be modelled using the concept of a bond-free property, first defined in [12] and later rephrased in [22] in terms of trajectories. We follow the formulation in [22]. Let $\bar{e} = (\bar{e}_1, \bar{e}_2)$, where $\bar{e}_1$ and $\bar{e}_2$ are two regular trajectory expressions. First, we define the following language operators.

$$\Phi_{\bar{e}}(L) = (((L \rightsquigarrow_{\bar{e}_1} A^+) \cap A^+) \sqcup\!\sqcup_{\bar{e}_2} A^*) \cup (((L \rightsquigarrow_{\bar{e}_1} A^*) \cap A^+) \sqcup\!\sqcup_{\bar{e}_2} A^+) \quad (4)$$
$$\Phi^{\mathbf{s}}_{\bar{e}}(L) = ((L \rightsquigarrow_{\bar{e}_1} A^*) \cap A^+) \sqcup\!\sqcup_{\bar{e}_2} A^* \quad (5)$$

Definition 4. ([22]) Let θ be an involution and $\bar{e}_1, \bar{e}_2$ be two regular trajectory expressions. The *bond-free property described by* $(\bar{e}_1, \bar{e}_2)$ is

$$\mathcal{B}_\theta(\bar{e}_1, \bar{e}_2) = \{L \subseteq A^* \mid \theta(L) \cap \Phi_{\bar{e}_1,\bar{e}_2}(L) = \emptyset\}. \quad (6)$$

The *strictly bond-free property described by* $(\bar{e}_1, \bar{e}_2)$ is

$$\mathcal{B}^{\mathbf{s}}_\theta(\bar{e}_1, \bar{e}_2) = \{L \subseteq A^* \mid \theta(L) \cap \Phi^{\mathbf{s}}_{\bar{e}_1,\bar{e}_2}(L) = \emptyset\}. \quad (7)$$

A *regular θ-trajectory property* is a bond-free property described by $(\bar{e}_1, \bar{e}_2)$, or a strictly bond-free property described by $(\bar{e}_1, \bar{e}_2)$, for some pair $(\bar{e}_1, \bar{e}_2)$.

Example 5. The θ-compliant property is a regular θ-trajectory property in $\mathcal{B}_\theta(1^*0^+1^*, 0^+)$: deleting x and y in any xwy (according to $1^*0^+1^*$), where $w \in L$ and at least one symbol gets deleted, and then inserting nothing (according to 0^+) cannot result into a word in $\theta(L)$. The θ-nonoverlapping property is a regular θ-trajectory property in $\mathcal{B}^{\mathbf{s}}_\theta(0^+, 0^+)$: deleting nothing and then inserting nothing in any word $w \in L$ cannot result into a word in $\theta(L)$. The strictly θ-compliant property is a regular θ-trajectory property in $\mathcal{B}^{\mathbf{s}}_\theta(1^*0^+1^*, 0^+)$: deleting x and y in any xwy (according to $1^*0^+1^*$), where $w \in L$, and inserting nothing (according to 0^+) cannot result into a word in $\theta(L)$.

2.6. Technical Tools and Related Techniques

Here we describe the technical tools and techniques used to establish the main results of the paper. As stated in the introduction, we seek new methods to formally express DNA code properties in a way that one can answer algorithmic questions about these properties. In doing so, we want to make sure that the new methods are more expressive than existing ones. In particular, our proposed approach is to use transducers to describe DNA code properties (Definition 6). While transducers have been used to describe classic code properties (Definition 3), the existing approach for describing DNA code properties is based on trajectories (Definition 4). The main tools and related techniques utilized to obtain our main results are as follows.

- Trajectories and related word operators. These have been used to define DNA code properties. Also, together with standard language combinatorial tools, they are used to show that a certain natural DNA code property is not a regular θ-trajectory property (Proposition 17).

- Transducers and related operations and algorithms. Operations (such as $\circ, \vee, \uparrow, \downarrow$) combining transducers, possibly with automata, are used to show that every regular θ-trajectory property is a θ-transducer property (Theorem 16). These operations as well as algorithms for testing transducer partial identity and functionality are used to decide the satisfaction and maximality questions (Remark 20, Theorem 21, Corollary 28, Theorem 26 and related lemmas).

- Rational and recognizable relations tools. It is known that it is undecidable, given two transducers, whether the relation realized by the first is a subset of the relation realized by the second one. One the other hand, if the second one realizes a recognizable relation, then the question is decidable (Theorem 26).

- Dependence theory applied to the theory of codes [4]. Dependence theory allows us to give short proofs of basic statements about codes. For example, the fact that every θ-transducer property is a 3-independence implies that any language satisfying the property can be embedded into a maximal one. Thus, it is a legitimate question to ask whether a given language is maximal. Moreover, if a property is not 3-independence then it cannot be a θ-transducer property (Proposition 14).

- The method of reduction between decision problems. This is used to establish undecidability of new problems based on known undecidable ones. For example, we reduce the well-known Post Correspondence Problem (PCP), which is undecidable, to a certain maximality problem (Corollary 31) as well as to its θ version, for any fixed antimorphic permutation θ (Theorem 33).

- The pumping type of argument for showing that a language cannot be accepted by an automaton is adapted here to show that one cannot describe desired DNA code properties with any input-preserving transducer (Proposition 12) and a technical result about the satisfaction of certain transducer DNA code properties (Lemma 24).

3. New Transducer-based DNA-related Properties

A question that arises from the discussion in Sect. 2.4 and 2.5 is whether existing transducer-based code properties include DNA code properties. It turns out that this is not the case: for instance the δ-nonoverlapping property, which seems to be the simplest DNA code property, cannot be described by any input-preserving transducer; see Proposition 11. In this section, we define new transducer-based properties that are appropriate for DNA-related applications, we demonstrate Proposition 11, and discuss how existing DNA code properties can be described with transducers. Then, in Sect. 4 we examine the relationship between the new transducer properties and the regular θ-trajectory properties which were proposed in [22].

Let θ be an (anti-)morphic permutation and $\mathbf{t}$ be a transducer, which are both defined over the same alphabet A. The transducer $\mathbf{t}$ is called *θ-input-preserving* if for all $w \in A^+$ in the domain of $\mathbf{t}$ we have $\theta(w) \in \mathbf{t}(w)$; $\mathbf{t}$ is called *θ-input-altering* if for all $w \in A^+$ we have $\theta(w) \notin \mathbf{t}(w)$. We use the previously defined terms input-altering and input-preserving $\mathbf{t}$, respectively, when $\theta = \mathrm{id}$. Note that $\theta(w) \in \mathbf{t}(w)$ is equivalent to $w \in \theta^{-1}(\mathbf{t}(w))$ as well as $\mathbf{t}^{-1}(\theta(w)) \ni w$.

Definition 6. A transducer $\mathbf{t}$ and an (anti-)morphic permutation θ, defined over the same alphabet, *describe* 3-independent properties in two ways:

1.) *strict θ-transducer property ($\mathcal{S}$-property)*: L satisfies the property $\mathcal{S}_{\theta,\mathbf{t}}$ if

$$\theta(L) \cap \mathbf{t}(L) = \emptyset \tag{8}$$

2.) *weak θ-transducer property ($\mathcal{W}$-property)*: L satisfies the property $\mathcal{W}_{\theta,\mathbf{t}}$ if

$$\forall w \in L \colon \theta(w) \notin \mathbf{t}(L \setminus w) \tag{9}$$

Any of the properties $\mathcal{S}_{\theta,\mathbf{t}}$ or $\mathcal{W}_{\theta,\mathbf{t}}$ is called a *θ-transducer property.*

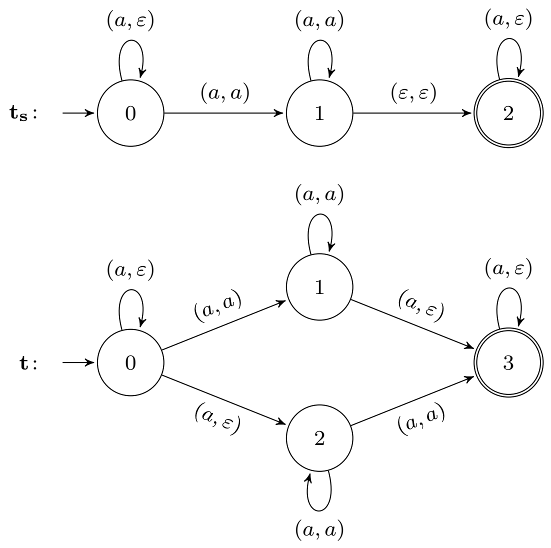


Figure 2: Together with θ, the transducer $\mathbf{t_s}$ describes the strictly θ-compliant property, and the transducer $\mathbf{t}$ describes the θ-compliant property.

Example 7. Consider the transducer $\mathbf{t}$ in Fig. 2. We explain here that $\mathbf{t}$ and θ describe the θ-compliant property. By definition, L satisfies the property if, by deleting x, y from a word $xwy \in L$ with $xy \neq \varepsilon$, we cannot get the nonempty word $w \in \theta(L)$. The transducer $\mathbf{t}$, on input xwy, can perform exactly the deletion of x, y such that at least one symbol is deleted and at least one is not deleted (thus, $w \neq \varepsilon$). Thus, $\mathbf{t}(L) \cap \theta(L) = \emptyset$ if and only if L is θ-compliant. Recall from Example 5 that this property is also described by $(\bar{e}_1, \bar{e}_2) = (1^*0^+1^*, 0^+)$ as a bond-free property. Again the idea is similar: the operator $\Phi_{\bar{e}_1,\bar{e}_2}$ performs the same action as that of $\mathbf{t}$.

Example 8. Consider the transducer $\mathbf{t_s}$ in Fig. 2. We explain here that $\mathbf{t_s}$ and θ describe the strictly θ-compliant property. By definition, L satisfies the property if, by deleting x, y from a word $xwy \in L$, we cannot get the nonempty word $w \in \theta(L)$. Note that here we allow $xy = \varepsilon$. The transducer $\mathbf{t_s}$, on input xwy, can perform exactly the deletion of x, y such that at least one symbol is not deleted (thus, $w \neq \varepsilon$). Thus, $\mathbf{t_s}(L) \cap \theta(L) = \emptyset$ if and only if L is strictly θ-compliant. Recall from Example 5 that this property is also described by $(\bar{e}_1, \bar{e}_2) = (1^*0^+1^*, 0^+)$ as a strictly bond-free property. Again the idea is similar: the operator $\Phi^{\mathbf{s}}_{\bar{e}_1,\bar{e}_2}$ performs the same action as that of $\mathbf{t_s}$.

Remark 9. For fixed $\mathbf{t}$, θ, and L, Condition (8) implies that for all $w \in L$ we have $\theta(w) \cap \mathbf{t}(L \setminus w) = \emptyset$ which is equivalent to Condition (9). In other words,

if L satisfies $\mathcal{S}_{\theta,\mathbf{t}}$, then L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$ as well.

The difference between $\mathcal{S}$-properties and $\mathcal{W}$-properties is that $\mathcal{S}_{\theta,\mathbf{t}}$ includes no language containing a word w such that $\theta(w) \in \mathbf{t}(w)$, while this case is allowed for some languages $L \in \mathcal{W}_{\theta,\mathbf{t}}$. If $\theta = \mathrm{id}$ and $\mathbf{t}$ is input-altering, or input-preserving, then the above defined properties specialize to the existing ones stated in Definition 3.

Remark 10. Every singleton language $L = \{w\}$ satisfies all properties $\mathcal{W}_{\theta,\mathbf{t}}$, as well as, all properties $\mathcal{S}_{\theta,\mathbf{t}}$ for which $\mathbf{t}$ is θ-input-altering. On the other hand, if $\mathbf{t}$ is not θ-input-altering then certain words could be excluded from all languages satisfying $\mathcal{S}_{\theta,\mathbf{t}}$. For example, if $\theta = \delta$ and $\mathbf{t}$ realizes the identity function then $\delta(\mathtt{AT}) = \mathtt{AT} \in \mathbf{t}(\mathtt{AT})$, which implies that no language satisfying $\mathcal{S}_{\theta,\mathbf{t}}$ can include the word AT.

As input-altering transducer properties are a subset of input-preserving transducer properties, we only consider the case of input-preserving transducer properties in the next two results, which demonstrate that existing transducer properties are not suitable for describing even simple DNA code properties.

Proposition 11. *The δ-nonoverlapping property is not describable by any input-preserving transducer.*

Proof. Assume that there is an input-preserving transducer $\mathbf{t}$ describing the δ-nonoverlapping property, that is, this property is $\mathcal{W}_{\mathrm{id},\mathbf{t}}$. By Remark 10, any property $\mathcal{W}_{\mathrm{id},\mathbf{t}}$ includes every singleton language; hence in particular $\{\mathtt{AT}\}$ must be δ-nonoverlapping. On the other hand, as $\mathtt{AT} = \delta(\mathtt{AT})$, we have that $\{\mathtt{AT}\}$ is not δ-nonoverlapping. □

The counter example language $\{\mathtt{AT}\}$ used to prove the previous result might seem artificial, as in practice code-related languages should have more than one element. However, the statement remains true even if we focus on languages containing more than one word. This case is handled in the next proposition.

In the proof of the proposition we assume that the transducer $\mathbf{t}$ is in normal form. It can be shown that every transducer is effectively equivalent to one in normal form [33]. A transducer $\mathbf{t}$ is in *normal form*, if the label of every edge is of the form (a, ε) or (ε, a), for some $a \in \Delta$.

Proposition 12. *There is no input-preserving transducer* $\mathbf{t}$ *that satisfies Equation* (3) *for all δ-nonoverlapping languages L having at least two elements.*

Proof. Assume the contrary, that is, there is an input-preserving transducer $\mathbf{t}$ in normal form such that for any DNA language $L \subseteq \Delta^*$ with at least two element we have

$$\delta(L) \cap L = \emptyset \quad \text{iff} \quad \forall\, u \in L: \ \mathbf{t}(u) \cap (L \setminus u) = \emptyset.$$

Assume that $\mathbf{t}$ has n states, for some positive integer n, and let $m > n$. We have that $\{\mathtt{A}^m\mathtt{C}^m, \mathtt{G}^m\mathtt{T}^m\}$ is not δ-nonoverlapping, so without loss of generality we have that $\mathtt{G}^m\mathtt{T}^m \in \mathbf{t}(\mathtt{A}^m\mathtt{C}^m)$. Consider an accepting path π of $\mathbf{t}$ whose label is $(\mathtt{A}^m\mathtt{C}^m, \mathtt{G}^m\mathtt{T}^m)$ and say π consists of N consecutive edges, for some positive integer N. Then, these edges are $s_{i-1} \xrightarrow{(x_i, y_i)}{}^{*} s_i$, for $i = 1, \ldots, N$, so that the concatenation of the x_i's is equal to $\mathtt{A}^m\mathtt{C}^m$ and the concatenation of the y_i's is equal to $\mathtt{G}^m\mathtt{T}^m$. As $\mathbf{t}$ is in normal form, we have $N = 4m$, and as $m > n$, there is a smallest integer $k \geq 1$ such that state s_k is equal to a previous one, that is $s_k = s_j$ such that $j < k$. By the choice of k, we have $k \leq n < m$. Let $x = x_1 \cdots x_j$, $u = x_{j+1} \cdots x_k$, $x' = x_{k+1} \cdots x_N$, and $y = y_1 \cdots y_j$, $v = y_{j+1} \cdots y_k$, $y' = y_{k+1} \cdots y_N$. As $j - k > 0$ and $\mathbf{t}$ is in normal form we have that

$$|u| > 0 \quad \text{or} \quad |v| > 0. \tag{10}$$

Using a standard pumping argument for finite state machines, we have that the path that results if we delete from π the $k - j$ edges between s_j and s_k is also an accepting path whose label is (xx', yy'). As each x_i and y_i is of length 0 or 1, we have $|xu| \leq k < m$ and $|yv| < m$, and also $|u| \leq k - j$ and $|v| \leq k - j$. This implies $xx' = \mathtt{A}^{m-|u|}\mathtt{C}^m$ and $yy' = \mathtt{G}^{m-|v|}\mathtt{T}^m$. As $xx' \neq yy'$ and $yy' \in \mathbf{t}(xx')$ we have that $\{xx', yy'\}$ is not δ-nonoverlapping, which implies $xx' = \delta(yy')$, that is, $\mathtt{A}^{m-|u|}\mathtt{C}^m = \mathtt{A}^m\mathtt{C}^{m-|v|}$ and, therefore, $|u| = |v| = 0$ which contradicts (10). $\square$

4. Expressiveness of Transducer-based Properties

In this section we examine the descriptive power of the newly defined transducer DNA code properties, that is, the θ-transducer properties. In Theorem 16 we show that these properties properly include the regular θ-trajectory properties. On the other hand, in Proposition 14 we show that there is a DNA code property that is not a θ-transducer property.

Remark 13. We note that every θ-transducer property is a 3-independence.

Proposition 14. *The θ-free property (defined below) [8] is not a θ-transducer property.*

(D) A language $L \subseteq A^*$ is θ-free *if and only if $L^2 \cap A^+\theta(L)A^+ = \emptyset$.*

Proof. By the above remark, it is sufficient to show that, for $\theta = \delta$ and $A = \Delta$, the θ-free property is not 3-independent. Assume the contrary and consider the language

$$K = \{\texttt{ACGT, CCAC, GTAA}\}.$$

This is not δ-free, as $\texttt{ACGT} = \delta(\texttt{ACGT})$ and $\texttt{CCACGTAA} \in \Delta^+\texttt{ACGT}\Delta^+$. On the other hand, one verifies that every nonempty subset of K of cardinality less than 3 is δ-free, so by our assumption also K must be δ-free, which is a contradiction. □

The remainder of this section is devoted to Theorem 16. Recall the DNA alphabet is $\Delta = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. The following DNA language property is considered in Theorem 16

$$\mathcal{H} = \{L \subseteq \Delta^* \mid H(u, \delta(v)) \geq 2, \text{ for all } u, v \in L\},$$

where $H(\cdot, \cdot)$ is the Hamming distance function with the assumption that its value is ∞ when applied on different length words. Note that $\mathcal{H}$ is described by δ and the transducer shown in Fig. 3. The transducer is designed to realize all word pairs (x, y) such that $H(x, y) \leq 1$. State 0 is final and realizes all pairs of the form (x, x). State 1 is final and realizes all pairs (x, y) with $H(x, y) = 1$.

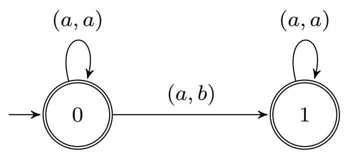


Figure 3: The transducer describing, together with δ, the $\mathcal{S}$-property $\mathcal{H}$.

Example 15. The following DNA languages do not satisfy $\mathcal{H}$:

$L_0 = \{\texttt{AGG, CCA}\}$, $\quad L_0' = \{\texttt{GAG, CCC}\}$.

For instance, $H(\texttt{CCA}, \delta(\texttt{AGG})) = 1$. The following languages satisfy $\mathcal{H}$:

$L_1 = \{\texttt{ACG, GAT}\}$, $\quad L_2 = \{\texttt{CAC, GCT}\}$,

$L_3 = \{\texttt{AAA, CCT}\}$, $\quad L_4 = \{\texttt{AAA, CTC}\}$, $\quad L_5 = \{\texttt{AAA, TCC}\}$.

For instance, as $\delta(\texttt{AAA}) = \texttt{TTT}$ and all words $u \in L_3$ contain at most one $\texttt{T}$, it follows that $H(u, \delta(\texttt{AAA})) \geq 2$. Now using $\delta(\texttt{CCT}) = \texttt{AGG}$, one verifies that $H(u, \delta(\texttt{CCT})) \geq 2$ for any $u \in L_3$. Thus, indeed L_3 satisfies $\mathcal{H}$.

We note that the actual definitions of bond-free properties in [22] are given in terms of a pair (T_1, T_2) of arbitrary sets of trajectories. However, here we only consider sets of trajectories that can be represented by regular expressions. Despite this restriction, the second statement of the following theorem remains true if one uses (T_1, T_2) instead of $(\bar{e}_1, \bar{e}_2)$, as the proof makes no use of the fact that the trajectory sets involved are regular.

Theorem 16. *1. Let θ be an antimorphic involution. Every regular θ-trajectory property is a θ-transducer property (in particular an $\mathcal{S}$-property). Moreover, if the property is described by a trajectory pair $(\bar{e}_1, \bar{e}_2)$, then it is also described as a strict property by a transducer of size $O(|\bar{e}_1| \cdot |\bar{e}_2|)$.*

2. Property $\mathcal{H}$ is a δ-transducer property, but not a (regular) δ-trajectory one.

Proof. For the <u>first</u> statement, we claim that, given any trajectory regular expression $\bar{a}$, each of the following operators is a transducer operator such that $|\mathbf{t}_i^{\bar{a}}| = O(|\bar{a}|)$:

$$\begin{aligned}
\mathbf{t}_1^{\bar{a}}(X) &= X \rightsquigarrow_{\bar{a}} A^* \\
\mathbf{t}_2^{\bar{a}}(X) &= X \sqcup\!\sqcup_{\bar{a}} A^* \\
\mathbf{t}_3^{\bar{a}}(X) &= X \rightsquigarrow_{\bar{a}} A^+ \\
\mathbf{t}_4^{\bar{a}}(X) &= X \sqcup\!\sqcup_{\bar{a}} A^+
\end{aligned}$$

The statement would then follow by combining appropriately the above transducers. More specifically, for any trajectory pair $\bar{e} = (\bar{e}_1, \bar{e}_2)$, we have that

$$\begin{aligned}
\Phi_{\bar{e}}^{\mathbf{s}}(L) &= \big(\mathbf{t}_2^{\bar{e}_2} \circ (\mathbf{t}_1^{\bar{e}_1} \uparrow \mathbf{a}_+)\big)(L) \\
\Phi_{\bar{e}}(L) &= \Big(\big(\mathbf{t}_2^{\bar{e}_2} \circ (\mathbf{t}_3^{\bar{e}_1} \uparrow \mathbf{a}_+)\big) \vee \big(\mathbf{t}_4^{\bar{e}_2} \circ (\mathbf{t}_1^{\bar{e}_1} \uparrow \mathbf{a}_+)\big)\Big)(L)
\end{aligned}$$

where $\mathbf{a}_+$ is any automaton accepting A^+, hence,

$$\mathcal{B}_\theta^{\mathbf{s}}(\bar{e}_1, \bar{e}_2) = \mathcal{S}_{\theta, \mathbf{t}_2^{\bar{e}_2} \circ (\mathbf{t}_1^{\bar{e}_1} \uparrow \mathbf{a}_+)} \quad \text{and} \quad \mathcal{B}_\theta(\bar{e}_1, \bar{e}_2) = \mathcal{S}_{\theta, \big(\mathbf{t}_2^{\bar{e}_2} \circ (\mathbf{t}_3^{\bar{e}_1} \uparrow \mathbf{a}_+)\big) \vee \big(\mathbf{t}_4^{\bar{e}_2} \circ (\mathbf{t}_1^{\bar{e}_1} \uparrow \mathbf{a}_+)\big)}$$

The claim about $\mathbf{t}_4^{\bar{a}}$ is already shown in [20], where we have that $|\mathbf{t}_4^{\bar{a}}| = O(|\bar{a}|)$. For the claim about $\mathbf{t}_2^{\bar{a}}$, first note that $X \sqcup\!\sqcup_{\bar{a}} A^* = (X \sqcup\!\sqcup_{\bar{a}} A^+) \cup (X \sqcup\!\sqcup_{\bar{a}} \{\varepsilon\})$, so $\mathbf{t}_2^{\bar{a}}$ is equal to $(\mathbf{t}_4^{\bar{a}} \vee \mathbf{t}_{\bar{a},\mathrm{id}})$, where $\mathbf{t}_{\bar{a},\mathrm{id}}$ is a transducer with $\mathbf{t}_{\bar{a},\mathrm{id}}(x) = x \sqcup\!\sqcup_{\bar{a}} \{\varepsilon\}$ and defined as follows. First note that by definition, $y \in x \sqcup\!\sqcup_{\bar{a}} \{\varepsilon\}$ if and only if $y = x$ and $0^{|x|} \in L(\bar{a})$. Let $\mathbf{a} = (Q, A, E, I, F)$ be an automaton accepting $L(\bar{a})$. This can be constructed such that $|\mathbf{a}| = O(|\bar{a}|)$. Then, $\mathbf{t}_{\bar{a},\mathrm{id}} = (Q, A, E', I, F)$ such that E' consists of all ε-edges in E and all edges $(p, (b, b), q)$, for $b \in A$, where $(p, 0, q)$ is any edge in E. Then, $\mathbf{t}_{\bar{a},\mathrm{id}}$ realizes a pair (x, y) if and only if $x = y$ and $\mathbf{a}$ accepts $0^{|x|}$. Moreover, we have that $|\mathbf{t}_{\bar{a},\mathrm{id}}| = O(|\bar{a}|)$.

In [32] it is observed that $y \in (x \rightsquigarrow_t w)$ if and only if $x \in (y \sqcup\!\sqcup_t w)$, for all words x, y, w and trajectories t, which implies that $\mathbf{t}_3^{\bar{a}}$ and $\mathbf{t}_1^{\bar{a}}$ are simply the inverses of the transducers $\mathbf{t}_4^{\bar{a}}$ and $\mathbf{t}_2^{\bar{a}}$, respectively. Hence, also $|\mathbf{t}_1^{\bar{a}}| = O(|\bar{a}|)$ and $|\mathbf{t}_3^{\bar{a}}| = O(|\bar{a}|)$.

For the <u>second</u> statement we recall that $\mathcal{H}$ is described by δ and the transducer shown in Fig. 3. We use the following notation: $\Phi_{\bar{e}}^{?}$ for either of the operators $\Phi_{\bar{e}}$ and $\Phi^{\mathbf{s}}_{\bar{e}}$, and $\mathcal{B}^{?}_{\theta}(\bar{e})$ for either of the properties $\mathcal{B}_{\theta}(\bar{e})$ and $\mathcal{B}^{\mathbf{s}}_{\theta}(\bar{e})$. For the second part of the statement, we argue by contradiction, so we assume that there is a pair of trajectory regular expressions $\bar{e} = (\bar{e}_1, \bar{e}_2)$ such that

$$\mathcal{H} = \mathcal{B}^{?}_{\theta}(\bar{e}_1, \bar{e}_2).$$

Using the definition of $\Phi^{?}_{\bar{e}}$, one verifies that

$$\Phi^{?}_{\bar{e}}(a) \subseteq aA^*, \text{ for all } a \in A.$$

Consider the DNA language $K = \{\mathtt{A}, \mathtt{C}\}$. We have that K does not satisfy $\mathcal{H}$, but on the other hand $\delta(K) \cap \Phi^{?}_{\bar{e}}(K) = \emptyset$, which means that K satisfies $\mathcal{B}^{?}_{\theta}(\bar{e}_1, \bar{e}_2)$, which leads to the required contradiction. □

The counter example used to prove the second statement of Theorem 16 might seem a little artificial, as the language $K = \{\mathtt{A}, \mathtt{C}\}$ consists of 1-letter words, which is of no practical value. The next result gives a stronger statement, as it requires that all words involved are of length at least 2.

Proposition 17. *The following property*

$$\mathcal{H}_2 = \{L \subseteq \Delta^* \mid |u| \geq 2 \text{ and } H(u, \delta(v)) \geq 2, \text{ for all } u, v \in L\}$$

is a δ-transducer property but not a δ-trajectory property.

The proof of this result requires a couple of intermediate results, which we present next.

Lemma 18. *Let x, y be any words and s, t be any trajectories. If $y \in ((x \rightsquigarrow_s A^*) \cap A^+) \sqcup_t A^*$ then*

$$|t| - |s| = |t|_1 - |s|_1 = |y| - |x| \quad \text{and} \quad |s|_1 < |x|.$$

Proof. The premise of the statement implies that $y \in z \sqcup_t w_2$ and $z \in ((x \rightsquigarrow_s w_1) \cap A^+)$ for some words z, w_1, w_2 with $|z| > 0$. Informally, this means that y results by deleting $|w_1|$ symbols from x, with $|w_1| < |x|$, and then inserting $|w_2|$ symbols. More formally as $|t| = |y|$ and $|s| = |x|$, we have that $|t| - |s| = |y| - |x|$. Also as $|z| = |x| - |w_1| = |s| - |s|_1$, we have that $|s| > |s|_1$ and, therefore, $|x| > |s|_1$, as required. Now, we have

$$|s|_1 = |w_1| = |x| - |z| = |x| - (|y| - |w_2|) = |x| - |y| + |t|_1$$

and, therefore, $|t|_1 - |s|_1 = |y| - |x|$. □

As we shall see further below the premise of the following lemma leads to a contradiction, but the lemma is helpful so that one can see the consequences of that premise.

Lemma 19. *Let $\bar{e} = (\bar{e}_1, \bar{e}_2)$ be a pair of trajectory regular expressions and assume that $\mathcal{H} = \mathcal{B}^{?}_{\theta}(\bar{e})$.*

1. *There is no pair (s,t) of trajectories in $L(\bar{e}_1)\times L(\bar{e}_2)$ such that $|s|=|t|=3$ and $|s|_1=|t|_1=2$.*
2. *If x,y are DNA words of length 3 and $(s,t)\in L(\bar{e}_1)\times L(\bar{e}_2)$ such that $x\neq\delta(y)$ and $y\in((x\rightsquigarrow_s\Delta^*)\cap\Delta^+)\sqcup\!\sqcup_t\Delta^*$ then $|s|=|t|=3$ and $|s|_1=|t|_1=1$.*
3. *We have that $010\in L(\bar{e}_1)$ or $010\in L(\bar{e}_2)$.*
4. *We have that $(001,001)\in L(\bar{e}_1)\times L(\bar{e}_2)$ or $(100,100)\in L(\bar{e}_1)\times L(\bar{e}_2)$.*

Proof. We shall use some of the seven languages in Example 15.

For the first statement, assume for the sake of contradiction that the two trajectories have equal length and exactly two 1s each. By applying $(\mathtt{AAA}\rightsquigarrow_s\Delta^*)\cap\Delta^+$ followed by $\sqcup\!\sqcup_t\Delta^*$, the result is $\Phi^?_{\bar{e}}(\mathtt{AAA})$ and is equal to $\mathtt{A}\Delta\Delta$ or $\Delta\mathtt{A}\Delta$ or $\Delta\Delta\mathtt{A}$, depending on whether $t=011$ or $t=101$ or $t=110$, respectively. More specifically, if $t=011$ then $\Phi^?_{\bar{e}}(\mathtt{AAA})$ contains $\delta(\mathtt{CCT})$, which contradicts the fact that L_3 satisfies $\mathcal{H}$. If $t=101$ then $\Phi^?_{\bar{e}}(\mathtt{AAA})$ contains $\delta(\mathtt{CTC})$, which contradicts the fact that L_4 satisfies $\mathcal{H}$. If $t=110$ then $\Phi^?_{\bar{e}}(\mathtt{AAA})$ contains $\delta(\mathtt{TCC})$, which contradicts the fact that L_5 satisfies $\mathcal{H}$.

For the second statement, Lemma 18 implies that $|s|=|t|=3$ and $|s|_1=|t|_1\leq 1$, and $x\neq\delta(y)$ implies that $|s|_1\neq 0$. Hence, $|s|_1=|t|_1=1$, as required.

For the third statement, the fact that L_0' does not satisfy $\mathcal{H}$ implies that there are words $u,v\in L_0'$ such that $\delta(v)\in\Phi^?_{\bar{e}}(u)$ and, therefore, there are words w_1,w_2 and $(s,t)\in L(\bar{e}_1)\times L(\bar{e}_2)$ such that

$$\delta(v)\in((u\rightsquigarrow_s w_1)\cap\Delta^+)\sqcup\!\sqcup_t w_2.$$

By the previous statement, $|s|=|t|=3$ and $|s|_1=|t|_1=1$, which implies $|w_1|=|w_2|=1$. For the sake of contradiction assume $s\neq 010$ and $t\neq 010$. Let $u=u_1u_2u_3$ with each u_i being a symbol. There are four cases about the values of s and t, all of which lead to contradictions. In particular, if $s=001$ and $t=001$ then $\delta(v)=u_1u_2w_2$, which implies $v=\bar{w}_2\bar{u}_2\bar{u}_1$. If $s=001$ and $t=100$ then $\delta(v)=w_2u_1u_2$, which implies $v=\bar{u}_2\bar{u}_1\bar{w}_2$. If $s=100$ and $t=001$ then $\delta(v)=u_2u_3w_2$, which implies $v=\bar{w}_2\bar{u}_3\bar{u}_2$. If $s=100$ and $t=100$ then $\delta(v)=w_2u_2u_3$, which implies $v=\bar{u}_3\bar{u}_2\bar{w}_2$. By inspection, we get that in all four cases the words $u_1u_2u_3$, v cannot be both in L_0'.

For the fourth statement, the fact that L_0 does not satisfy $\mathcal{H}$ implies that there are words $u,v\in L_0$ such that $\delta(v)\in\Phi^?_{\bar{e}}(u)$ and, therefore, there are words w_1,w_2 and $(s,t)\in L(\bar{e}_1)\times L(\bar{e}_2)$ such that

$$\delta(v)\in((u\rightsquigarrow_s w_1)\cap\Delta^+)\sqcup\!\sqcup_t w_2.$$

By a previous statement, $|s|=|t|=3$ and $|s|_1=|t|_1=1$, which implies $|w_1|=|w_2|=1$. Let $u=u_1u_2u_3$ with each u_i being a symbol. The rest of the proof consists of four parts:

$s=010$ leads to a contradiction;

$t = 010$ leads to a contradiction;
$s = 001$ implies $t = 001$;
$s = 100$ implies $t = 100$.

For the first part, if $s = 010$ then depending on whether $t = 001$ or $t = 010$ or $t = 100$, we have that $\delta(v) = u_1u_3w_2$ or $\delta(v) = u_1w_2u_3$ or $\delta(v) = w_2u_1u_3$, and hence, $v = \bar{w}_2\bar{u}_3\bar{u}_1$ or $v = \bar{u}_3\bar{w}_2\bar{u}_1$ or $v = \bar{u}_3\bar{u}_1\bar{w}_2$. By inspection we have that, in any case, it is impossible to have $u, v \in L_0$. For the second part, if $t = 010$ then depending on whether $s = 001$ or $s = 100$, we have that $\delta(v) = u_1w_2u_2$ or $\delta(v) = u_2w_2u_3$, and hence, $v = \bar{u}_2\bar{w}_2\bar{u}_1$ or $v = \bar{u}_3\bar{w}_2\bar{u}_2$. By inspection we have that, in any case, it is impossible to have $u, v \in L_0$. For the third part, if $s = 001$ then, as t cannot be 010, we have that $t = 100$ or $t = 001$. The case $t = 100$ implies $\delta(v) = w_2u_1u_2$ and then $v = \bar{u}_2\bar{u}_1\bar{w}_2$, which contradicts the fact that $u, v \in L_0$. Hence, $t = 001$. Finally for the last part, if $s = 100$ then, as t cannot be 010, we have that $t = 100$ or $t = 001$. The case $t = 001$ implies $\delta(v) = u_2u_3w_2$ and then $v = \bar{w}_2\bar{u}_3\bar{u}_2$, which contradicts the fact that $u, v \in L_0$. Hence, $t = 100$. □

Proof. (Of Proposition 17.) The fact that $\mathcal{H}_2$ is a δ-transducer $\mathcal{S}$-property is established using the transducer in Fig. 4. For the second part of the statement,

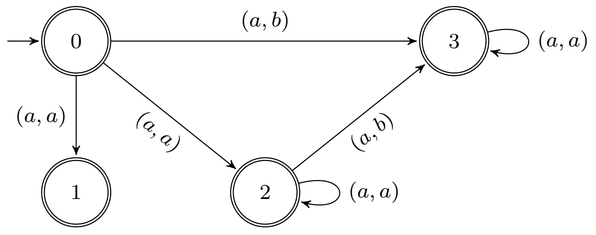


Figure 4: The transducer describing, together with δ, the $\mathcal{S}$-property $\mathcal{H}_2$.

we argue by contradiction, so we assume that there is a pair of trajectory regular expressions $(\bar{e}_1, \bar{e}_2)$ such that

$$\mathcal{H}_2 = \mathcal{B}^{?}_{\delta}(\bar{e}_1, \bar{e}_2).$$

By Lemma 19, we have that $001 \in L(\bar{e}_2)$ or $100 \in L(\bar{e}_2)$, and that $001 \in L(\bar{e}_1)$ or $100 \in L(\bar{e}_1)$. Moreover, we can distinguish the following four cases, which all lead to contradictions. We also consider the languages L_1 and L_2 defined in Example 15.

Case '$010 \in L(\bar{e}_1)$ *and* $001 \in L(\bar{e}_2)$*'.* Then, `GCT` results into `GT`, then into `GTG` and then into `CAC` using, respectively, the operations $\rightsquigarrow_{010}$, $\sqcup\!\sqcup_{001}$ and δ, which contradicts the fact that L_2 satisfies $\mathcal{H}$.

Case '$010 \in L(\bar{e}_1)$ *and* $100 \in L(\bar{e}_2)$*'.* Then, `GAT` results into `GT`, then into `CGT` and then into `ACG` using, respectively, the operations $\rightsquigarrow_{010}$, $\sqcup\!\sqcup_{100}$ and δ, which contradicts the fact that L_1 satisfies $\mathcal{H}$.

Case '$001 \in L(\bar{e}_1)$ and $010 \in L(\bar{e}_2)$'. Then, `ACG` results into `AC`, then into `ATC` and then into `GAT` using, respectively, the operations $\rightsquigarrow_{001}$, $\sqcup\!\sqcup_{010}$ and δ, which contradicts the fact that L_1 satisfies $\mathcal{H}$.

Case '$100 \in L(\bar{e}_1)$ and $010 \in L(\bar{e}_2)$'. Then, `CAC` results into `AC`, then into `AGC` and then into `GCT` using, respectively, the operations $\rightsquigarrow_{100}$, $\sqcup\!\sqcup_{010}$ and δ, which contradicts the fact that L_2 satisfies $\mathcal{H}$. □

5. The Satisfaction Problem

For $\theta = \text{id}$ and for input-altering and -preserving transducers the satisfaction problem is decidable in polynomial time [20]. In particular, for a regular language L given via an automaton $\mathbf{a}$, Condition (2) can be decided in time $\mathcal{O}(|\mathbf{t}||\mathbf{a}|^2)$, and Condition (3) can be decided in time $\mathcal{O}(|\mathbf{t}|^2|\mathbf{a}|^4)$. In the next theorem, we generalize the above to the satisfaction of θ-transcducer properties, and we improve the decision complexity of Condition (3).

We assume below that θ is a fixed, but arbitrary, involution, and that A is also a fixed, but arbitrary, alphabet.

Remark 20. The proofs of Theorem 21 and 26 make use of two results of [34]. The first result is that there is a $O(|\mathbf{t}|^2)$-time algorithm to decide whether a given transducer $\mathbf{t}$ is functional. The second result is that there is a $O(|\mathbf{t}|)$-time algorithm to decide whether a given transducer $\mathbf{t}$ *realizes a partial identity*: that is, whether $y \in \mathbf{t}(x)$ implies $y = x$, for all words x, y. We note that the general method of using transducer functionality to decide code-related properties of regular languages was already used in [35]. Those properties include whether the language in question is a UD code (Uniquely Decipherable code) or an immutable code. The method has also been applied in the case of error-detecting properties [31], and now in our present context of DNA-related properties.

Theorem 21. *Let $\mathbf{a}$ be an automaton and let $\mathbf{t}$ be a transducer.*

1. *It is decidable in time $O(|\mathbf{t}||\mathbf{a}|^2)$ whether $L(\mathbf{a})$ satisfies the property $\mathcal{S}_{\theta,\mathbf{t}}$.*
2. *Assume $\mathbf{t}$ is a θ-input-preserving transducer. It is decidable in time $O(|\mathbf{t}|^2|\mathbf{a}|^4)$ whether $L(\mathbf{a})$ satisfies the property $\mathcal{W}_{\theta,\mathbf{t}}$.*
3. *Assume θ is morphic. It is decidable in time $O(|\mathbf{t}|^2|\mathbf{a}|^4)$ whether $L(\mathbf{a})$ satisfies the property $\mathcal{W}_{\theta,\mathbf{t}}$.*
4. *Assume $\theta = \text{id}$ (so θ is morphic). It is decidable in time $O(|\mathbf{t}||\mathbf{a}|^2)$ whether $L(\mathbf{a})$ satisfies the property $\mathcal{W}_{\theta,\mathbf{t}}$.*

Proof. Let $L = L(\mathbf{a})$. For the first statement, the algorithm uses a product construction [30] to compute an automaton $\mathbf{b}$ of size $O(|\mathbf{t}||\mathbf{a}|)$ accepting $\mathbf{t}(L)$, and then modifies the automaton $\mathbf{a}$ to an automaton $\mathbf{a}^\theta$ of size $O(|\mathbf{a}|)$ accepting $\theta(L)$. If θ is morphic, this can be done be replacing each edge (p, a, q) of $\mathbf{a}$ by the edge $(p, \theta(a), q)$. If θ is antimorphic, the above change in the transitions is applied on the automaton $\mathbf{a}^R$ which is the reverse of $\mathbf{a}$, that is, the same as $\mathbf{a}$

except that the roles of start and final states in $\mathbf{a}$ are exchanged in $\mathbf{a}^R$. Then, the algorithm computes the product automaton of size $O(|\mathbf{t}||\mathbf{a}|^2)$ accepting $\theta(L) \cap \mathbf{t}(L)$ and tests in linear time whether the graph of the product automaton has any path from an initial to a final state.

For the <u>second</u> statement, assume that $\mathbf{t}$ is θ-input-preserving, and let $\mathbf{s} = \mathbf{t} \downarrow \mathbf{a}^\theta \uparrow \mathbf{a}$ be the transducer of size $O(|\mathbf{t}||\mathbf{a}|^2)$ realizing the relation

$$\{(x, y) \mid y \in \mathbf{t}(x),\ x \in \theta(L(\mathbf{a})),\ y \in L(\mathbf{a})\}$$

and obtained by two product constructions: first on the input of $\mathbf{t}$ with $\mathbf{a}^\theta$; then, on the output of the resulting transducer with $\mathbf{a}$. We show next that testing whether or not L satisfies the input-preserving transducer property $\mathcal{W}_{\theta,\mathbf{t}}$ is equivalent to testing whether the transducer $\mathbf{s}$ is functional—this can be done in quadratic time (see Remark 20), so the time complexity part of the second statement follows.

Now assume L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$ and consider words v, w, x such that $v, w \in \mathbf{s}(x)$. We show that $v = w$. By definition of $\mathbf{s}$, we have $x = \theta(x_1)$, for some $x_1 \in L$, and $v, w \in L$ and $x \in \mathbf{t}(v) \cap \mathbf{t}(w)$. Hence,

$$\theta(x_1) \in \mathbf{t}(v) \cap \mathbf{t}(w).$$

Then, as $\theta(x_1) \notin \mathbf{t}(L - x_1)$, we have that $x_1 = v$ and $x_1 = w$; hence, $v = w$, as required. Conversely, assume that $\mathbf{s}$ is functional, but suppose for the sake of contradiction that L does not satisfy $\mathcal{W}_{\theta,\mathbf{t}}$, that is, there is $w \in L$ and $x \in L \setminus w$ (so $x \neq w$) such that $\theta(w) \in \mathbf{t}(x)$. This implies that $x \in \mathbf{t}^{-1}(\theta(L)) \cap L$ and, therefore, $x \in \mathbf{s}(\theta(w))$. Moreover, as $\mathbf{t}$ is θ-input-preserving, we have that $\theta(w) \in \mathbf{t}(w)$ and that $w \in \mathbf{s}(\theta(w))$. As $\mathbf{s}$ is functional, we get $x = w$, which is a contradiction.

For the <u>third</u> statement, assume θ is morphic. Let $\mathbf{t}'$ be the transducer resulting from $\mathbf{t}$, if we add a new state f that is both, initial and final, and the transitions $(f, (a, \theta(a)), f)$ for all letters $a \in A$. Then, we have that $\mathbf{t}'(x) = \mathbf{t}(x) \cup \theta(x)$ and $\mathbf{t}'$ is θ-input-preserving. Moreover, it follows that $\mathcal{W}_{\theta,\mathbf{t}} = \mathcal{W}_{\theta,\mathbf{t}'}$. Hence, the statement now follows from the previous one.

For the <u>fourth</u> statement, assume $\theta = \mathrm{id}$, so θ is morphic. Then, the condition $L \in \mathcal{W}_{\mathrm{id},\mathbf{t}}$ is equivalent to whether or not $\mathbf{s}$ realizes a partial identity function, which can be decided in linear time with respect to the size of $\mathbf{s}$—see Remark 20. Note that the partial identity test does not require that $\mathbf{t}$ is input-preserving if $\theta = \mathrm{id}$. □

Remark 22. We note that deciding the satisfaction question for any θ-trajectory property involves testing the emptiness conditions in (6) or (7), which requires time $\mathcal{O}(|\mathbf{a}|^2|\bar{e}_1||\bar{e}_2|)$. Such a property can be expressed as θ-transducer $\mathcal{S}$-property (recall Theorem 16) using a transducer of size $\mathcal{O}(|\bar{e}_1||\bar{e}_2|)$ and, therefore, the satisfaction question can still be solved within the same asymptotic time complexity. We also note that [17] presents algorithms for testing whether a regular language (given via an automaton $\mathbf{a}$) is a UD code (Uniquely Decipherable code), a θ-compliant language, a θ-k-hairpin-free language. Our approach in this paper does not include the case of the UD code property. Assuming a fixed-size alphabet, the satisfaction test for θ-compliance in [17] is of time $O(|\mathbf{a}|^2)$—this agrees with our approach here if we assume as fixed the transducer $\mathbf{t}$ describing that property.

Table 1 summarizes under which conditions the satisfaction and maximality problems are decidable for regular languages. For the case of non-restricted transducer $\mathcal{W}$-properties, we show decidability using a different method; see Sect. 5.1. The undecidability result holds for every fixed permutation θ over an alphabet with at least two letters, in particular, all results apply to the DNA-involution δ. All maximality results are discussed in Sect. 6.

Problem	Property $\mathcal{S}_{\theta,\mathbf{t}}$		Property $\mathcal{W}_{\theta,\mathbf{t}}$	
	no restriction	$\mathbf{t}$ is θ-i.-alter.	no restriction	$\mathbf{t}$ is θ-i.-preserv.
SAT	decidable in time $\mathcal{O}(\lvert\mathbf{t}\rvert\lvert\mathbf{a}\rvert^2)$ Theorem 21		decidable Theor. 21,26	decidable in time $\mathcal{O}(\lvert\mathbf{t}\rvert^2\lvert\mathbf{a}\rvert^4)$ Theor. 21
MAX	undecidable Corollary 31	decidable, PSPACE-hard Theorem 27, Corollary 28		

Table 1: (Un-)decidability of the satisfaction (SAT) and the maximality (MAX) problems for a fixed antimorphic permutation θ, a given transducer $\mathbf{t}$, and a regular language L given via an automaton $\mathbf{a}$.

5.1. The Satisfaction Problem for non-restricted $\mathcal{W}$-properties

Here we establish the decidability of non-restricted transducer $\mathcal{W}$-properties for regular languages, when θ is antimorphic. We do not concern the complexity of this algorithm; optimizing the algorithm and analyzing its complexity is part of future research.

Notation. Let $\mathbf{t}$ be a transducer, let θ be an antimorphic permutation, and let L be a regular language over the alphabet A. Let $\mathbf{a}$ and $\mathbf{a}^\theta$ be NFAs accepting the languages L and $\theta(L)$, respectively—note that $\mathbf{a}^\theta$ is constructed in the proof of Theorem 21. Let

$$\mathbf{s} = (Q_\mathbf{s}, A, E_\mathbf{s}, I_\mathbf{s}, F_\mathbf{s}) = \mathbf{t} \downarrow \mathbf{a} \uparrow \mathbf{a}^\theta$$

be the transducer such that $y \in \mathbf{s}(x)$ if and only if $y \in \mathbf{t}(x)$, $x \in L$, and $y \in \theta(L)$. We consider $\mathbf{s}$ to be *trim*, i. e., every state in $Q_\mathbf{s}$ lies on a path that leads from an initial state to a final sate. Furthermore, $\mathbf{s}$ is considered to be in *normal form* such that every edge is either labeled (a, ε) or (ε, a) for some letter $a \in A$. Thus, for any path $p \xrightarrow{(x,y)}{}^{*} q$ of length ℓ (the path has ℓ edges) in $\mathbf{s}$ we have $|xy| = \ell$. Note that $|\mathbf{s}| \in \mathcal{O}(|\mathbf{t}|\,|\mathbf{a}|^2)$.

Lemma 23. *The regular language L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$ if and only if for all words $x, y \in A^+$*

$$y \in \mathbf{s}(x) \implies \theta(x) = y.$$

Proof. We will prove the contrapositive: $L \notin \mathcal{W}_{\theta,\mathbf{t}}$ if and only if there exists $x, y \in A^+$ such that $y \in \mathbf{s}(x)$ and $\theta(x) \neq y$. Recall that $L \notin \mathcal{W}_{\theta,\mathbf{t}}$ if and only if there exists $w \in L$ such that $\theta(w) \in \mathbf{t}(L \setminus w)$.

Assume that $L \notin \mathcal{W}_{\theta,\mathbf{t}}$ and, therefore, $w \in L$ exists such that $\theta(w) \in \mathbf{t}(L \setminus w)$. Let $x \in L \setminus w$ such that $\theta(w) \in \mathbf{t}(x)$ and $y = \theta(w) \in \theta(L)$. Clearly, we have $y \in \mathbf{s}(x)$ and $y \neq \theta(x)$.

Conversely, assume that $x, y \in A^+$ exists such that $y \in \mathbf{s}(x)$ and $y \neq \theta(x)$. Let $w = \theta^{-1}(y)$. As $y \in \theta(L)$, we have that $w \in L$ $x \in L \setminus w$, and $\theta(w) \in \mathbf{t}(x) \subseteq \mathbf{t}(L \setminus w)$. Therefore, $L \notin \mathcal{W}_{\theta,\mathbf{t}}$. □

Let $T_{\mathbf{s}} = \{(x_1, x_2, x_3) \in (A^*)^3 \mid |x_1x_2x_3| \leq |\mathbf{s}|\}$ be a set of word triples. Note that the length restrictions for the words ensures that $T_{\mathbf{s}}$ is a finite set. For each triple $t = (x_1, x_2, x_3) \in T_{\mathbf{s}}$ we define a relation

$$R_t = \{(x_1(x_2)^k x_3, \theta(x_1(x_2)^k x_3)) \mid k \in \mathbb{N}\} \subseteq A^* \times A^*.$$

Note that we allow that any word of x_1, x_2, x_3 is empty; in particular, if $x_2 = x_3 = \varepsilon$, then R_t contains only one pair of words $(x_1, \theta(x_1))$.

Lemma 24. *The regular language L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$ if and only if the relation $R(\mathbf{s})$ realized by $\mathbf{s}$ satisfies*

$$R(\mathbf{s}) \subseteq \bigcup_{t \in T_{\mathbf{s}}} R_t. \tag{11}$$

Proof. Recall that for every $(x, y) \in R_t$ with $t \in T_{\mathbf{s}}$ we have $\theta(x) = y$. If $\mathbf{s}$ satisfies Equation (11), then for all (x, y) which are realized by $\mathbf{s}$, we have $\theta(x) = y$; and by Lemma 23 L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$.

Conversely, suppose that L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$, and let (x, y) be a pair of words that is realized by $\mathbf{s}$, and note that $\theta(x) = y$ by Lemma 23. Note also that $|w| = |\theta(w)| = |\theta^{-1}(w)|$, for all words w.

If $|x| \leq |\mathbf{s}|$, then $(x, \theta(x)) = (x, y) \in R_t$ for $t = (x, \varepsilon, \varepsilon) \in T_{\mathbf{s}}$. Otherwise, every accepting path in $\mathbf{s}$ that is labeled by $(x, \theta(y))$ contains more than $|\mathbf{s}|$ edges, and therefore, must have a repeating state p

$$s \xrightarrow{(x_1,y_1)}{}^{*}\ p \xrightarrow{(x_2,y_2)}{}^{*}\ p \xrightarrow{(x_3,y_3)}{}^{*}\ f$$

such that $x = x_1x_2x_3$, $\theta(x) = y_1y_2y_3$, $s \in I_{\mathbf{s}}$, $f \in F_{\mathbf{s}}$, $x_2y_2 \neq \varepsilon$, $|x_1x_2y_1y_2| \leq |\mathbf{s}|$ (using the pigeonhole principle). By Lemma 23 for all $i \in \mathbb{N}$

$$x_1x_2^i x_3 = \theta^{-1}(y_1y_2^i y_3) = \theta^{-1}(y_3)\big(\theta^{-1}(y_2)\big)^i \theta^{-1}(y_1).$$

Firstly note that, as $|x_1x_3| = |y_3y_1|$ and $|x_1x_2x_3| = |y_3y_2y_1|$, we have that $|x_2| = |y_2|$. Now, consider $i = 2|x|$. Because $|x_1x_2x_3| \geq |\mathbf{s}| \geq |x_1x_2y_1y_2|$, we have that $\theta^{-1}(y_2)\theta^{-1}(y_1)$ is a suffix of x_3. Since i is sufficiently large, the suffix x_2x_3 of $x_1x_2^i x_3$ cannot overlap with the prefix $\theta^{-1}(y_3)$ of $x_1x_2^i x_3$. Hence, there exists a suffix u of $\theta^{-1}(y_2)$ and an integer $j \geq 2$ such that

$$x_2x_3 = u\big(\theta^{-1}(y_2)\big)^j \theta^{-1}(y_1).$$

Choose v such that $\theta^{-1}(y_2) = vu$ and note that $x_2 = uv$ because $|x_2| = |y_2|$. Let $x_3' = u\theta^{-1}(y_1)$ and observe that $x_3 = u(vu)^{j-1}\theta^{-1}(y_1) = x_2^{j-1}x_3'$. Furthermore, $|x_1x_2x_3'| \leq |x_1x_2y_1y_2| \leq |\mathbf{s}|$. We conclude that $(x, \theta(x)) = (x_1x_2^jx_3', \theta(x_1x_2^jx_3')) \in R_t$ for $t = (x_1, x_2, x_3') \in T_{\mathbf{s}}$. □

In order to test whether or not Equation (11) is satisfied, we perform two separate tests. Firstly, we test whether or not $\mathbf{s}$ satisfies the weaker condition

$$\mathbf{s} \subseteq \bigcup_{(x_1,x_2,x_3)\in T_s} (x_1x_2^*x_3) \times \theta(x_1x_2^*x_3). \tag{12}$$

Secondly, we ensure that

$$\forall x, y\colon y \in \mathbf{s}(x) \implies |x| = |y|\,. \tag{13}$$

Lemma 25. *Equation (11) is satisfied if and only if Equations (12) and (13) are satisfied.*

Proof. If Equation (11) is satisfied, then Equation (12) is satisfied because $R_{(x_1,x_2,x_3)} \subseteq (x_1x_2^*x_3) \times \theta(x_1x_2^*x_3)$ for $(x_1, x_2, x_3) \in T_{\mathbf{s}}$. Also note that for all $(x, y) \in R_t$ with $t \in T_{\mathbf{s}}$ we have $|x| = |y|$; therefore, Equation (11) implies Equation (13).

Conversely, assume that Equations (12) and (13) are satisfied. For all (x, y) that are realized by $\mathbf{s}$ we have there exists $(x_1, x_2, x_3) \in T_{\mathbf{s}}$ and $i, j \in \mathbb{N}$ such that $x = x_1x_2^ix_3$ and $y = \theta(x_1x_2^jx_3)$. Since the equation $|x| = |y|$ must also be satisfied, it is clear that $i = j$ and, hence, $(x, y) \in R_{(x_1,x_2,x_3)}$. We conclude that Equations (12) and (13) imply Equation 11. □

Below we assume that θ is a fixed, but arbitrary, antimorphic involution, and that A is a fixed, but arbitrary, alphabet.

Theorem 26. *Let L be a regular language given as automaton $\mathbf{a}$ and let $\mathbf{t}$ be a given transducer both defined over the alphabet A. It is decidable whether L satisfies $\mathcal{W}_{\theta,\mathbf{t}}$ or not.*

Proof. According to Lemmas 24 and 25 we have to decide whether or not the two Equations (12) and (13) are satisfied for the transducer $\mathbf{s} = \mathbf{t} \downarrow \mathbf{a} \uparrow \mathbf{a}^\theta$. It is known that it is decidable whether or not a given transducer is included in a recognizable relation (that is a relation $\bigcup_{i=1}^n A_i \times B_i$ for regular languages A_i, B_i); see [29]. Therefore, the inclusion in Equation (12) is decidable.

Equation (13) can be decided as follows. Let $\mathbf{s}^0$ be the transducer obtained from $\mathbf{s}$ by changing the alphabet to $\{0\}$ and every edge $(p, (x, y), q)$ of $\mathbf{s}$ to $(p, (0^{|x|}, 0^{|y|}), q)$. Then we have that

$$R(\mathbf{s}^0) = \{(0^{|x|}, 0^{|y|}) \mid (x, y) \in R(\mathbf{s})\}.$$

Moreover, Eq. (13) is equivalent to

$$\forall i, j \in \mathbb{N}\colon 0^j \in \mathbf{s}^0(0^i) \implies i = j,$$

which is equivalent to whether $R(\mathbf{s}^0)$ is a partial identity. □

6. The Maximality Problem

For $\theta = \mathrm{id}$ and for input-altering and -preserving transducers the maximality problem is decidable, but PSPACE-hard [20]. Here we show how to decide maximality of a regular language L with respect to a θ-transducer property; see Theorem 27. This result only holds when we consider $\mathcal{W}$-properties or when we consider $\mathcal{S}$-properties for θ-input-altering transducers. As in the case of existing transducer properties, it turns out that the maximality problem is PSPACE-hard; see Corollary 28. When we consider general $\mathcal{S}$-properties, the maximality problem becomes undecidable; see Corollary 31.

Theorem 27. *For an antimorphic permutation θ, a transducer $\mathbf{t}$, and a regular language L, all defined over A_k^*, such that either*

i.) $L \in \mathcal{W}_{\theta,\mathbf{t}}$ *or*

ii.) $L \in \mathcal{S}_{\theta,\mathbf{t}}$ *and* $\mathbf{t}$ *is θ-input altering,*

we have that L is maximal with property $\mathcal{W}_{\theta,\mathbf{t}}$ (resp., $\mathcal{S}_{\theta,\mathbf{t}}$) if and only if

$$L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L)) = A_k^*. \tag{14}$$

Proof. i.) Suppose $L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L)) = A_k^*$. For every word $w \in L^c$ we have $\theta(w) \in \mathbf{t}(L)$ or $w \in \mathbf{t}^{-1}(\theta(L))$. In the former case, we immediately obtain that $L \cup w$ does not satisfy $\mathcal{W}_{\theta,\mathbf{t}}$. In the latter case, there exists $u \in L$ such that $\theta(u) \in \mathbf{t}(w)$, and therefore, $L \cup w$ does not satisfy $\mathcal{W}_{\theta,\mathbf{t}}$. We conclude that L is maximal with respect to $\mathcal{W}_{\theta,\mathbf{t}}$

Conversely, suppose there exists a word w such that $w \notin L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L))$. Clearly, $w \in L^c$. Furthermore, we must have $\theta(w) \notin \mathbf{t}(L)$ and $\theta(u) \notin \mathbf{t}(w)$ for all $u \in L$. Since $L \in \mathcal{W}_{\theta,\mathbf{t}}$, we also have that $\theta(u) \notin \mathbf{t}(L \setminus u)$ for all $u \in L$. Thus, we obtain that $\forall u \in (L \cup w)\colon \theta(u) \notin \mathbf{t}((L \cup w) \setminus u)$, and therefore, L is not maximal with respect to $\mathcal{W}_{\theta,\mathbf{t}}$.

ii.) Suppose $L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L)) = A_k^*$. For all $w \in L^c$ we have $\theta(w) \cap \mathbf{t}(L) \neq \emptyset$ or $\mathbf{t}(w) \cap \theta(L) \neq \emptyset$. Thus, $L \cup w$ does not satisfy $\mathcal{S}_{\theta,\mathbf{t}}$ and L is maximal with respect to $\mathcal{S}_{\theta,\mathbf{t}}$

Conversely, suppose there exists a word w such that $w \notin L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L))$. Hence, $\theta(w) \cap \mathbf{t}(L) = \emptyset$ and $\mathbf{t}(w) \cap \theta(L) = \emptyset$. Furthermore, we have $\theta(L) \cap \mathbf{t}(L) = \emptyset$ because $L \in \mathcal{W}_{\theta,\mathbf{t}}$ and $\theta(w) \cap \mathbf{t}(w) = \emptyset$ because $\mathbf{t}$ is θ-input-altering. We conclude that $L \cup w$ satisfies $\mathcal{W}_{\theta,\mathbf{t}}$, and therefore, L is not maximal with respect to $\mathcal{W}_{\theta,\mathbf{t}}$. □

We note that it is PSPACE-hard to decide whether or not Equation (14) holds when L is given as NFA because it is PSPACE-hard to decide universality of a regular language given as NFA ($L \subseteq A_k^*$ is universal if $L = A_k^*$) [36].

Corollary 28. *For an antimorphic permutation θ, a transducer $\mathbf{t}$, and a regular language L given as NFA, all defined over A_k^*, such that either*

i.) $L \in \mathcal{W}_{\theta,\mathbf{t}}$ *or*

ii.) $L \in \mathcal{S}_{\theta,\mathbf{t}}$ and $\mathbf{t}$ is θ-input altering,
we have that it is PSPACE*-hard to decide whether or not L is maximal with property $\mathcal{W}_{\theta,\mathbf{t}}$ (resp., $\mathcal{S}_{\theta,\mathbf{t}}$).*

Proof. According to Theorem 27 deciding maximality of L with property $\mathcal{W}_{\theta,\mathbf{t}}$ (resp., $\mathcal{S}_{\theta,\mathbf{t}}$) is equivalent to deciding universality of $L \cup \theta^{-1}(\mathbf{t}(L)) \cup \mathbf{t}^{-1}(\theta(L))$. Let $\mathbf{t}_\emptyset$ be a transducer without final state which does not realize any pair of words. Now, L is maximal with property $\mathcal{S}_{\theta,\mathbf{t}_\emptyset}$ (resp., $\mathcal{W}_{\theta,\mathbf{t}_\emptyset}$) if and only if L is universal—a problem which is known to be PSPACE-hard. □

In the rest of this section we show that it is undecidable whether or not a transducer is θ-input-preserving. This question relates directly to the maximality problem of the empty language $\emptyset$ with respect to the property $\mathcal{S}_{\theta,\mathbf{t}}$, as stated in Corollary 31. We will reduce the famous, undecidable Post correspondence problem to the problem of deciding whether or not a given transducer is θ-input-preserving.

Definition 29. The *Post correspondence problem* (PCP) is as follows: given an alphabet Σ, and words $\alpha_0, \alpha_1, \dots, \alpha_{\ell-1} \in \Sigma^+$ and $\beta_0, \beta_1, \dots, \beta_{\ell-1} \in \Sigma^+$, decide whether or not there exists a non-empty sequence of integers $i_1, i_2, \dots, i_n \in A_\ell = \{0, 1, \dots, \ell-1\}$ such that

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}.$$

It is well-known that PCP is undecidable, even if $\Sigma = A_2$ is the binary alphabet.

Theorem 30. *For every fixed antimorphic permutation θ over A_k^* with $k \geq 2$ it is undecidable whether or not a given transducer is θ-input-preserving.*

Proof. Let $\alpha_0, \alpha_1, \dots, \alpha_{\ell-1} \in \Sigma^+$ and $\beta_0, \beta_1, \dots, \beta_{\ell-1} \in \Sigma^+$ be the PCP instance $\mathcal{A}$. We will define a transducer $\mathbf{t}_\mathcal{A}$ which accepts all pairs $(w, \theta(w))$ unless w is a binary encoding of a word uv where $u \in \Sigma^+$ and $v \in A_\ell^+$ such that v describes an integer sequence $i_1, i_2, \dots, i_n$ that is a solution of $\mathcal{A}$ and u is the corresponding solution word. For the ease of notation, we assume that Σ and A_ℓ are two disjoints alphabet and we let $\Gamma = \Sigma \cup A_\ell$ be their union. For $m = \lceil \log_2 |\Gamma| \rceil$, we let $h\colon \Gamma \to A_2^m$ be a morphic block code; i. e., an encoding of Γ into binary words of length m such that $h(a) = h(b)$ implies $a = b$ for all $a, b \in \Gamma$. Our goal is to define $\mathbf{t}_\mathcal{A}$ such that $\theta(w) \notin \mathbf{t}_\mathcal{A}(w)$ if and only if $w = h(uv)$ for $u \in \Sigma^+$, $v \in A_\ell^+$, $n = |v|$, and

$$u = \alpha_{v_{[n]}}\alpha_{v_{[n-1]}}\cdots\alpha_{v_{[1]}} = \beta_{v_{[n]}}\beta v_{[n-1]}\cdots\beta_{v_{[1]}}.$$

The transducer $\mathbf{t}_\mathcal{A}$ will consist of 3 effectively constructable components $\mathbf{t}_R$, $\mathbf{t}_\alpha$, and $\mathbf{t}_\beta$. Each component can be seen as a fully functional transducer such that $\mathbf{t}_\mathcal{A}$ becomes the union of the three transducers; this implies that

$$y \in \mathbf{t}_\mathcal{A}(x) \iff y \in \mathbf{t}_R(x) \cup \mathbf{t}_\alpha(x) \cup \mathbf{t}_\beta(x).$$

Each transducer component "validates" a certain property of a word w, by accepting all word pairs $(w, \theta(w))$ which do not have that property:

1.) $\mathbf{t}_R$ accepts $(w, \theta(w))$ if and only if $w \notin h(\Sigma^+ A_\ell^+)$;

2.) for $w \in h(uv)$ with $u \in \Sigma^+$ and $v \in A_\ell^+$, $\mathbf{t}_\alpha$ accepts $(w, \theta(w))$ if and only if $u \neq \alpha_{v_{[n]}} \alpha_{v_{[n-1]}} \cdots \alpha_{v_{[1]}}$; and

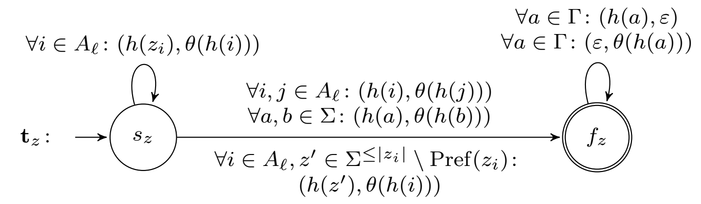
3.) for $w \in h(uv)$ with $u \in \Sigma^+$ and $v \in A_\ell^+$, $\mathbf{t}_\beta$ accepts $(w, \theta(w))$ if and only if $u \neq \beta_{v_{[n]}} \beta_{v_{[n-1]}} \cdots \beta_{v_{[1]}}$.

The first component ensures that every pair $(w, \theta(w))$ that is not accepted by $\mathbf{t}_\mathcal{A}$ must have the desired form $w \in h(uv)$ with $u \in \Sigma^+$ and $v \in A_\ell^+$. Components $\mathbf{t}_\alpha$ and $\mathbf{t}_\beta$ ensure that

$$\alpha_{v_{[n]}} \alpha_{v_{[n-1]}} \cdots \alpha_{v_{[1]}} = u = \beta_{v_{[n]}} \beta_{v_{[n-1]}} \cdots \beta_{v_{[1]}}$$

is the solution word that corresponds the integer sequence $v_{[n]}, v_{[n-1]}, \ldots, v_{[1]}$ if $(w, \theta(w))$ is not accepted by $\mathbf{t}_\mathcal{A}$. Therefore, every word pair $(w, \theta(w))$ which is not accepted by $\mathbf{t}_\mathcal{A}$ yields a solution for $\mathcal{A}$ and, vice versa, every solution for $\mathcal{A}$ yields a word pair $(w, \theta(w))$ that cannot be accepted by $\mathbf{t}_\mathcal{A}$. We conclude that $\mathbf{t}_\mathcal{A}$ is θ-input-preserving if and only if the PCP instance $\mathcal{A}$ has no solution. This implies that for fixed antimorphic θ over A_k^* with $k \geq 2$ it is undecidable whether or not a given transducer is θ-input-preserving because the PCP is undecidable.

Now, let us describe the transducer component $\mathbf{t}_R$ and recall that it has to work over the alphabet A_k. It is well known that for any two regular languages R_1 and R_2 there effectively exists a transducer which accepts the relation $R_1 \times R_2$. There is $\mathbf{t}_R$ such that $\mathbf{t}_R = (A_k^* \setminus h(\Sigma^+ A_\ell^+)) \times A_k^*$. It is easy to observe that we have $\mathbf{t}_R(w) = A_k^*$ if $w \notin h(\Sigma^+ A_\ell^+)$, and $\mathbf{t}_R(w) = \emptyset$ if $w \in h(\Sigma^+ A_\ell^+)$. Therefore, we have $\theta(w) \notin \mathbf{t}_R(w)$ if and only if $w \notin h(\Sigma^+ A_\ell^+)$. Note that this in particular implies that, if $\theta(w) \notin \mathbf{t}_R(w)$, then $w \in h(\Gamma^*) \subseteq (A_2^m)^*$. The other two transducer components $\mathbf{t}_\alpha$ and $\mathbf{t}_\beta$ will only work over word pairs from $h(\Gamma^*) \times \theta(h(\Gamma^*))$.



$\forall i \in A_\ell \colon (h(z_i), \theta(h(i)))$
$\forall a \in \Gamma \colon (h(a), \varepsilon)$
$\forall a \in \Gamma \colon (\varepsilon, \theta(h(a)))$
$\forall i, j \in A_\ell \colon (h(i), \theta(h(j)))$
$\forall a, b \in \Sigma \colon (h(a), \theta(h(b)))$
$\mathbf{t}_z \colon$
s_z
f_z
$\forall i \in A_\ell, z' \in \Sigma^{\leq |z_i|} \setminus \mathrm{Pref}(z_i) \colon$
$(h(z'), \theta(h(i)))$

Figure 5: For $z \in \{\alpha, \beta\}$ the two transducers $\mathbf{t}_\alpha$ and $\mathbf{t}_\beta$ enforce that w encodes a solution of the PCP instance $\mathcal{A}$ if $\theta(w) \notin (\mathbf{t}_\alpha + \mathbf{t}_\beta)(w)$ and $w \in h(\Sigma^+ A_\ell^+)$.

Finally, we define the two transducers $\mathbf{t}_\alpha$ and $\mathbf{t}_\beta$ which are based on the words α_i and β_i, respectively. For $z \in \{\alpha, \beta\}$ we define $\mathbf{t}_z$ as shown in Fig. 5. For a pair of words $(x, y) \in \mathbf{t}_z$, it is easy to see that $x \in h(\Gamma^*)$ and $y \in \theta(h(\Gamma^*))$. Furthermore, the edges from the final state f_z to itself ensure that if $(x, y) \in \theta$, then for all words $x' \in h(\Gamma^*)$ and $y' \in \theta(h(\Gamma^*))$, we have $(xx', yy') \in \mathbf{t}_z$ (we will not leave the final state anymore once it is reached, unless the word pair is not defined over $h(\Gamma^*) \times \theta(h(\Gamma^*))$). There are three possibilities to switch from state s_z to the final state f_z:

1.) we read a word from $h(A_\ell)$ in the first component and a words from $\theta(h(A_\ell))$ in the second component;

2.) we read a word from $h(\Sigma)$ in the first component and a words from $\theta(h(\Sigma))$ in the second component; or

3.) we read the word $\theta(h(i))$ with $i \in A_\ell$ in the second component and in the first component we read a word $h(z')$ such that z' is not a prefix of z_i and z_i is not a prefix z' because of the length restriction on z'.

For $x \in h(\Gamma^*)$ let u denote the longest word in Σ^* such that $h(u)$ is a prefix of x (thus, either $x = h(u)$ or $x = h(uix')$ for an integer $i \in A_\ell$ and $x' \in \Gamma^*$); and for $y \in \theta(h(\Gamma^*))$ let v denote the longest word in A_ℓ^* such that $\theta(h(v))$ is a prefix of y and let $n = |v|$ (thus, either $y = \theta(h(v))$ or $y = \theta(h(y'av)) = \theta(h(v))\theta(h(a))\theta(h(y'))$ for a symbol $a \in \Sigma$ and $y' \in \Gamma^*$). Because $\theta(h(v_{[n]}))\theta(h(v_{[n-1]})) \cdots \theta(h(v_{[1]}))$ is a prefix of y we obtain that the pair (x, y) is accepted by $\mathbf{t}_z$ if $u \neq z_{v_{[n]}} z_{v_{[n-1]}} \cdots z_{v_{[1]}}$. Conversely, if $u = z_{v_{[n]}} z_{v_{[n-1]}} \cdots z_{v_{[1]}}$, then $(h(u), \theta(h(v)))$ labels a path from s_z to s_z; since there is no edge from s_z which is labeled $(h(i), \varepsilon)$, $(\varepsilon, \theta(h(a)))$, or $(h(i), \theta(h(a)))$ for $i \in A_\ell$ and $a \in \Sigma$, we obtain that (x, y) cannot not be accepted by $\mathbf{t}_z$.

Suppose $\theta(w) \notin \mathbf{t}_z(w)$ and $w \in h(uv)$ for words $u \in \Sigma^+$ and $v \in A_\ell^+$. Following our notion from the previous paragraph, u is the longest word in Σ^* such that $h(u)$ is a prefix of w, and v is the longest word in A_ℓ^* such that $\theta(h(v))$ is a prefix of $\theta(w)$. Therefore, we obtain that $u = z_{v_{[n]}} \cdots z_{v_{[1]}}$. □

This leads to the undecidability of the maximality problem of a regular language L with respect to a θ-transducer-property $\mathcal{S}_{\theta,\mathbf{t}}$.

Corollary 31. *For every fixed antimorphic permutation θ over A_k^* with $k \geq 2$, it is undecidable whether or not the empty language $\emptyset$ is maximal with respect to the property $\mathcal{S}_{\theta,\mathbf{t}}$, for a given transducer $\mathbf{t}$.*

Proof. Clearly, the empty language satisfies $\mathcal{S}_{\theta,\mathbf{t}}$. For a word w, the language $\{w\}$ satisfies $\mathcal{S}_{\theta,\mathbf{t}}$ if and only if $\theta(w) \notin \mathbf{t}(w)$. Therefore, $\emptyset$ is maximal with property $\mathcal{S}_{\theta,\mathbf{t}}$ if and only if $\mathbf{t}$ is θ-input-preserving. Theorem 30 concludes the proof. □

7. Undecidability of the θ-PCP and the θ-input-altering Transducer Problem

Analogous to the undecidable PCP (see Definition 29), we introduce the θ version of the PCP and prove that it is undecidable as well; see Theorem 33. Further, we utilize the θ version of the PCP in order to show that it is undecidable whether or not a transducer is θ-input-altering; see Corollary 34.

Definition 32. For a fixed antimorphic permutation θ over A_k^*, we introduce the *θ-Post correspondence problem* (θ-PCP): given words $\alpha_0, \alpha_1, \ldots, \alpha_{\ell-1} \in A_k^+$ and $\beta_0, \beta_1, \ldots, \beta_{\ell-1} \in A_k^+$, decide whether or not there exists a non-empty sequence of integers $i_1, \ldots, i_n \in A_\ell = \{0, 1, \ldots, \ell - 1\}$ such that

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \theta(\beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}).$$

Theorem 33. *For every fixed antimorphic permutation θ over A_k^* with $k \geq 2$ the θ-PCP is undecidable.*

Proof. In order to prove that θ-PCP is undecidable, we will state an effective reduction of any PCP instance $\mathcal{A}$ over alphabet A_2 to a θ-PCP instance $\mathcal{T}$ over alphabet A_k such that $\mathcal{A}$ has a solution if and only if $\mathcal{T}$ has a solution. Let $\alpha_0, \alpha_1, \ldots, \alpha_{\ell-1} \in A_2^+$ and $\beta_0, \beta_1, \ldots, \beta_{\ell-1} \in A_2^+$ be an instance of the PCP which we call $\mathcal{A}$.

Note that θ and θ^{-1} are well-defined over $A_2 \subseteq A_k$. We define two morphisms g, h on A_2^* such that

$$g(0) = 00, \qquad g(1) = 01, \qquad h(0) = 10, \qquad h(1) = 11.$$

Note that for each pair of letters $z \in A_2^2$ we have either $z \in h(A_2)$ or $z \in g(A_2)$. Moreover, we let

$$\begin{aligned} \gamma_j &= g(\alpha_j), & \delta_j &= \theta^{-1}(h(\beta_j^R)), & &\text{for } j = 0, \ldots, \ell - 1,\\ \gamma_\ell &= h(0), & \delta_\ell &= \theta^{-1}(g(0)),\\ \gamma_{\ell+1} &= h(1), & \delta_{\ell+1} &= \theta^{-1}(g(1)). \end{aligned}$$

be the θ-PCP instance $\mathcal{T}$.

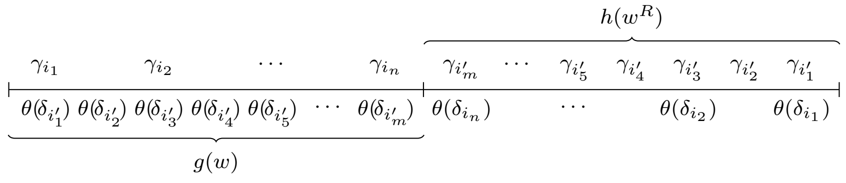

$h(w^R)$
γ_{i_1} γ_{i_2} $\cdots$ γ_{i_n} $\gamma_{i'_m}$ $\cdots$ $\gamma_{i'_5}$ $\gamma_{i'_4}$ $\gamma_{i'_3}$ $\gamma_{i'_2}$ $\gamma_{i'_1}$
$\theta(\delta_{i'_1})$ $\theta(\delta_{i'_2})$ $\theta(\delta_{i'_3})$ $\theta(\delta_{i'_4})$ $\theta(\delta_{i'_5})$ $\cdots$ $\theta(\delta_{i'_m})$ $\theta(\delta_{i_n})$ $\cdots$ $\theta(\delta_{i_2})$ $\theta(\delta_{i_1})$
$g(w)$

Figure 6: Transforming the solution $i_1, i_2, \ldots, i_n$ of the PCP instance $\mathcal{A}$ into the solution $i_1, i_2, \ldots, i_n, i'_m, i'_{m-1}, \ldots, i'_1$ of the θ-PCP instance $\mathcal{T}$; all variables are defined in the text.

First, let us show that if $\mathcal{A}$ has a solution than $\mathcal{T}$ has a solution as well. Let $i_1, i_2, \ldots, i_n \in A_\ell$ with $n \geq 1$ be a solution of the PCP instance $\mathcal{A}$ and let w be the word corresponding to this solution; i. e.,

$$w = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} = \beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}.$$

Figure 6 illustrates the following construction. Let $m = |w|$. For $j = 1, \ldots, m$ we let $i'_j = \ell$ if $w_{[j]} = 0$ and $i'_j = \ell + 1$ if $w_{[j]} = 1$; these indeces are chosen such that

$$\begin{aligned}
&\gamma_{i'_m}\gamma_{i'_{m-1}}\cdots\gamma_{i'_1} = h(w^R),\\
&\delta_{i'_m}\delta_{i'_{m-1}}\cdots\delta_{i'_1} = \theta^{-1}(g(w_{[m]}))\theta^{-1}(g(w_{[m-1]}))\cdots\theta^{-1}(g(w_{[1]})) = \theta^{-1}(g(w)).
\end{aligned}$$

The integer sequence $i_1, i_2, \dots, i_n,\ i'_m, i'_{m-1}, \dots, i'_1$ is a solution of the θ-PCP instance $f(\alpha)$ because

$$\begin{aligned}
\theta(\delta_{i_1}\cdots\delta_{i_n}\delta_{i'_m}\cdots\delta_{i'_1}) &= \theta(\delta_{i'_m}\cdots\delta_{i'_1}) &&\cdot\,\theta(\delta_{i_n})\cdots\theta(\delta_{i_1})\\
&= \theta(\theta^{-1}(g(w))) &&\cdot\,\theta(\theta^{-1}(h(\beta^R_{i_n})))\cdots\theta(\theta^{-1}(h(\beta^R_{i_1})))\\
&= g(w) &&\cdot\,h(\beta^R_{i_n})\cdots h(\beta^R_{i_1})\\
&= g(\alpha_{i_1})\cdots g(\alpha_{i_n}) &&\cdot\,h(w^R)\\
&= \gamma_{i_1}\cdots\gamma_{i_n} &&\cdot\,\gamma_{i'_m}\cdots\gamma_{i'_1}.
\end{aligned}$$

Vice versa, let $i_1, i_2, \dots, i_n \in A_{\ell+2}$ with $n \geq 1$ be a solution of the θ-PCP instance $\mathcal{T}$ and let w be the word corresponding to this solution, that is,

$$w = \gamma_{i_1}\gamma_{i_2}\cdots\gamma_{i_n} = \theta(\beta_{i_1}\beta_{i_2}\cdots\beta_{i_n}) = \theta(\beta_{i_n})\cdots\theta(\beta_{i_2})\theta(\beta_{i_1}).$$

Recall that for every word γ_{i_j} we have that either $\gamma_{i_j} \in g(A_2^+)$ (in case $i_j < \ell$) or $\gamma_{i_j} \in h(A_2)$ (in case $i_j \geq \ell$). Since $g(A_2)$ and $h(A_2)$ contain mutually distinct two-letter words, for every pair of letters $p = w_{[2r-1;2r]}$ with $r \in \mathbb{N}$: if $p \in g(A_2)$, then p is covered by a factor γ_{i_j} with $i_j < \ell$; and if $p \in h(A_2)$, then p equals to a factor γ_{i_j} with $i_j \geq \ell$. Symmetrically, for $p = w_{[2r-1;2r]}$ with $r \in \mathbb{N}$: if $p \in h(A_2)$, then p is covered by a factor $\theta(\delta_{i_j})$ with $i_j < \ell$; and if $p \in g(A_2)$, then p equals to a factor $\theta(\delta_{i_j})$ with $i_j > \ell$.

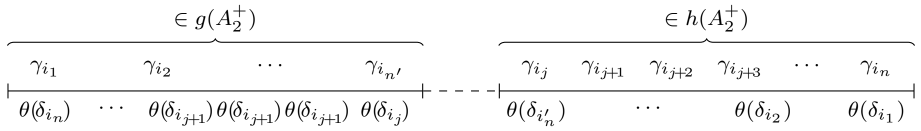


Figure 7: Transforming the solution $i_1, i_2, \dots, i_n$ of the θ-PCP instance $\mathcal{T}$ into the solution $i_1, i_2, \dots, i_{n'}$ of the PCP instance $\mathcal{A}$; all variables are defined in the text.

Consider the case where $i_1 < \ell$. Figure 7 illustrates the following construction. In this case, $\gamma_{i_1} = g(\alpha_{i_1})$ is a prefix of w and $\theta(\delta_{i_1}) = h(\beta^R_{i_1})$ is a suffix of w; thus, $w_{[1;2]} \in g(A_2)$ and $w_{[|w|-1;|w|]} \in h(A_2)$. Further, we obtain that $i_n \geq \ell$ because γ_{i_n} has to cover $w_{[|w|-1;|w|]} \in h(A_2)$. There exists an integer n' with $1 \leq n' < n$ such that $i_1, i_2, \dots, i_{n'} < \ell$ but $i_{n'+1} \geq \ell$. We will show that the sequence $i_1, i_2, \dots, i_{n'}$ is a solution of the PCP instance $\mathcal{A}$ by comparing the longest prefix of w which belongs to $g(A_2^+)$ with the longest suffix of w which belongs to $h(A_2^+)$. Let m be an even integer such that $w_{[1;m]} \in g(A_2^+)$ but $w_{[m+1;m+2]} \in h(A_2)$. Because $i_{n'+1}$ has to match with the first letter pair in w which belongs to $h(A_2)$, it is not difficult to see that

$$w_{[1;m]} = \gamma_{i_1}\gamma_{i_2}\cdots\gamma_{i_{n'}} = g(\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_{n'}}).$$

Because $w_{[1;m]} \in g(A_2^+)$ and $w_{[m+1;m+2]} \in h(A_2)$, there exists an integer $j < n$ such that $i_j, i_{j+1} \dots, i_n \geq \ell$, $i_{j-1} < \ell$, and

$$w_{[1;m]} = \theta(\delta_{i_j}\delta_{i_{j+1}} \cdots \delta_{i_n}) = \theta(\delta_{i_n}) \cdots \theta(\delta_{i_{j+1}})\theta(\delta i_j).$$

Due to the design of the word pairs $(\gamma_\ell, \delta_\ell)$ and $(\gamma_{\ell+1}, \delta_{\ell+1})$ and because

$$\theta(\delta_{i_n}) \cdots \theta(\delta_{i_{j+1}})\theta(\delta_{i_j}) = g(\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}})$$

is a prefix of w, we have that $\gamma_{i_j}\gamma_{i_{j+1}} \cdots \gamma_{i_n} = h((\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}})^R)$ is a suffix of w. Since $i_{j-1} < \ell$, we see that this suffix $h((\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}})^R)$ of w is preceded by a letter pair from $g(A_2)$. This implies that the suffix $\theta(\delta_{i_{n'}}) \cdots \theta(\delta_{i_2})\theta(\delta_{i_1})$ of w equals $h((\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}})^R)$. Therefore,

$$\begin{aligned} h((\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}})^R) &= \theta(\delta_{i_{n'}}) \cdots \theta(\delta_{i_2})\theta(\delta_{i_1}) \\ &= h(\beta_{i_{n'}}^R) \cdots h(\beta_{i_2}^R)h(\beta_{i_1}^R) \\ &= h((\beta_{i_1}\beta_{i_2} \cdots \beta_{i_{n'}})^R). \end{aligned}$$

We conclude that $\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_{n'}} = \beta_{i_1}\beta_{i_2} \cdots \beta_{i_{n'}}$ and, therefore, $i_1, i_2, \dots, i_{n'}$ is a solution of the PCP instance $\mathcal{A}$.

The case when $i_1 \geq \ell$ can be treated analogously, where we compare the longest prefix of w which belongs to $h(A_2^+)$ and the longest suffix of w which belongs to $g(A_2^+)$. In this case, there exists $n' \leq n$ such that $i_{n'}, i_{n'+1}, \dots, i_n$ is a solution of the PCP instance $\mathcal{A}$. □

We can utilize the θ-PCP in order to prove that it is undecidable whether or not a transducer is θ-input-altering, even for one-state transducers.

Corollary 34. *For every fixed antimorphic permutation θ over A_k^* with $k \geq 2$ it is undecidable whether or not a given (one-state) transducer is θ-input-altering.*

Proof. Let $\alpha_0, \alpha_1, \dots, \alpha_{\ell-1} \in A_k^+$ and $\beta_0, \beta_1, \dots, \beta_{\ell-1} \in A_k^+$ be the θ-PCP instance $\mathcal{A}$. We let $\mathbf{t}_\mathcal{A}$ be the one-state transducer shown in Fig. 8. Clearly, we have $y \in \mathbf{t}_\mathcal{A}(x)$ if and only if there exists an integer sequence $i_1, i_2, \dots, i_n \in A_\ell$ such that $x = \alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_n}$ and $y = \theta^2(\beta_{i_1})\theta^2(\beta_{i_2}) \cdots \theta^2(\beta_{i_n}) = \theta^2(\beta_{i_1}\beta_{i_2} \cdots \beta_{i_n})$; note that θ^2 is always morphic, even if θ is not.

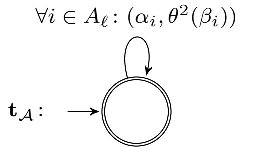

$\forall i \in A_\ell \colon (\alpha_i, \theta^2(\beta_i))$
$\mathbf{t}_\mathcal{A}\colon$

Figure 8: $\mathbf{t}_\mathcal{A}$ encodes the θ-PCP instance $\alpha_0, \alpha_1, \dots, \alpha_{\ell-1}$, $\beta_0, \beta_1, \dots, \beta_{\ell-1}$.

Recall that it is allowed for θ-input-altering transducers to accept the empty word pair $(\varepsilon, \varepsilon)$. We have $w \in \theta^{-1}(\mathbf{t}_\mathcal{A}(w))$ for some word $w \in A_k^+$ if and only if there exists an integer sequence $i_1, i_2, \dots, i_n$ such that

$$\alpha_{i_1}\alpha_{i_2} \cdots \alpha_{i_n} = w = \theta^{-1}\left(\theta^2(\beta_{i_1}\beta_{i_2} \cdots \beta_{i_n})\right) = \theta(\beta_{i_1}\beta_{i_2} \cdots \beta_{i_n}).$$

Therefore, $\mathbf{t}_{\mathcal{A}}$ is θ-input-altering if and only if the θ-PCP instance $\mathcal{A}$ has a solution. Theorem 33 concludes the proof. $\square$

8. A Hierarchy of DNA-related θ-transducer Properties

In [7–9] the authors consider numerous properties of languages inspired by reliability issues in DNA computing. As $\theta = \delta$ is an involution in the DNA setting, we assume that θ is antimorphic over A^* and $\theta^2 = \mathrm{id}$. The relationships between some of the defined 3-independent DNA-related properties are displayed in Fig. 9. All properties have in common that they forbid certain "constellations" of words. Consider a language $L \subseteq A^+$ and two words $uwv, \theta(xwy) \in A^+$ with $w \neq \varepsilon$ as shown in the top property in Fig. 9. The same type of notation can be employed for all properties in the figure, where some properties require that x, y, u, or v are empty, e. g., for $x = y = \varepsilon$ we obtain the θ-compliant property. In the case of θ-nonoverlapping all of x, y, u, v are empty:

(A) a language L is *θ-nonoverlapping* if for all $w \in A^+$, we have $w \notin L$ or $\theta(w) \notin L$. This is equivalent to requiring that $L \cap \theta(L) = \emptyset$.

For all the above properties, except θ-nonoverlapping, the language L has property P, if $uwv \in L$ and $\theta(xwy) \in L$ implies that $uvxy = \varepsilon$. For lack of a better term, in this section we refer to such properties P as *normal*. For example,

(B) a language L is *θ-compliant* if for all $w \in A^+$ and $u, v \in A^*$, we have $uwv, \theta(w) \in L \implies uv = \varepsilon$; and

(C) a language L is *θ-5′-overhang-free* if for all $w \in A^+$ and $u, y \in A^*$, we have $uw, \theta(wy) \in L \implies uy = \varepsilon$.

Previous papers considered *strict versions* P^{s} for some (but not all) of the above normal properties P, by adding the θ-nonoverlapping condition to the condition defining P. Here, we generalize the concept of strict properties P^{s}: if $uwv \in L$ and $\theta(xwy) \in L$, then L does not satisfy the strict property P^{s} (even if $uvxy = \varepsilon$). For example,

(D) a language L is *strictly θ-compliant* if for all $w \in A^+$ and for all $u, v \in A^*$, we have $uwv \notin L$ or $\theta(w) \notin L$; and

(E) a language L is *strictly θ-5′-overhang-free* if for all $w \in A^+$ and $u, y \in A^*$, we have $uw \notin L$ or $\theta(wy) \notin L$.

Note that θ-nonoverlapping is actually a strict property while its "normal version" would be the property that is trivially satisfied by every language in A^+.

Furthermore, we introduce the *weak version* of a normal property which generalizes properties like the (weakly) overlap-free property where it is allowed for a word to overlap with itself, but not with another word: for a language L which satisfies the weak property $P^{\mathbf{w}}$, if the words uwv and $\theta(xwy)$ belong to L, then $uvxy = \varepsilon$ or $uwv = \theta(xwy)$. For example,

(F) a language L is *weakly θ-5′-overhang-free* if for all $w \in A^+$ and $u, y \in A^*$, we have $uw, \theta(wy) \in L$ implies $uy = \varepsilon$ or $uw = \theta(wy)$.

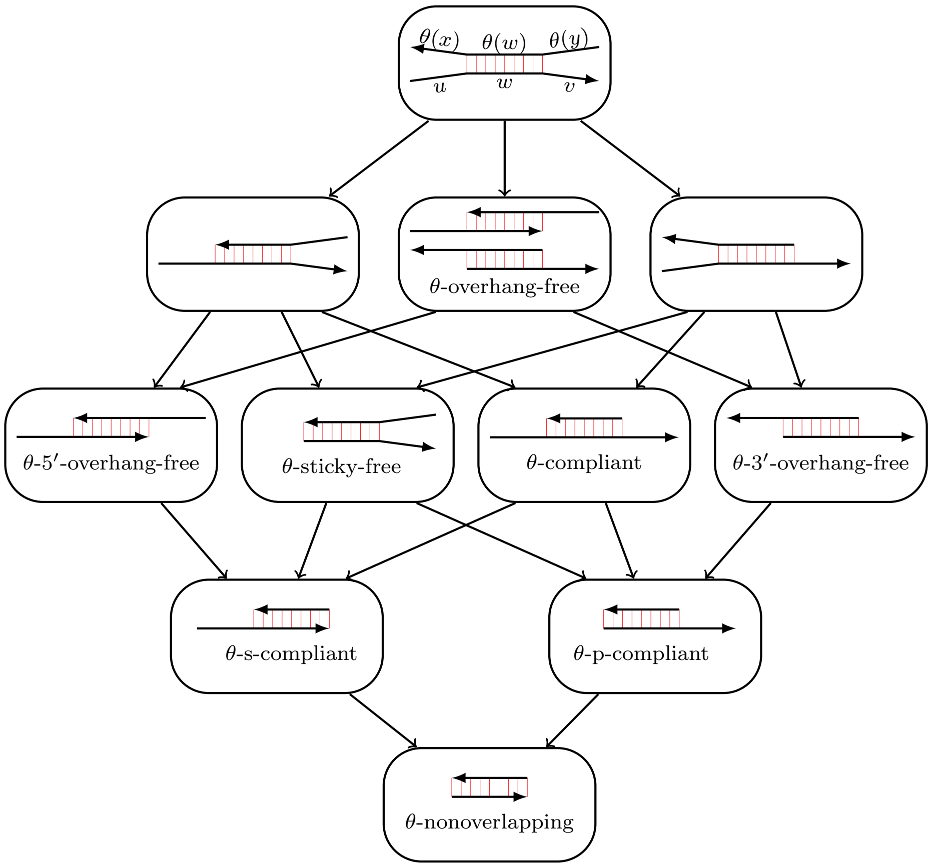


Figure 9: Correlation of various 3-independent DNA language properties—enriched version of Fig. 2 in [9]. For each property the forbidden constellation of words (or single strands) is depicted. Words are represented as arrows such that the first letter (the 5′-end) is the blunt end of the arrow and last letter (the 3′-end) is the arrow tip. Vertical lines between arrows represent bonding between θ-complementary parts of the two words.

Note that for some properties, like θ-compliant, the weak property $P^{\mathbf{w}}$ coincides with the (normal) property P.

If a language L satisfies the strict property $P^{\mathbf{s}}$, then it also satisfies the corresponding (normal) property P; and if L satisfies the (normal) property P, then it also satisfies the corresponding weak property $P^{\mathbf{w}}$. Furthermore, there is a normal, strict, and weak hierarchy of properties which is shown in Fig. 9, where θ-nonoverlapping only exists in the strict hierarchy. For all three hierarchies an arrow $P^{\mathbf{x}} \to Q^{\mathbf{x}}$ (for $\mathbf{x} \in \{\varepsilon, \mathbf{s}, \mathbf{w}\}$) between two properties $P^{\mathbf{x}}$ and $Q^{\mathbf{x}}$ means that if a language L satisfies property $P^{\mathbf{x}}$, then it also satisfies property $Q^{\mathbf{x}}$.

Let us discuss how these properties can be described as θ-transducer properties. The type of the property ($\mathcal{W}$-property or $\mathcal{S}$-property) and the type of the transducer (unrestricted, θ-input-altering, θ-input-preserving) is important when it comes to the complexity of the satisfaction problem and the decidability of the maximality problem; see Table 1. Firstly, observe that L is θ-nonoverlapping if L satisfies the θ-transducer property $\mathcal{S}_{\theta,\mathbf{t}_{\mathrm{id}}}$ where $\mathbf{t}_{\mathrm{id}}$ is a transducer realizing the identity relation. Since any strict property, including θ-nonoverlapping, is not satisfied by a singleton language $\{w\}$ that consists of one θ-palindrome $w = \theta(w)$, strict properties cannot be described as $\mathcal{S}$-properties by a θ-input-altering transducer or as $\mathcal{W}$-properties, according to Remark 10.

Figure 10 shows two families of transducers describing any of the DNA-related properties that we discussed in this section. Depending on whether or not u (resp., v, x, y) is empty one has to omit a set of edges in each transducer. The $\mathcal{S}$-properties $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{s}}$ are the strict properties, the $\mathcal{S}$-properties $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}}$ are the normal properties, and the $\mathcal{W}$-properties $\mathcal{W}_{\theta,\mathbf{t}_\mathbf{w}}$ are the weak properties. If we omit red and orange edges (i. e., $xy = \varepsilon$), then $\mathbf{t}_\mathbf{w}$ is θ-input-altering because the input word is strictly longer than the output word. Therefore, $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}} = \mathcal{W}_{\theta,\mathbf{t}_\mathbf{w}}$, i. e., the normal property coincides with the corresponding weak property. The case when all blue and green edges are omitted is symmetric when input and output swap roles. We demonstrate this construction in Examples 35 and 36.

Example 35. Let $\mathbf{t}_\mathbf{s}^C$ and $\mathbf{t}_\mathbf{w}^C$ be the two transducers that are obtained by omitting all red and orange edges in $\mathbf{t}_\mathbf{s}$ and $\mathbf{t}_\mathbf{w}$ (Fig. 10), respectively. Then $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{s}^C}$ is the strict θ-compliant property, whereas $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}^C}$ is the (normal) θ-compliant property. Since $\mathbf{t}_\mathbf{w}^C$ is θ-input-altering, $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}^C}$ is equal to $\mathcal{W}_{\theta,\mathbf{t}_\mathbf{w}^C}$ and the properties θ-compliant and weak θ-compliant coincide.

Example 36. Let $\mathbf{t}_\mathbf{s}^{5OF}$ and $\mathbf{t}_\mathbf{w}^{5OF}$ be the two transducers that are obtained by omitting all red and green edges in $\mathbf{t}_\mathbf{s}$ and $\mathbf{t}_\mathbf{w}$ (Fig. 10), respectively. Then $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{s}^{5OF}}$ is the strict θ-5′-overhang-free property, $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}^{5OF}}$ is the (normal) θ-5′-overhang-free property, and $\mathcal{W}_{\theta,\mathbf{t}_\mathbf{w}^{5OF}}$ is the weak θ-5′-overhang-free property.

For $\theta = \delta$, the DNA involution, observe that the word $\mathtt{AACG}$ can have a θ-5′-overhang with itself (as $x = \mathtt{AA}$, $w = \theta(w) = \mathtt{CG}$, and $y = \mathtt{TT} = \theta(\mathtt{AA})$). As expected, $\mathbf{t}_\mathbf{w}^{5OF}$ does realize the word pair $(\mathtt{AACG}, \mathtt{CGTT})$ and, therefore, the language $\{\mathtt{AACG}\}$ does not satisfy the (normal) θ-5′-overhang-free property $\mathcal{S}_{\theta,\mathbf{t}_\mathbf{w}^{5OF}}$, however, $\{\mathtt{AACG}\}$ does satisfy the weak θ-5′-overhang-free property $\mathcal{W}_{\theta,\mathbf{t}_\mathbf{w}^{5OF}}$.

Lastly, note that the (strict, weak) θ-overhang-free property is different from the other properties in Fig. 9 in so far that it forbids two word constellations: θ-5′-overhangs and θ-3′-overhangs. This property can be described by a transducer which contains two components, where one component covers the θ-5′-overhangs and the other component covers the θ-3′-overhangs.

9. Conclusions

We have defined a transducer-based method for describing DNA code properties which is strictly more expressive than the trajectory method. In doing so, the satisfaction question remains efficiently decidable. We have demonstrated

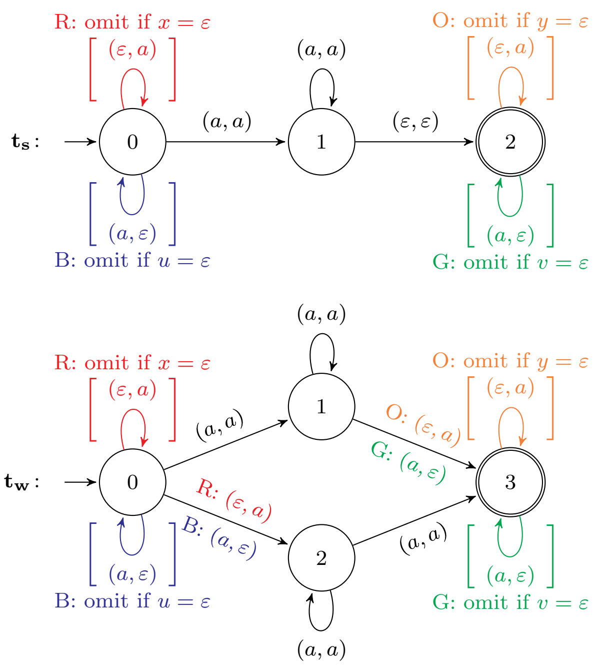


Figure 10: The family of transducers which describes all properties shown in Fig. 9. Each of the two transducer families describes 16 different transducers: We can either omit or include each of the red, orange, blue and green edges. These edges are omitted depending on the property that is described, for example, omit all red edges if $x = \varepsilon$ in Fig. 9. We have included initial letters of colour names (B, G, O, R) to accommodate readers using a non-coloured copy of this figure.

with examples that the new method is capable of describing many 3-independent properties having various subtle differences.

The topic of description methods for code properties requires further attention. One important aim is the implementation of the concepts involved. Already the software [21] for classic properties can be used for answering the satisfaction problem for all $\mathcal{S}_{\theta,\mathbf{t}}$ properties introduced here, assuming one has prepared automata for both $\theta(L)$ and L. A future version of [21] could include new $\mathcal{W}_{\theta,\mathbf{t}}$ properties, and a future version of [23] should allow computation of $\theta(L)$ from given θ and automaton for L. Another important aim of this research program is to increase the expressive power of description methods. The formal method of [18] is quite expressive, using a certain type of first order formulae to describe properties. It could perhaps be further worked out in a way that some of these formulae can be mapped to transducers. We also note that if the defining method is too expressive then even the satisfaction problem could become undecidable, as in the method of multiple sets of trajectories in [37].

The maximality problem for some types of properties is decidable, but it is undecidable for others. While some versions of the maximality question for trajectory properties are decidable, the case of any given pair of regular trajectories and any given regular language is not addressed in [22], so we consider this to be an interesting problem to solve.

The maximality problems are phrased in terms of any fixed antimorphic permutation. This direction of generalizing decision questions is also applied to the classic Post Correspondence Problem, where we demonstrate that it remains undecidable. A consequence of this is that the question of whether a given transducer is θ-input-altering is also undecidable. It is interesting to note that if, instead of fixing θ, we fix the transducer $\mathbf{t}$ to be the identity, or the transducer defining the $\mathcal{S}$-property $\mathcal{H}$ (see Fig. 3 in Sect. 4), then the question of whether or not

$$\theta(L) \cap \mathbf{t}(L) = \emptyset$$

is decidable (given any regular language L and antimorphic permutation θ).

References

[1] L. Kari, S. Konstantinidis, S. Kopecki, Transducer descriptions of DNA code properties and undecidability of antimorphic problems, in: J. Shallit, A. Okhotin (Eds.), Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings, 2015, pp. 141–152.

[2] H. Shyr, G. Thierrin, Codes and binary relations, in: M. P. Malliavin (Ed.), Séminaire d'Algèbre Paul Dubreil, Paris 1975–1976 (29ème Année), Vol. 586 of Lecture Notes in Mathematics, 1977, pp. 180–188.

[3] H. Shyr, Free Monoids and Languages, 2nd Edition, Hon Min Book Company, Taichung, 1991.

[4] H. Jürgensen, S. Konstantinidis, Codes, in: Rozenberg and Salomaa [28], pp. 511–607.

[5] J. Berstel, D. Perrin, C. Reutenauer, Codes and Automata, Cambridge University Press, 2009.

[6] E. Baum, DNA sequences useful for computation, in: 2nd DIMACS Workshop on DNA-based computers, Princeton University, 1996, pp. 122–127.

[7] L. Kari, R. Kitto, G. Thierrin, Codes, involutions, and DNA encodings, in: Formal and Natural Computing, Springer, 2002, pp. 376–393.

[8] S. Hussini, L. Kari, S. Konstantinidis, Coding properties of DNA languages, Theoretical Computer Science 290 (2003) 1557–1579.

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

[10] G. Mauri, C. Ferretti, Word design for molecular computing: a survey, in: DNA 9, LNCS 7505, Springer-Verlag, 2004, pp. 37–47.

[11] N. Jonoska, K. Mahalingam, J. Chen, Involution codes: with application to DNA coded languages, Natural Computing 4 (2005) 141–162.

[12] L. Kari, S. Konstantinidis, P. Sosík, On properties of bond-free DNA languages, Theoretical Computer Science 334 (2005) 131–159.

[13] L. Kari, S. Konstantinidis, P. Sosík, Bond-free languages: formalizations, maximality and construction methods, IJFCS 16 (2005) 1039–1070.

[14] N. Jonoska, L. Kari, K. Mahalingam, Involution solid and join codes, Fundamenta Informaticae 86 (2008) 127–142.

[15] D. Genova, K. Mahalingam, Generating DNA code words using forbidding and enforcing systems, in: Theory and Practice of Natural Computing, LNCS 7505, Springer-Verlag, 2012, pp. 376–393.

[16] C.-M. Fan, J.-T. Wang, C.-C. Huang, Some properties of involution binary relations, Acta Informatica DOI 10.1007/s00236-014-0208-8.

[17] R. Zaccagnino, R. Zizza, C. Zottoli, Testing dna code words properties of regular languages, Theoretical Computer Science 608 (2015) 84–97.

[18] H. Jürgensen, Syntactic monoids of codes, Acta Cybernetica 14 (1999) 117–133.

[19] M. Domaratzki, Trajectory-based codes, Acta Informatica 40 (2004) 491–527.

[20] K. Dudzinski, S. Konstantinidis, Formal descriptions of code properties: decidability, complexity, implementation, IJFCS 23:1 (2012) 67–85.

[21] FAdo, Tools for formal languages manipulation, URL address: `http://fado.dcc.fc.up.pt/` Accessed in November, 2015.

[22] M. Domaratzki, Bond-free DNA language classes, Natural Computing 6 (2007) 371–402.

[23] LaSer, Independent LAnguage SERver, URL address: `http://laser.cs.smu.ca/independence/` Accessed in November, 2015.

[24] D. Kephart, J. LeFevre, CODEGEN: The generation and testing of DNA code words, in: IEEE Congress on Evolutionary Computation, CEC2004, 2004, pp. 1865–1873.

[25] D. Bärmann, Aufzählen von DNA-codes, Master's thesis, Universität Potsdam, in German (2006).

[26] N. Aubert, A DNA-based finite state transducer, Master's thesis, CNRS, bioinformatics (2010).

[27] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[28] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. I, Springer-Verlag, Berlin, 1997.

[29] J. Berstel, Transductions and Context-Free Languages, B.G. Teubner, Stuttgart, 1979.

[30] S. Yu, Regular languages, in: Rozenberg and Salomaa [28], pp. 41–110.

[31] S. Konstantinidis, Transducers and the properties of error-detection, error-correction and finite-delay decodability, Journal Of Universal Computer Science 8 (2002) 278–291.

[32] L. Kari, P. Sosík, Aspects of shuffle and deletion on trajectories, Theoretical Computer Science 332 (2005) 47–61.

[33] J. Sakarovitch, Elements of Automata Theory, Cambridge University Press, Berlin, 2009.

[34] C. Allauzen, M. Mohri, Efficient algorithms for testing the twins property, Journal of Automata, Languages and Combinatorics 8 (2) (2003) 117–144.

[35] T. Head, A. Weber, Deciding code related properties by means of finite transducers, pp. 260–272.

[36] L. Stockmeyer, A. Meyer, Word problems requiring exponential time (preliminary report), in: Proceedings of the 5th annual ACM symposium on Theory of computing, ACM, 1973, pp. 1–9.

[37] M. Domaratzki, K. Salomaa, Codes defined by multiple sets of trajectories, Theoretical Computer Science 366 (2006) 182–193.