Theoretical Computer Science 550 (2014) 90–99

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Decidability of involution hypercodes

Da-Jung Cho, Yo-Sub Han *, Sang-Ki Ko

Department of Computer Science, Yonsei University, Seoul 120-749, Republic of Korea

ARTICLE INFO

Article history:
Received 5 March 2013
Received in revised form 4 April 2014
Accepted 15 July 2014
Available online 23 July 2014
Communicated by J. Karhumäki

Keywords:
Involution hypercodes
Decidability
DNA codes
Edit-distance

ABSTRACT

Given a finite set X of strings, X is a hypercode if a string in X is not a subsequence of any other string in X. We consider hypercodes for involution codes, which are useful for DNA strand design, and define an involution hypercode. We then tackle the involution hypercode decidability problem; that is, to determine whether or not a given language is an involution hypercode. Based on the hypercode properties, we design a polynomial runtime algorithm for regular languages. We also prove that it is decidable whether or not a context-free language is an involution hypercode. Note that it is undecidable for some other involution codes such as involution prefix codes, suffix codes, and k-intercodes.

1. Introduction

In DNA computing and bioinformatics, a major topic is the encoding of DNA information based on the DNA code properties [4,6,7]. Many researchers have investigated the algebraic and code-theoretic properties of DNA encoding based on formal language theory [17,19,20,24,26]. DNA strands consist of four types of bases: *adenine* (A), *guanine* (G), *cytosine* (C), and *thymine* (T). The DNA alphabet, $\Delta = \{A, G, C, T\}$, encodes the characteristics of every cellular organism. Hussini et al. [17] generated DNA code words that avoid undesirable bonds and considered the decidability problem for these DNA code words. Jonoska et al. [19] introduced involution codes based on natural involution mapping, θ; $A \leftrightarrow T$ and $G \leftrightarrow C$. They defined different types of involution codes such as θ-prefix-code, θ-suffix-code, θ-bifix-code, θ-infix-code, θ-outfix-code, θ-intercode, θ-comma-free-code, and θ-strict-code, and investigated the algebraic properties of each involution code (θ-code). Jonoska and her co-authors [20] also extended the notion of solid codes and join codes to involution solid codes (θ-solid) and involution join codes (θ-join). Kari and Mahalingam [24] continued the study of the algebraic properties of DNA languages that avoid intermolecular cross hybridizations and made several observations on the closure properties of these languages. Kari et al. [25] applied the hairpin-free DNA structure to algebraic codes.

DNA transcription is the process of creating a complementary RNA copy of a sequence of DNA, and DNA translation produces a specific amino acid chain using mRNA produced by the transcription [30]. When a DNA replication occurs in the process of transcription, errors may occur at any time and these errors cause a mutation. The errors caused by the addition or deletion of a nucleotide shift the specific codons in the mRNA during the process of DNA translation. We call this type of mutation a *frameshift mutation*. For example, in Fig. 1, we have a mutated amino acid, Gly, and a chain termination because of the frameshift caused by the newly added DNA C between C and A. This demonstrates that a frameshift gives rise to a protein synthesized with an added or deleted nucleotide, which is often shortened and nonfunctional.

* Corresponding author.
E-mail addresses: dajung@cs.yonsei.ac.kr (D.-J. Cho), emmous@cs.yonsei.ac.kr (Y.-S. Han), narame7@cs.yonsei.ac.kr (S.-K. Ko).

http://dx.doi.org/10.1016/j.tcs.2014.07.016
0304-3975/

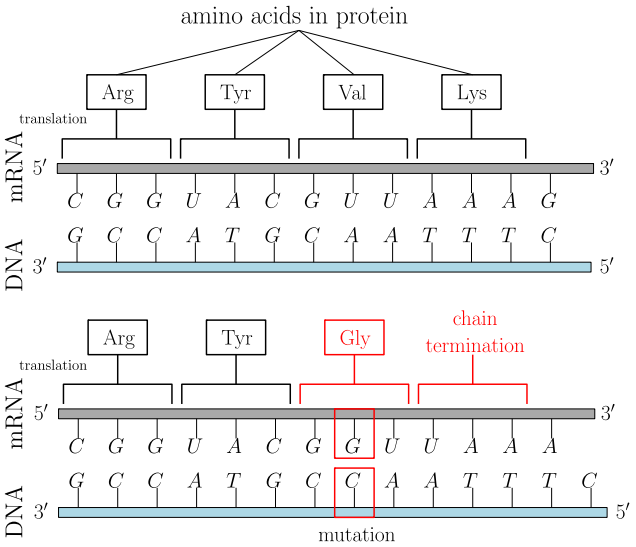

Fig. 1. An example of frameshift mutation.

Frameshift mutations motivate us to examine the operations of insertion and deletion of DNA. We generalize these operations and allow multiple insertions or deletions. This leads us to study involution hypercodes. In other words, we investigate DNA codes that do not allow multiple insertions or deletions. In coding theory, a set X of strings is a hypercode if a string is not a subsequence of any other string in X [31]. Thus, a hypercode is always finite. However, a θ-hypercode may not be finite since an involution hypercode (θ-hypercode) is defined over an involution mapping, θ, of a language.

One can efficiently decide whether or not a regular language is a certain code, whereas the same question is often undecidable for a context-free language [1,11,12,21]. Recently, Jonoska et al. [20] showed that it is decidable whether or not a regular language is a certain type of an involution code. Kephart and Lefevre [26] designed a polynomial algorithm that checks whether or not a regular language is θ-infix and θ-comma-free using the square automata. Thus, it is natural to investigate the decidability of θ-hypercodes for regular languages and context-free languages.

We briefly recall several involution codes and their properties in Section 2. Then, in Section 3, we formally define θ-hypercodes and study their properties. Next, we design two algorithms that determine whether or not a given regular language is a hypercode or a θ-hypercode based on 1) the intersection emptiness test of two FAs and 2) the alignment automaton [2] of two FAs. We examine the decidability problem for context-free languages and prove the decidability for θ-hypercodes and the undecidability for some other θ-codes. We mention a possible future direction and conclude the paper in Section 4.

2. Preliminaries

Let Σ be a finite alphabet and Σ^* be a set of all strings over Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language, the symbol λ denotes the null string, and Σ^+ denotes $\Sigma^* \setminus \{\lambda\}$. For two strings x and y over Σ, we say that x is a *prefix* of y if $y = xz$ for a string $z \in \Sigma^*$. We say that x is a *proper prefix* of y if x is a prefix of y and $x \neq y$. Similarly, x is a *suffix* of y if $y = zx$ for some string $z \in \Sigma^*$. We define x to be an *infix* (or substring) of y if $y = uxv$ for two strings $u, v \in \Sigma^*$.

A finite-state automaton (FA) M is specified by a tuple $M = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. Let $|Q|$ be the number of states in Q, and $|\delta|$ be the number of transitions in δ. Then the size $|M|$ of M is $|Q| + |\delta|$. Given a transition $\delta(p, a) = q$, we say that p has an *out-transition* and q has an *in-transition*. We define M to be *non-returning* if the start state of M does not have any in-transitions, and M to be *non-exiting* if a final state of M does not have any out-transition. A string x over Σ is accepted by M if there is a labeled path from s to a final state such that this path reads x. We call this path an *accepting path*. Then, the language $L(M)$ of M is the set of all strings spelled out by accepting paths in M. We assume that M has only useful states; that is, each state of M appears in an accepting path.

Given a language L over Σ, let

$$\mathcal{P}(L) = \{u \mid uv \in L \text{ for some } u \in \Sigma^* \text{ and } v \in \Sigma^+\} \quad \text{and}$$
$$\mathcal{S}(L) = \{u \mid vu \in L \text{ for some } u \in \Sigma^* \text{ and } v \in \Sigma^+\}.$$

For a language L over Σ, we define a language L to be

- *prefix-free* if $L \cap L\Sigma^+ = \emptyset$;
- *suffix-free* if $L \cap \Sigma^+ L = \emptyset$;
- *infix-free* if $\Sigma^* L \Sigma^+ \cap L = \emptyset$ and $\Sigma^+ L \Sigma^* \cap L = \emptyset$;
- a *k-intercode* if $L^{k+1} \cap \Sigma^+ L^k \Sigma^+ = \emptyset$ for $k \geq 1$;
- *overlap-free* if $\mathcal{P}(L) \cap \mathcal{S}(L) = \emptyset$;
- a *solid code* if L is infix-free and $\mathcal{P}(L) \cap \mathcal{S}(L) = \emptyset$.

We define a mapping $\alpha : \Sigma^* \to \Sigma^*$ to be a *morphism* of Σ^* if $\alpha(uv) = \alpha(u)\alpha(v)$ for all $u, v \in \Sigma^*$. Similarly, we define α to be an *antimorphism* of Σ^* if $\alpha(uv) = \alpha(v)\alpha(u)$.

There are three ways of combining two DNA strands into a single DNA strand in DNA encoding [24]:

1. The DNA mirror image function μ, which is an antimorphism: $\mu(uv) = \mu(v)\mu(u)$ for $u, v \in \Sigma^*$.
2. The DNA complementary image function γ, which is a morphism: $\gamma(A) = T, \gamma(C) = G$ for $A, T, G, C \in \Delta$.
3. The Watson–Crick complement, the composite of the mirror image function and the complementary image function.

We use the Watson–Crick involution (unless mentioned otherwise, we just call this "involution" for simplicity), $\theta : \Sigma^* \to \Sigma^*$, where θ is an antimorphic and θ^2 is an identity mapping.

Definition 2.1. (See Jonoska et al. [20].) Let $\theta : \Sigma^* \to \Sigma^*$ be an involution. Given a language L over Σ, we say that L is

- θ-*strict* if $L \cap \theta(L) = \emptyset$;
- θ-*prefix* if $L \cap \theta(L)\Sigma^+ = \emptyset$;
- θ-*suffix* if $L \cap \Sigma^+\theta(L) = \emptyset$;
- θ-*infix* if $\Sigma^*\theta(L)\Sigma^+ \cap L = \emptyset$ and $\Sigma^+\theta(L)\Sigma^* \cap L = \emptyset$;
- a θ–*k-intercode* if $L^{k+1} \cap \Sigma^+\theta(L^k)\Sigma^+ = \emptyset$;
- θ-*overlap-free* if $\mathcal{P}(L) \cap \mathcal{S}(\theta(L)) = \emptyset$ and $\mathcal{S}(L) \cap \mathcal{P}(\theta(L)) = \emptyset$;
- a θ-*solid code* if L is θ-infix and θ-overlap-free.

The reader may refer to the textbooks [16,33] for complete knowledge in automata theory, and Berstel et al. [1] or Jürgensen and Konstantinidis [21] for code theory.

3. Decidability of involution hypercodes

Given a string $x = x_1x_2\cdots x_n$, a string $z = z_1z_2\cdots z_m$ is a *subsequence* of x, where $n, m \geq 1$ if there exists a strictly increasing sequence $(i_1, i_2, \ldots, i_m)$ of indices of x such that $x_{i_j} = z_j$ for all $j = 1, 2, 3, \ldots, m$. If $z \neq x$, we say that z is a proper subsequence. We then say that x is a *supersequence* of z.

Definition 3.1. (See Shyr and Thierrin [31].) Given a language L, we define L to be a *hypercode* if a string in L is not a subsequence of any other string in L.

In Definition 2.1, Jonoska et al. [20] relied on proper prefix, suffix and infix to define the corresponding involution codes; for instance, L is θ-prefix if a string $w \in \theta(L)$ is not a proper prefix of any string in L. Similar to their definition, we define a θ-hypercode as follows:

Definition 3.2. Let $\theta : \Sigma^* \to \Sigma^*$ be an involution. Given a language L over Σ, we define L to be a θ-*hypercode* if no string in $\theta(L)$ is a *proper subsequence* of a string in L.

The decidability problem for hypercodes (or θ-hypercodes) is defined as follows: Given a language L, determine whether or not L is a hypercode (or a θ-hypercode). Note that a hypercode is always finite [31] whereas a θ-hypercode may not be finite. For instance, $L = L(a^+)$ is not a hypercode but is a θ-hypercode when $\theta(a) = b$. This is because a θ-hypercode is defined over two languages L and $\theta(L)$ while a hypercode is defined over L. Since we consider proper subsequences for defining θ-hypercodes, we need an algorithm that checks the existence of a string $x \in L$ that is a proper subsequence of a string $y \in \theta(L)$.

3.1. Regular hypercodes and involution regular hypercodes

We first consider the hypercode decision problem when L is regular. Since a hypercode is finite [31], an input is always finite and, thus, it is decidable by checking if one string is a subsequence of another string in the set in polynomial time. However, it is certainly undesirable to compare all pairs of strings for checking.

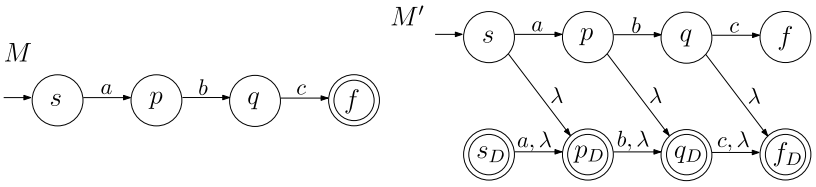

Fig. 2. An example of constructing an FA M' accepting proper subsequences of $L(M)$.

Before we start designing an algorithm, we briefly recall related research on subsequences and supersequences. Gruber et al. [8,9] considered the *Higman–Haines sets* [10] and defined DOWN(L) and UP(L):

$$\text{DOWN}(L) = \left\{ v \in \Sigma^* \;\middle|\; v \text{ is a subsequence of } w \in L \right\},$$
$$\text{UP}(L) = \left\{ v \in \Sigma^* \;\middle|\; v \text{ is a supersequence of } w \in L \right\}.$$

Then, they demonstrated that given an FA M of n states, one can find an NFA M' of at most n states accepting DOWN($L(M)$). Recently, Okhotin [28] examined sets of *scattered substrings* (namely, DOWN(L)) and *scattered superstrings* (namely, UP(L)) of a language L. He showed that given a DFA of n states for L, a DFA for DOWN(L) needs $2^{\frac{n}{2}-2}$ states, and a DFA for UP(L) needs $2^{n-2}+1$ states. When L is a context-free language, it is known that both DOWN(L) and UP(L) are regular [18,32]. Note that all of these studies consider subsequences and supersequences for DOWN(L) and UP(L), respectively.

For simplicity, we assume that an input FA M has no λ-transitions—We can always transform an n-state NFA with λ-transitions to an equivalent n-state NFA without λ-transitions [16]. We also assume that an input FA M is non-exiting since a hypercode is prefix-free [31]. If a final state of M has an out-transition, then we immediately know that $L(M)$ is not a hypercode. Otherwise, we can merge all final states into a single final state since they are all equivalent. Therefore, we assume that there is a single final state in M. Furthermore, we assume that M has no cycles since hypercodes are always finite.

Given a set Q of states, let $Q_D = \{q_D \mid q \in Q\}$. Given an FA $M = (Q, \Sigma, \delta, s, f)$, Gruber et al. [8] showed that $M_D = (Q_D, \Sigma, \delta_D, s_D, Q_D)$ accepts DOWN($L(M)$), where δ_D is defined as follows: for each a transition $\delta(p,a) = q$ in M, $\delta_D(p_D, a) = q_D$ and $\delta_D(p_D, \lambda) = q_D$.

Note that we construct M_D by adding λ-transition to all transitions in δ and make all states to be final states in M. We now slightly modify the construction by Gruber et al. [8] and make an FA $M' = (Q \cup Q_D, \Sigma, \delta', s, Q_D)$ accepting proper subsequences of $L(M)$ as follows: First, given M, we construct M_D as described above. Then, for each transition $\delta(p,a) = q$ in M, we add a λ-transition from p to q_D in δ'. We also add all transitions of M and M_D in δ'. Next, we make all states in M to be non-final states. Fig. 2 illustrates an example construction for M'.

The λ-transition from Q to Q_D in M' guarantees that M only accepts proper subsequences of a string of $L(M)$. Based on M' we establish the following result:

Theorem 3.3. *Given an FA $M = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(M)$ is a hypercode in $O(|M|^2)$ worst-case time.*

Proof. Given an FA M, we can construct an FA M' recognizing all proper subsequences of $L(M)$, where $|M'| = O(|M|)$. Since we can check the intersection emptiness of M and M' in $O(|M||M'|)$ time [16,33], we can determine whether or not $L(M)$ is a hypercode in $O(|M|^2)$ time. □

Next, we tackle the decision problem for θ-hypercode. Note that for θ-hypercodes, an input language is not necessarily finite. The decidability has already been proved implicitly in the literature: Domaratzki [5] defined that for any $T \subseteq \{0,1\}^*$, a language L is a T-code if $L \neq \emptyset$ and $(L \sqcup\sqcup_T \Sigma^+) \cap L = \emptyset$, where $\sqcup\sqcup_T$ is a string operation defined by shuffle on trajectories [15]. Note that for $T = (0+1)^*$ if L is a T-code, then L is a hypercode [5]. Moreover, Mateescu et al. [15] showed that if L_1, L_2, and T are regular, then $L_1 \sqcup\sqcup_T L_2$ is also regular. Therefore, it is decidable whether or not L is a θ-hypercode by checking whether or not the intersection of two regular languages $(L \sqcup\sqcup_T \Sigma^+)$ and $\theta(L)$ is empty, where $T = (0+1)^*$. Note that this is equivalent to a *Higman–Haines set* [8,9] if we consider Σ^* instead of Σ^+. One remaining question is how quickly we can decide θ-hypercodes.

We present two similar yet different algorithms that decide θ-hypercodes in polynomial time in the size of an input FA. The first algorithm is based on the intersection emptiness test and the second algorithm is based on the edit-distance. For the sake of simple explanation, we assume that an input FA M has a single final state. If not, we can always compute an equivalent FA with a single final state as follows: Given such $M = (Q, \Sigma, \delta, s, F)$, we introduce a new final state f' and modify the transition function δ' based on δ as

$$\delta'(p,a) = \begin{cases} \{q \mid q \in \delta(p,a)\} & \text{if } \delta(p,a) \cap F = \emptyset, \\ \{q \mid q \in \delta(p,a)\} \cup \{f'\} & \text{otherwise,} \end{cases}$$

where $p, q \in Q$ and $a \in \Sigma$. Then $L(M) = L(M')$.

Before we establish the solution for θ-hypercode decidability problem, we first show that it is sufficient to consider the case in which a string $x \in \theta(L(M))$ is a subsequence of a string $y \in L(M)$.

Observation 3.4. *A string $x \in \theta(L)$ is a subsequence of string $y \in L$ if and only if $\theta(x) \in L$ is a subsequence of $\theta(y) \in \theta(L)$.*

Observation 3.4 guarantees that if there is no pair of strings $x \in \theta(L)$ and $y \in L$ such that x is a subsequence of y, then we can conclude that L is a θ-hypercode. Now we construct an FA M_θ that accepts $\theta(L(M))$ from an FA M. Given an FA $M = (Q, \Sigma, \delta, s, f)$, we define $M_\theta = (Q, \Sigma, \delta_\theta, f, s)$, where δ_θ is defined as follows: For each transition $\delta(p, a) = q$ in M, we have $\delta_\theta(q, \theta(a)) = p$. Once we have M_θ, we can construct M'_θ that accepts proper subsequences of $L(M_\theta)$ in $O(|M_\theta|)$ time as before. Therefore, using a similar approach in the proof of Theorem 3.3, we can determine whether or not $L(M)$ is a θ-hypercode efficiently.

Theorem 3.5. *Given an FA $M = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(M)$ is a θ-hypercode in $O(|M|^2)$ worst-case time.*

3.2. Edit-distance and hypercodes

In Section 3.1, we have presented algorithms for deciding whether or not a given regular language is a hypercode or a θ-hypercode. Now we introduce an edit-distance approach to solve the same problem. The edit-distance between regular languages is to compute the minimum cost of transforming a string in one language to a string in the other language [3,23,27]. We can determine whether or not a regular language L is a θ-hypercode by computing the edit-distance between two regular languages L and $\theta(L)$. Given two regular languages L and R, we can construct an alignment FA $\mathcal{A}(L, R)$ that accepts all alignments transforming all strings in L into all strings in R.

An alignment or an edit string is defined as a sequence of edit operations. We use $\Omega = \{(a \to b) \mid a, b \in \Sigma \cup \{\lambda\}\}$ to denote an alphabet of all edit operations. There are three types of edit operations *insertion*, *deletion*, and *substitution*. For example, an edit operation $(a \to \lambda)$ means deleting a, and an edit operation $(\lambda \to a)$ means inserting a. Lastly, $(a \to b)$ is an edit operation for substituting a with b. We call a string $w \in \Omega^*$ an *edit string* or an *alignment*. Let a morphism h between Ω^* and $\Sigma^* \times \Sigma^*$ be

$$h\big((a_1 \to b_1)(a_2 \to b_2) \cdots (a_n \to b_n)\big) = (a_1 \cdots a_n, b_1 \cdots b_n).$$

Now we can re-define the edit string using h as follows:

Definition 3.6. An edit string w is a sequence of edit-operations transforming a string x into a string y if and only if $h(w) = (x, y)$.

We construct an alignment FA for determining whether or not a given regular language L is a θ-hypercode. Given an FA $M = (Q, \Sigma, \delta, s, f)$ and an involution mapping θ, we first construct $M_\theta = (Q', \Sigma, \delta', s', f')$ that accepts $L(\theta(M))$. Here we only use deletion and substitution operations $\Omega = \{(a \to a) \mid a \in \Sigma\} \cup \{(a \to \lambda) \mid a \in \Sigma\}$ as an alphabet for an alignment FA $\mathcal{A}(M, M_\theta)$ since a subsequence is obtained by deleting one or more characters from string $x \in L(M)$. Here the substitutions can only substitute a character with the same one. We call these operations *identical substitutions*. Then, we construct an alignment FA $\mathcal{A}(M, M_\theta)$ as follows:

$$\mathcal{A}(M, M_\theta) = (Q_\mathcal{A}, \Omega, \delta_\mathcal{A}, s_\mathcal{A}, f_\mathcal{A}),$$

where

- $Q_\mathcal{A} = Q \times Q'$ is a set of states,
- $\Omega = \{(a \to a) \mid a \in \Sigma\} \cup \{(a \to \lambda) \mid a \in \Sigma\}$ is an alphabet of edit operations,
- $\delta_\mathcal{A}((p_i, p'_i), (a \to a)) = \{(p_{i+1}, p'_{i+1}) \mid p_{i+1} \in \delta(p_i, a), p'_{i+1} \in \delta'(p'_i, a), a \in \Sigma\}$ is a transition function for identical substitutions,
- $\delta_\mathcal{A}((p_i, p'_i), (a \to \lambda)) = \{(p_{i+1}, p'_i) \mid p_{i+1} \in \delta(p_i, a), a \in \Sigma\}$ is a transition function for deletions,
- $s_\mathcal{A} = (s, s')$ is the start state,
- $f_\mathcal{A} = (f, f')$ is the final state.

Note that an alignment FA $\mathcal{A}(M, M_\theta)$ simulates an FA M and an FA M_θ simultaneously with the same input character for identical substitutions. Moreover, $\mathcal{A}(M, M_\theta)$ simulates an FA M by reading $a \in \Sigma$ while M_θ consumes λ for deletion operations.

Theorem 3.7. *Let $\theta : \Sigma^* \to \Sigma^*$ be an involution and an FA $M = (Q, \Sigma, \delta, s, f)$. Let $\mathcal{A}(M, M_\theta) = (Q_\mathcal{A}, \Omega, \delta_\mathcal{A}, s_\mathcal{A}, f_\mathcal{A})$ be the alignment FA. Then $L(M)$ is a θ-hypercode if and only if there is no path from $(i_1, j_1) \to \cdots \to (i_k, j_k)$ in $\mathcal{A}(M, M_\theta)$ that satisfies the following conditions:*

1. $(i_1, j_1) = (s, s')$ *and* $(i_k, j_k) = (f, f')$.
2. *There exists at least one deletion operation in an accepting path.*

Proof. ($\Longrightarrow$) Assume that there is an accepting path from (s, s') to (f, f') that has at least one deletion operation. Since (s, s') is the start state and (f, f') is a final state, there exists a corresponding accepting sequence in $\mathcal{A}(M, M_\theta)$. Let $X_w = (p_0, p'_0) \xrightarrow{(a_1 \to b_1)} (p_1, p'_1) \xrightarrow{(a_2 \to b_2)} \cdots \xrightarrow{(a_k \to b_k)} (p_k, p'_k)$ be a sequence of an accepting path, where $p_i \in Q$, $p'_i \in Q'$, $(a_i \to b_i) \in \Omega$ and $1 \le i \le k$. Now we know that the string $x = a_1 \cdots a_k$ is in $L(M)$ and the string $y = b_1 \cdots b_k$ is in $L(M_\theta)$. By the construction of $\mathcal{A}(M, M_\theta)$, b_i should be either equal to a_i or λ for $1 \le i \le k$. Since this path spells out at least one b_i $(= \lambda)$, the string $y = b_1 \cdots b_k$ is a proper subsequence of $x = a_1 \cdots a_k$—a contradiction.

($\Longleftarrow$) Assume that $L(M)$ is not a θ-hypercode. Then, there exists $y = b_1 \cdots b_k \in L(M_\theta)$ that is a proper subsequence of $x = a_1 \cdots a_k \in L(M)$. Thus, there exists an accepting sequence $X_w = (p_0, p'_0) \xrightarrow{(a_1 \to b_1)} (p_1, p'_1) \xrightarrow{(a_2 \to b_2)} \cdots \xrightarrow{(a_k \to b_k)} (p_k, p'_k)$. By the definition of morphism h, it is immediate that there is an alignment $w = (a_1 \to b_1)(a_2 \to b_2) \cdots (a_k \to b_k)$ in $\mathcal{A}(M, M_\theta)$ such that

$$h(w) = (x, y), \quad \text{where } a_i \in \Sigma, b_i \in \Sigma \cup \{\lambda\} \text{ for } 1 \le i \le k.$$

Since y is a subsequence of x and $|y| < |x|$, it is immediate from the construction of $\mathcal{A}(M, M_\theta)$ that at least one edit operation in w should be a deletion operation—a contradiction. □

Hence, we can determine whether or not a given language $L(M)$ is a θ-hypercode using an alignment FA in $O(|Q||Q'| + |\delta||\delta'|)$ worst-case time.

We observe that the alignment FA $\mathcal{A}(M, M_\theta)$ accepts all edit strings w such that

1. $h(w) = (x, y)$,
2. $x \in L(M)$ and $y \in L(M_\theta)$, and
3. x is a supersequence of y.

We can obtain the following two sets from $L(\mathcal{A}(M, M_\theta))$.

$$\mathbb{H}_L = \{x \mid h(w) = (x, y) \text{ such that } x \ne y \text{ for } w \in L(\mathcal{A}(M, M_\theta))\},$$
$$\mathbb{T}_L = \{y \mid h(w) = (x, y) \text{ such that } x \ne y \text{ for } w \in L(\mathcal{A}(M, M_\theta))\}.$$

In other words, $\mathbb{H}_L$ contains all strings that are supersequence of a string from $L(M_\theta)$ and $\mathbb{T}_L$ contains all strings that are subsequence of a string from $L(M)$. Then, from these two sets, we can generate θ-hypercodes as follows:

Corollary 3.8. *Given an FA M and an involution θ,*

1. $\mathbb{H}_L = \{x \in L(M) \mid x \text{ is a supersequence of } y \in L(M_\theta)\}$ *and* $L(M) \setminus \mathbb{H}_L$ *is a θ-hypercode.*
2. $\mathbb{T}_L = \{y \in L(M_\theta) \mid y \text{ is a subsequence of } x \in L(M)\}$ *and* $L(M_\theta) \setminus \mathbb{T}_L$ *is a θ-hypercode.*

3.3. Involution context-free hypercodes

In Sections 3.1 and 3.2, we proposed decision algorithms for hypercodes or involution regular hypercodes based on the intersection emptiness of two FAs and the edit-distance. Besides the regular language family, we design an algorithm for involution context-free hypercodes: namely, the input language is now context-free. Since the code decision problem (such as prefix, suffix, infix, k-intercodes) for a context-free language is often undecidable [21], we first tackle the decision problems for θ-prefix, θ-suffix, θ-infix, θ-overlap-free, θ-k-intercode, and θ-solid codes, which turn out to be undecidable as well.

Theorem 3.9. *There is no algorithm that determines whether or not a given linear language L is θ-prefix, θ-suffix, θ-infix, θ-overlap-free, a θ-k-intercode, or a θ-solid code.*

Proof. We only prove the θ-k-intercode case for $k = 1$ (θ-comma-free code). The other proofs are similar. Note that the following proof can be easily generalized for $k \ge 1$.

It is already known that the problem of determining whether or not a given linear language L is an intercode of index m is undecidable [22]. We use a similar proof for an involution linear language and establish the undecidability result for a θ-hypercode. Let Σ be an alphabet and (U, V) be an instance of Post's Correspondence Problem [29], where $U = (u_0, u_1, \ldots, u_{n-1})$ and $V = (v_0, v_1, \ldots, v_{n-1})$. Assume that the symbols $0, 1, \#, \$, \phi, \%$ are not in Σ. Let $\Sigma' = \Sigma \cup \{0, 1, \#, \$, \phi, \%\}$. For any nonnegative integer i, let $\beta(i)$ be the shortest binary representation of i. Let $\theta(0) = 0$, $\theta(1) = 1$, $\theta(\#) = \#$, $\theta(\$) = \$$, $\theta(\phi) = \phi$, and $\theta(\%) = \%$.

Consider a linear grammar $G = (N, \Sigma', P, S)$, where

- $N = \{S, T_U, T_V\}$ is a nonterminal alphabet,
- Σ' is a terminal alphabet,
- S is the sentence symbol, and
- P has the following rules:
  - $S \to \%\beta_i\phi T_U u_i\# \mid \%\beta_i\$u_i\# \mid \%\beta_i\phi T_U u_i\#\# \mid \#\theta(v_i)T_V\phi\theta(\beta_i)\% \mid \%\beta_i\$u_i\#\# \mid \#\theta(v_i)\$\theta(\beta_i)\% \mid \#\#\theta(v_i)T_V\phi\theta(\beta_i)\% \mid \#\#\theta(v_i)\$\theta(\beta_i)\%$,
  - $T_U \to \beta_i\phi T_U u_i \mid \beta_i\$u_i$, and
  - $T_V \to \theta(v_i)T_V\phi\theta(\beta_i) \mid \theta(v_i)\$\theta(\beta_i)$

  for $i \in \{0, 1, \ldots, n-1\}$.

Then, $L(G)$ consists of the following four types (T1–T4) of strings:

T1. $\%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$u_{i_0}\cdots u_{i_{n-1}}\#$,
T2. $\%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$u_{i_0}\cdots u_{i_{n-1}}\#\#$,
T3. $\#\theta(v_{i_{n-1}})\cdots\theta(v_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%$, and
T4. $\#\#\theta(v_{i_{n-1}})\cdots\theta(v_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%$

for all $m \in \mathbb{N}$ and $i_j \in \{0, 1, \ldots, n-1\}$ for $0 \le j \le m-1$. Then, once we apply the involution to $L(G)$, the involution language $\theta(L(G))$ becomes a set of the following four types (T5–T8) of strings:

T5. $\#\theta(u_{i_{n-1}})\cdots\theta(u_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%$,
T6. $\#\#\theta(u_{i_{n-1}})\cdots\theta(u_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%$,
T7. $\%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$v_{i_0}\cdots v_{i_{n-1}}\#$, and
T8. $\%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$v_{i_0}\cdots v_{i_{n-1}}\#\#$.

Given a context-free grammar G and an involution θ, $L(G)$ and $\theta(L(G))$ have strings that start with # and %. Note that $L(G)$ is θ-comma-free if there exist two strings w_1 and w_2 in $L(G)$ such that $w_1w_2 = v_1wv_2$, where $v_1, v_2 \in \Sigma^+$ and $w \in \theta(L(G))$. Since w_1, w_2, and w start with % and #, w cannot appear in the middle of w_1 and w_2. Then, there is a string $w_1w_2 = v_1wv_2$ if PCP has a solution.

We now show that $L(G)$ is not θ-comma-free if and only if PCP has a solution.

($\Longleftarrow$) It is easy to verify that if PCP has a solution, then L is not θ-comma-free since a string $w = \%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$u_{i_0}\cdots u_{i_{n-1}}\#$ in $L(G)$ is a prefix of a string $x = \%\beta_{i_{n-1}}\phi\cdots\phi\beta_{i_0}\$v_{i_0}\cdots v_{i_{n-1}}\#\#$ in $\theta(L(G))$. Note that if $L(G)$ is θ-comma-free, then $L(G)$ is also θ-prefix.

($\Longrightarrow$) If $L(G)$ is not θ-comma-free, it implies that

$$v_1wv_2 = w_1w_2,$$

where $w_1, w_2 \in L(G)$, $w \in \theta(L(G))$, and $v_1, v_2 \in \{U \cup V \cup \Sigma'\}^+$. Note that w_1, w_2, and w start with only % or #, and both % and # do not appear in the middle of w_1, w_2, and w. Thus, w is either a prefix of w_1w_2 or a suffix of w_1w_2. This follows that w_1 is in T4 or w_2 is in T2 since v_1 and v_2 are not λ. Similarly, w should be in T5 or T7. Thus, there are six combinations for w_1, w_2, and w as follows:

1. $w_1 \in$ T4, $w_2 \notin$ T2, and $w \in$ T5.
2. $w_1 \in$ T4, $w_2 \notin$ T2, and $w \in$ T7 (impossible).
3. $w_1 \notin$ T4, $w_2 \in$ T2, and $w \in$ T5 (impossible).
4. $w_1 \notin$ T4, $w_2 \in$ T2, and $w \in$ T7.
5. $w_1 \in$ T4, $w_2 \in$ T2, and $w \in$ T5.
6. $w_1 \in$ T4, $w_2 \in$ T2, and $w \in$ T7.

We only prove the first case. The fifth case is similar to the first case, and the fourth and the sixth cases are symmetric to the first and the fifth cases. In the first case, both w_1w_2 and v_1wv_2 are as follows:

$$w_1w_2 = \#\#\theta(v_{i_{n-1}})\cdots\theta(v_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%w_2,$$
$$v_1wv_2 = v_1\#\theta(u_{i_{n-1}})\cdots\theta(u_{i_0})\$\theta(\beta_{i_0})\phi\cdots\phi\theta(\beta_{i_{n-1}})\%v_2.$$

Since we assume that $w_1w_2 = v_1wv_2$ as depicted in Fig. 3, and $v_1 = \#$, the string w is the same as the suffix of length $|w_1| - 1$ of w_1, and $v_2 = w_2$. This follows that there is a PCP solution. Hence, it is undecidable whether or not $L(G)$ is a θ-comma-free code since PCP is undecidable. □

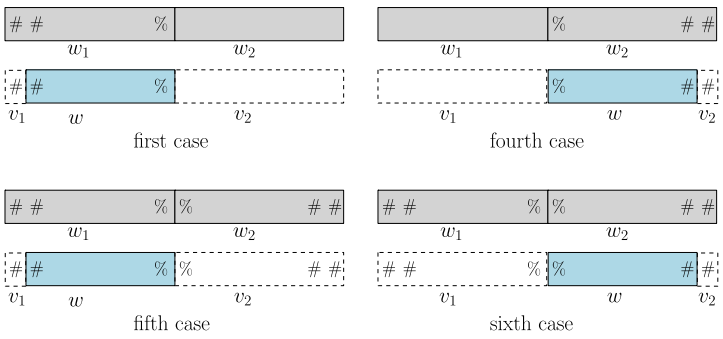

Fig. 3. An example of possible combinations for w_1w_2 and v_1wv_2.

We next examine the decidability of θ-hypercodes for context-free languages. We use an edit-distance approach to solve the θ-hypercode decision problem. Recall that it is already proved that computing the minimum edit-distance between two context-free languages is impossible [27]. Recently, two of the authors showed how to compute the edit-distance between a regular language and a context-free language [13], and Han et al. [14] designed efficient algorithms for computing the edit-distance between a context-free grammar and an FA. Since we cannot compute the edit-distance between $L(G)$ and $\theta(L(G))$ directly, we instead obtain a finite subset of $L(G)$ that is sufficient to solve the problem.

Definition 3.10. Given a CFG $G = (V, T, S, P)$, we define

$$H\big(L(G)\big) = \big\{w \in L(G) \;\big|\; \text{there is no proper subsequence of } w \text{ in } L(G)\big\}.$$

For instance, when $L = \{invent, topic, toc, int\}$, $H(L(G)) = \{toc, int\}$.

Definition 3.11. Given a CFG $G = (V, T, S, P)$, let $\mathcal{T}_w$ be a parse tree for w in G. We define $\mathbb{S}(L(G)) = \{w \in L(G) \mid$ for each path from S to a leaf in $\mathcal{T}_w$, all variables (internal nodes) are distinct$\}$.

Lemma 3.12. *$\mathbb{S}(L(G))$ is finite.*

Proof. Given a CFG $G = (V, T, S, P)$, let m be the number of variables and k be the length of the longest production in P. Then, the length of the longest string w in $\mathbb{S}(L(G))$ is at most k^m since each path from S to a leaf in $\mathcal{T}_w$ for w cannot use the same variable twice. If $\mathbb{S}(L(G))$ is infinite, then we should be able to pick a string $w' \in L(G)$ of length greater than k^m. However, it is impossible to yield a string of length k^m with the parse tree whose height is less than or equal to $m-1$. Thus, $\mathbb{S}(L(G))$ is finite. □

Note that since $\mathbb{S}(L(G))$ is finite, we can obtain $\mathbb{S}(L(G))$ from a CFG G by enumerating all trees that satisfy the condition in Definition 3.11.

Lemma 3.13. $H(L(G)) \subseteq \mathbb{S}(L(G))$.

Proof. We prove the statement by showing that if a string $w \notin \mathbb{S}(L(G))$, then $w \notin H(L(G))$. Let $w \in L(G) \setminus \mathbb{S}(L(G))$. Then, given a CFG $G = (V, T, S, P)$, there exists a derivation for w as follows: $S \Rightarrow xAy \overset{*}{\Longrightarrow} xuAvy \overset{*}{\Longrightarrow} xu\beta vy \overset{*}{\Longrightarrow} w$. From the derivation, we can find another derivation, $S \Rightarrow xAy \overset{*}{\Longrightarrow} x\beta y \overset{*}{\Longrightarrow} w' \in L(G)$. Thus, $w \notin H(L(G))$ since w' is a subsequence of w. □

Lemma 3.14. *For any string $w \in \mathbb{S}(L(G)) \setminus H(L(G))$, w is a supersequence of a string in $H(L(G))$.*

Recall that our problem is to determine whether or not there exists a pair of strings $x \in L(G)$ and $y \in \theta(L(G))$ such that x is a proper subsequence of y. Now we know that it is sufficient to check whether or not there exists such a pair of strings $x \in \mathbb{S}(L(G))$ and $y \in \theta(L(G))$ based on Lemmas 3.12, 3.13, and 3.14.

Han et al. [13] introduced an alignment PDA that accepts all possible alignments between all pairs of strings; one is from a context-free language and the other is from a regular language. We construct an alignment PDA between a context-free language $\theta(L(G))$ and a finite language $\mathbb{S}(L(G))$. First, we construct an FA $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$ accepting $L(M) = \mathbb{S}(L(G))$. Then, based on a PDA $N = (Q_N, \Sigma, \Gamma, \delta_N, s_N, Z_0, F_N)$ for $\theta(L(G))$ and the new FA $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$,

we compute an alignment PDA G_P between $\theta(L(G))$ and $\mathbb{S}(L(G))$ that allows only deletions and identical substitutions as follows:

$$G_P = (Q_P, \Omega, \delta_P, s_P, F_P),$$

where

- $Q_P = Q_N \times Q_M$ is a set of states,
- $\Omega = \{(a \to a) \mid a \in \Sigma\} \cup \{(a \to \lambda) \mid a \in \Sigma\}$ is an alphabet of edit operations,
- $\delta_P((i_u, j_u), (a \to a), \gamma) = \{((i_{u+1}, j_{u+1}), \varphi) \mid (i_{u+1}, \varphi) \in \delta_N(i_u, a, \gamma), j_{u+1} \in \delta_M(j_u, a), a \in \Sigma, \gamma \in \Gamma, \varphi \in \Gamma^*\}$ is a transition function for identical substitutions,
- $\delta_P((i_u, j_u), (a \to \lambda), \gamma) = \{((i_{u+1}, j_u), \varphi) \mid (i_{u+1}, \varphi) \in \delta_N(i_u, a, \gamma), a \in \Sigma, \gamma \in \Gamma, \varphi \in \Gamma^*\}$ is a transition function for deletions,
- $s_P = (s_N, s_M)$ is the start state,
- $F_P = F_N \times F_M$ is a set of final states.

Namely, G_P simulates a PDA N for $\theta(L(G))$ and an FA M for $\mathbb{S}(L(G))$ simultaneously. For identical substitutions, G_P simulates both by reading the same character on M and N. For deletions, G_P simulates N by reading an input character $a \in \Sigma$ while M reads λ, which is an empty character.

From the alignment PDA, we establish the following statement.

Lemma 3.15. *The alignment PDA G_P accepts an alignment $w \in \Omega^*$ if and only if $h(w) = (x, y)$, where $x \in \mathbb{S}(L(G))$ is a subsequence of $y \in \theta(L(G))$.*

Lemma 3.15 guarantees that $L(G)$ is not a θ-hypercode if and only if G_P accepts an alignment containing at least one deletion operation. Thus, we only need to verify whether or not there exists such an accepting path in G_P. Let $\mathcal{H} : \Omega \to \Omega \cup \{\lambda\}$ be a morphism defined as follows:

$$\mathcal{H}(a \to b) = \begin{cases} \lambda & \text{if } a = b, \\ (a \to b) & \text{otherwise.} \end{cases}$$

Let $\mathcal{H}(L(G_P))$ denote the morphism of $L(G_P)$ by $\mathcal{H}$. If $\mathcal{H}(L(G_P)) \setminus \{\lambda\}$ is not empty, then there exists an alignment with a deletion operation in $L(G_P)$ and, thus, $L(G)$ is not a θ-hypercode. Now, we can decide whether or not L is a θ-hypercode by the emptiness test of $\mathcal{H}(L(G_P)) \setminus \{\lambda\}$. It is clear that $L(G_P)$ is context-free and $\mathcal{H}(L(G_P))$ is also context-free. Since context-free languages are closed under set difference with regular languages and the emptiness of a context-free language is decidable [16,33], we can determine whether or not $\mathcal{H}(L(G_P)) \setminus \{\lambda\}$ is empty.

Theorem 3.16. *It is decidable whether or not a given context-free language L is a θ-hypercode.*

4. Conclusions

We have considered the hypercode decision problem when the language L is regular. We have proved that it is decidable to determine whether or not a given regular language is a hypercode in polynomial time. We also have defined an involution regular hypercode, which may be infinite whereas a hypercode is always finite [31] and presented two decision algorithms for involution regular hypercodes. Lastly, we have demonstrated that it is decidable for an involution context-free hypercode using a modified edit-distance computation. We have examined some other involution context-free codes and proved their undecidability. An efficient algorithm that decides θ-hypercodes for context-free languages remains to be found.

Acknowledgements

This research was supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).

We wish to thank the referees for the care they put into reading the previous version of this manuscript. Their comments including the references on Higman–Haines sets and an FA construction recognizing all proper subsequences of a given language were invaluable in depth and detail. The current version owes much to their efforts.

References

[1] J. Berstel, D. Perrin, C. Reutenauer, Codes and Automata, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, 2009.
[2] H. Bunke, Edit distance of regular languages, in: Proceedings of the 5th Annual Symposium on Document Analysis and Information Retrieval, 1996, pp. 113–124.
[3] C. Choffrut, G. Pighizzini, Distances between languages and reflexivity of relations, Theoret. Comput. Sci. 286 (1) (2002) 117–138.
[4] R. Deaton, M. Garzon, R.C. Murphy, J.A. Rose, D.R. Franceschetti, S.E. Stevens Jr., Genetic search of reliable encodings for DNA-based computation, in: Late Breaking Papers at the Genetic Programming, 1996 Conference Stanford University, July 28–31, 1996, 1996, pp. 9–15.

[5] M. Domaratzki, Trajectory-based codes, Acta Inform. 40 (6) (2004) 491–527.
[6] M. Garzon, P. Neathery, R. Deaton, R.C. Murphy, D.R. Franceschetti, S.E. Stevens Jr., A new metric for DNA computing, in: Genetic Programming 1997: Proceedings of the Second Annual Conference, 1997, pp. 472–478.
[7] M. Garzon, R. Deaton, L.F. Nino, E. Stevens, M. Wittner, Encoding genomes for DNA computing, in: Genetic Programming 1998: Proceedings of the Third Annual Conference, 1998, pp. 684–690.
[8] H. Gruber, M. Holzer, M. Kutrib, The size of Higman–Haines sets, Theoret. Comput. Sci. 387 (2) (2007) 167–176.
[9] H. Gruber, M. Holzer, M. Kutrib, More on the size of Higman–Haines sets: effective constructions, Fund. Inform. 91 (1) (2009) 105–121.
[10] L.H. Haines, On free monoids partially ordered by embedding, J. Combin. Theory 6 (1) (1969) 94–98.
[11] Y.-S. Han, Y. Wang, D. Wood, Infix-free regular expressions and languages, Internat. J. Found. Comput. Sci. 17 (2) (2006) 379–393.
[12] Y.-S. Han, Y. Wang, D. Wood, Prefix-free regular languages and pattern matching, Theoret. Comput. Sci. 389 (1–2) (2007) 307–317.
[13] Y.-S. Han, S.-K. Ko, K. Salomaa, The edit-distance between a regular language and a context-free language, Internat. J. Found. Comput. Sci. 24 (7) (2013) 1067–1082.
[14] Y.-S. Han, S.-K. Ko, K. Salomaa, Approximate matching between a context-free grammar and a finite-state automaton, in: Proceedings of the 18th International Conference on Implementation and Application of Automata, 2013, pp. 146–157.
[15] T. Harju, A. Mateescu, A. Salomaa, Shuffle on trajectories: the Schützenberger product and related operations, in: Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science, 1998, pp. 503–511.
[16] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, 2 edition, Addison-Wesley, Reading, MA, 1979.
[17] S. Hussini, L. Kari, S. Konstantinidis, Coding properties of DNA languages, Theoret. Comput. Sci. 290 (3) (2003) 1557–1579.
[18] L. Ilie, Decision problems on orders of words, PhD thesis, University of Turku, 1998.
[19] N. Jonoska, K. Mahalingam, J. Chen, Involution codes: with application to DNA coded languages, Nat. Comput. 4 (2) (2005) 141–162.
[20] N. Jonoska, L. Kari, K. Mahalingam, Involution solid and join codes, Fund. Inform. 86 (1–2) (2008) 127–142.
[21] H. Jürgensen, S. Konstantinidis, Codes, in: Word, Language, Grammar, in: Handbook of Formal Languages, vol. 1, 1997, pp. 511–607.
[22] H. Jürgensen, K. Salomaa, S. Yu, Decidability of the intercode property, Elektron. Inf.verarb. Kybern. (1993) 375–380.
[23] L. Kari, S. Konstantinidis, Descriptional complexity of error/edit systems, J. Autom. Lang. Comb. 9 (2004) 293–309.
[24] L. Kari, K. Mahalingam, DNA codes and their properties, in: Proceedings of the 12th International Meeting on DNA Computing, 2006, pp. 127–142.
[25] L. Kari, E. Losseva, S. Konstantinidis, P. Sosík, G. Thierrin, A formal language analysis of DNA hairpin structures, Fund. Inform. 71 (4) (2006) 453–475.
[26] D. Kephart, J. Lefevre, CODEGEN: the generation and testing of DNA code words, in: Proceedings of IEEE Congress on Evolutionary Computation, 2004, pp. 1865–1873.
[27] M. Mohri, Edit-distance of weighted automata, in: Proceedings of the 7th International Conference on Implementation and Application of Automata, 2003, pp. 1–23.
[28] A. Okhotin, On the state complexity of scattered substrings and superstrings, Fund. Inform. 99 (3) (2010) 325–338.
[29] E.L. Post, A variant of a recursively unsolvable problem, Bull. Amer. Math. Soc. (N.S.) 52 (4) (1946) 264–268.
[30] P. Raven, G. Johnson, K. Mason, J. Losos, S. Singer, Biology, McGraw-Hill Higher Education, 2007.
[31] H.J. Shyr, G. Thierrin, Hypercodes, Inf. Control 24 (1974) 45–54.
[32] J. van Leeuwen, Effective constructions in well-partially-ordered free monoids, Discrete Math. 21 (3) (1978) 237–252.
[33] D. Wood, Theory of Computation, John Wiley & Sons, Inc., New York, NY, 1987.