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/

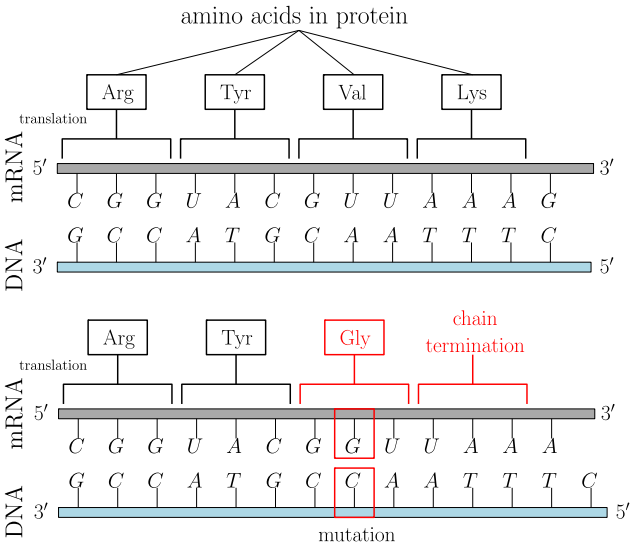

Fig. 1. An example of frameshift mutation.

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