Theoretical Computer Science 608 (2015) 84–97

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Testing DNA code words properties of regular languages ☆

Rocco Zaccagnino [a], Rosalba Zizza [a,*], Carlo Zottoli

[a] *Dipartimento di Informatica, Università degli Studi di Salerno, via Giovanni Paolo II 132, 84084 Fisciano (SA), Italy*

ARTICLE INFO

Article history:
Received 30 January 2015
Received in revised form 10 August 2015
Accepted 26 August 2015
Available online 2 September 2015

Keywords:
DNA computing
Formal languages
Regular languages
Automata theory

ABSTRACT

One aspect of DNA Computing is the possibility of using DNA molecules for solving some "complicated" computational problems. In this context, the *DNA code word design problem* assumes a fundamental role: given a problem encoded in DNA strands and biochemical processes, the final computation is a concatenation of the input DNA strands that must allow us to recover the solution of the given problem in terms of the input (unique decipherability). Thus the initial set of DNA strands must be a code. In addition, it should satisfy some restrictions, called here DNA properties, in order to prevent them from interacting in undesirable ways. So a new interest towards the design of efficient algorithms for testing whether a language X is a code, has arisen from (wet) DNA Computing, but, as far as we know, only when X is a finite set. In this paper we provide an algorithm for testing whether an infinite but regular set of words is a code that avoids some DNA properties among unwanted intermolecular and intramolecular hybridizations.

1. Introduction

Variable-length codes were investigated in depth for the first time by Schützenberger (1955), by linking the theory of codes with classical noncommutative algebra. Informally, given a finite alphabet Σ, a code is a set of words X on Σ such that any message composed from its elements is uniquely decipherable (UD) and so X guarantees a secure and economic transmission. Sardinas and Patterson [26] characterized codes through a decidable property for a regular set. When X is a finite set, efficient algorithms have been designed for testing the UD property, by using different approaches (see [4]). Others are known in the regular case [3,4,12,22].

Recently a new interest with respect to the design and implementation of efficient algorithms for testing the UD property has arisen from (wet) DNA Computing [1,24]. Briefly, every DNA molecule can be viewed as an oriented single-stranded sequence of nucleotides, i.e., a word over the finite alphabet $\Sigma = \{A, C, G, T\}$. The simple property of Watson–Crick complementarity (C and T are respectively complementary to G and A, and vice versa), which allows the precise matching between two oppositely-oriented complementary strands, can be used to assemble DNA strands (hybridization). One aspect of DNA Computing is the possibility of using DNA molecules for solving some "complicated" computational problems (like showed by Adleman in his seminal paper [1]). Briefly, in [1] a small instance of the Hamiltonian Path Problem, a well-known NP-complete problem, has been solved by encoding vertices and edges of the graph into a set of suitable molecules. So the

☆ Partially supported by the *FARB* Project "*Aspetti algebrici e computazionali nella teoria dei codici, degli automi e dei linguaggi formali*" (University of Salerno, 2013), the *FARB* Project "*Codici e sistemi di splicing nella teoria dei linguaggi formali*" (University of Salerno, 2014) and the *MIUR PRIN* 2010–2011 grant "*Automata and Formal Languages: Mathematical and Applicative Aspects*", code H41J12000190001.

* Corresponding author.
E-mail addresses: zaccagnino@dia.unisa.it (R. Zaccagnino), zizza@dia.unisa.it (R. Zizza).

http://dx.doi.org/10.1016/j.tcs.2015.08.034
0304-3975/

solution, if exists, is nothing but a concatenation of initial molecules, and it has been obtained by biochemical processes, such as Gel Electrophoresis, Length Separation, Polymerase Chain Reaction (PCR). Thus, the input data in DNA Computing must be encoded into the form of single or double DNA strands. Due to the importance of this step for the success of computations, much attention has been devoted to define suitable sets of code words.

Two different approaches have been adopted in the literature to investigate "good" DNA encodings. In [9], solutions based on biochemical conditions have been proposed (see also [21] and the bibliography of [9]). Here we follow the theoretical approach of [14–18], where the problem is faced from a formal language point of view, by defining a list of properties of words and languages, which if fulfilled by a given set of DNA strands, can guarantee the robustness during computation. In order to be used for the encoding of the problem, the first property that a set of words should have is the UD property. Otherwise, starting from the final computation, when it is a concatenation of the input DNA strands, we are not able to uniquely recover the solution of the problem. In addition, as complementary parts of single strands can bind together forming a double stranded DNA sequence, we have to impose some restrictions (in the sequel called *DNA properties*) on these sets of code words languages to prevent them from interacting in undesirable ways. Two classes of hybridizations may be considered: *intermolecular* hybridizations, i.e., two different single stranded bind together, forming a double-stranded DNA molecule; *intramolecular* hybridization, i.e., a piece of a long single stranded molecule hybridizes with another piece of the same molecule. As an example, *hairpins* occur when a factor of a long single stranded DNA molecule is complementary to another factor of the same length. In both situations these hybridizations are unwanted for DNA Computing, since the obtained molecules cannot be used for encoding information. (Another aspect, which will not be considered here, is that molecular processes are not error-free: two DNA sequences which are not perfectly Watson–Crick complementary may hybridize, producing false positives or negatives.) Among them, θ-*compliance*, θ-*freedom* and θ-k-*hairpin-freeness* have been defined, where θ is a morphism or antimorphism on the alphabet and k is a positive integer. The first two properties avoid two intermolecular hybridizations for which the Watson–Crick complement of a word v in X cannot be a factor of another word of X (resp. the concatenation of two words of X). It is already known that the problem of deciding whether a regular language X is θ-compliant or θ-free is decidable and can be solved in quadratic time with respect to the size of a finite state automaton recognizing X [14]. The θ-k-hairpin-freeness property guarantees that no word of the language contains both a factor v and its involution $\theta(v)$. Obviously, the length of v is a parameter with respect to considering the property (otherwise, the property could be trivial, e.g., when $k = 1$). Thus, more precisely, a language L is θ-k-hairpin-free if there exists no word v such that v and $\theta(v)$ are factors of the same word in L, with $|v| \geq k \geq 1$ and θ is an involution. In this way, we require that "long" factors do not hybridize and we decide how long they are by fixing k. It is known that it is possible to decide whether a regular language is θ-k-hairpin-free, for a given involution θ and an integer k.

For the sake of completeness, we must say that in the literature hairpins are not everywhere considered "bad". As an example, in [25] hairpins are used for performing computation and so it is crucial to design algorithms for generating words with hairpins, as well as the hairpin completion operation [5,20]. Our paper does not investigate hairpins from this "positive" point of view.

As far as we know, all known algorithms and implementations for testing intermolecular properties have been designed when X is a finite language [19,23]. It is evident that, for practical usage, only finite sets should be considered. Nevertheless, larger and larger instances of NP-complete problems are coded through a (big) set of different words satisfying DNA properties. In addition, consider a finite set X which has been verified to be "good" for DNA computing. If we need to increase the instance of only one word w, and so we examine $X \cup \{w\}$, we must check again the whole set $X \cup \{w\}$ w.r.t. the DNA properties and the UD property. On the contrary, it could be more useful to test DNA properties of an infinite set of DNA words, which have the same structural description, provided by a regular expression. In this way, we have a regular infinite set X which is a code satisfying some DNA properties, we can choose any subset Y of X to encode our problem, and we can add any word from X to Y, maintaining the goodness of our set. Furthermore, the set X is not a list of many different words, but it is described by a succinct regular expression. This is our motivation for effectively testing DNA properties on regular infinite sets.

In this perspective, we firstly consider the UD property, a necessary property for good encodings. It cannot be efficiently tested for a regular set, if X is given by an ambiguous automaton (otherwise, if X is given by an unambiguous automaton, efficient techniques are known [4]). In [6], the authors used the algorithm of [22] for testing UD property, that presents many advantages w.r.t. others (see Section 3 for details). In addition, McCloskey's technique has been also used in the same paper for testing the two intermolecular properties above mentioned. The time complexity of the algorithm is $O(n^2)$, where n is the size of the finite state automaton recognizing X. Following the approach of CodeGen [19], in [6] a Java implementation is also provided, which takes care of space requirements for implementing automata. The software implements automata by using primitive data types and no external library.

This paper extends results presented in [6], where only intermolecular hybridizations were considered. Here we also present an algorithm and its implementation for testing θ-k-hairpin-freeness property for regular languages, based on common characteristics of algorithms for intermolecular properties developed in [6]. The implementation of [6] has been revised for guarantee a unified approach to the detection of the whole properties considered in this paper.

The paper is organized as follows. In Section 2 we gathered basics on words, automata and codes. Sections 3–6 review the results of [6] by fixing definitions and providing lacking proofs. Precisely, Section 3 presents the problem of testing the UD property and the algorithm of [22] is detailed in Section 4. DNA properties are defined in Section 5 and Section 6 shows how we can test them on a regular set. Section 7 regards detection of hairpin-free languages. Conclusions give a glance