Proceedings of the 25th Annual International Conference of the IEEE EMBS
Cancun, Mexico • September 17-21, 2003

Static and Dynamic Properties of DNA Languages

L. Kari[1], S. Konstantinidis[2]
[1]Department of Computer Science, University of Western Ontario, London, Ontario, N6A 5B7, Canada
[2]Department of Math. & Computing Science, Saint Mary's University, Halifax, Nova Scotia, B3H 3C3, Canada

Abstract—**We provide an overview of some current developments on code-related properties of DNA languages. A DNA language is a set of words, each of which is made up of the letters A, C, G, T. Such a word is meant to represent a physical DNA strand. A collection of DNA strands can be stored in-vitro and, either serve the purpose of a database, or undergo a sequence of controlled bio-operations that would constitute a meaningful computation. In both cases, the strands should be chosen in such a way that they would not form unwanted hybridizations with each other and any errors in the nucleotides comprising the strands can be detected. These requirements can be translated in the framework of formal language theory by considering DNA languages whose words satisfy certain combinatorial properties. We consider two types of desirable properties: static and dynamic. The former ensure that no unwanted hybridizations can occur. The latter ensure that, after a permitted bio-operation is applied to the strands, the resulting strands also satisfy the desirable properties.**

Keywords—**DNA codes, DNA computing, formal languages**

I. Introduction

The possibility of DNA computing is based on the fact that information can be encoded using words over the four-letter alphabet {A, C, G, T}, which can then be represented physically by DNA strands. Moreover, these strands can be processed using certain bio-molecular techniques, which we call *bio-operations*, such as hybridization, denaturation, separation by length, cutting and pasting at desired locations, etc. In most proposed DNA-based algorithms, the initial DNA solution encoding the input to the problem will contain some DNA strands which represent single *codewords*, and some which represent strings of concatenated codewords.

Several attempts have been made to address the issue of "good encodings" by trying to find sets of codewords which are unlikely to form undesired bonds with each other by hybridization [3], [7]. For example genetic and evolutionary algorithms have been developed which select for sets of DNA sequences that are less likely to form undesirable bonds [4], [5]. [6] has developed a program to create DNA sequences to meet logical and physical parameters such as uniqueness, melting temperatures and G/C ratio as required by the user. [10] has designed a software for constraint-based nucleotide selection. [8] has investigated encodings for DNA computing in virtual test tubes. [19] used combinatorial methods to calculate bounds on the size of a set of fixed-length codewords (as a function of codeword length) which are less likely to mis-hybridize.

In this overview we present some of the main ideas and results from [15], [13], and [16], in which certain requirements of avoiding unwanted hybridizations and detecting random nucleotide errors in DNA strands are formalized as language properties. We define the *DNA involution* t to be the mapping that evaluates the Watson-Crick complement of a DNA strand as follows:

- If $w = B_1B_2...B_n$ is a DNA word, with each B_i being a letter in {A, C, G, T}, then t(w) is the word $t(B_n)...t(B_2)t(B_1)$. Moreover we have that t(A) = T, t(T) = A, t(C) = G, t(G) = C. For example, t(AAGCTC) = GAGCTT.

By convention, a word of the form $B_1B_2...B_n$ will represent the corresponding single DNA strand in the 5' to 3' orientation: $5' - B_1B_2...B_n - 3'$.

When a collection of DNA strands is stored in vitro, certain hybridizations can be formed between strands due to the Watson-Crick complementarity property of nucleotides. Figures 1 – 3 show three of the many possible ways that this could happen. From the point of view of DNA computing, such formations are normally undesirable because the data involved in them cannot be processed. In the next section we address this problem from the point of view of formal languages.

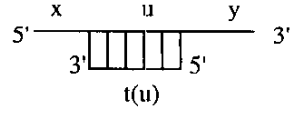

Fig. 1. The strand t(u) sticks to the strand xuy

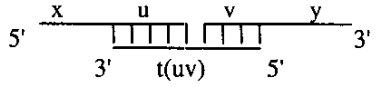

Fig. 2. The strand t(uv) sticks to the concatenation of the strands xu and vy

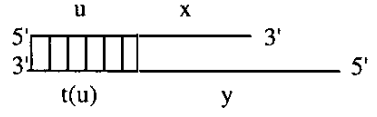

Fig. 3. The strand yt(u) sticks to the strand ux

II. STICKY-FREE LANGUAGES

A language is any set of words. In this section we consider languages L with the following properties.

- *strictly t-compliant* [15]: when no two words in L are of the form xuy and t(u). With this property, no words in L can form the structure shown in Fig. 1.
- *strictly t-free* [13]: when no three words in L are of the form xu, vy, t(uv). With this property, no words in L can form the structure shown in Fig. 2.
- *strictly t-sticky-free* [16]: when no two words in L are of the form ux, yt(u). With this property, no words in L can form the structure shown in Fig. 3.

For example, the language X = {ACTA, ATAA, ATTA} is strictly t-compliant but not strictly t-free because ATTA is equal to t(TAAT) and ACTA ends with TA and ATAA starts with AT. On the other hand, the language Y = {AATCC, AATGTCC, AATTTCC} satisfies all the preceding properties. Constructing languages with short words that satisfy these properties is not a difficult task. If longer words are needed, however, a computer search might be intractable (long DNA words might be needed in certain DNA computations such as in Adleman's experiment [1]). The following results provide a mathematical method of generating DNA languages with arbitrarily long and many words.

- Every strictly t-free language is also strictly t-compliant [13].
- If K is strictly t-free then also K+ is strictly t-free [13].
- If K is strictly t-sticky-free then also K+ is strictly t-sticky-free [16].

The language K+ obtains by concatenating one or more words of K; therefore, K+ is an infinite language. For example, using the language Y, we can generate the language Y+ that satisfies the three properties and includes the words of Y and concatenations of these words such as AATCCAATCC, AATGTCCAATCC, etc.

This was a particular method of obtaining large DNA languages from simpler ones. The reader is referred to [13] and [16] for other methods as well as for additional properties of DNA languages.

III. OPERATION-INVARIANT LANGUAGES

Several theoretical models of DNA computation have been proposed, most of which involve the concept of a multi-set of words [14], [20], [11]. Informally, a multi-set of words M, say, is a collection of words such that a word might occur in M more than once. A *word operation*, say f, is a function that when applied to M it alters some of the words in M, resulting thus in a new multi-set N. A multi-set represents a test tube containing DNA strands and the operation represents a physical bio-operation that is applied to some of these strands – this could be, for instance, the action of a certain restriction enzyme. Consider for example, the splicing operation f specified by the expression (splicing rule)

(ACC#GG, TT#AAA)

If M contains two strands of the form xACCGGy and uTTAAAv, and f is applied on M then these strands would be replaced with the strands xACCAAAv and uTTGGy. In general, for a fixed but arbitrary set of operations F, the notation M=>N represents the fact that the multi-set N results from M by performing some operation f in F. Similarly, the notation M=>+N represents the fact that N results from M by performing a sequence of one or more operations in F. A *multi-set system* is a triple SYS = (Σ, A, F), where Σ is the word alphabet, A is the initial multi-set of words, and F is the set of permitted operations. The *computation language* of SYS is the set of words that appear in the steps of all possible sequences of operations that start from the initial multi-set A.

The computation language, say L, of SYS should satisfy properties such as the ones defined in the previous section, as this would ensure that the undesirable hybridizations shown in Fig. 1—3 will not occur during any computation of SYS. This means that the language L would be F-*invariant*: if any operation of F is applied to some words of L then the resulting words will also be in L. In [16] we provide polynomial-time algorithms for testing whether a given regular language is invariant for a given set of operations F. Moreover, we discuss a method of choosing the initial multi-set A and the set of operations F such that the computation language of SYS is a subset of an F-invariant language that satisfies the three desirable properties. The method requires choosing *a comma-free code* K that is strictly t-free and strictly t-sticky-free and the operations in F are K-delimited. A set K of words is a comma-free code[1] if no three words u, v, w in K satisfy the equation uv = xwy – see [16] for explanations on K-delimited operations. It can be shown that, under these assumptions about K and F, the language K+ is F-invariant. Moreover, by the results of the previous section, K+ is also strictly t-free and strictly t-sticky-free. Thus, if the initial multi-set A contains only words from K+ then the computation language of SYS will be a subset of K+.

In [13] we provide a sequence K(n;m) of comma-free codes satisfying the above properties such that the information rates of these codes tend to (1 – 1/m) as n tends

[1] The concept of comma-free code was first introduced in [9]. At that time it was believed by many that the biological code is comma-free, but this conjecture was disproved later with the work of Niernberg [2]. Nevertheless, comma-free codes continue to be of interest and, in fact, they have been used in deep space communications [21].

to infinity – the parameter m is fixed but arbitrary. An example of such a code is the set K(1;3) that consists of the following DNA words:

AAATCCC,	AAATAATCCC,	AAATACTCCC
AAATAGTCCC,	AAATATTCCC,	AAATCATCCC
AAATCCTCCC,	AAATCGTCCC,	AAATCTTCCC
AAATGATCCC,	AAATGCTCCC,	AAATGGTCCC
AAATGTTCCC,	AAATTATCCC,	AAATTCTCCC
AAATTGTCCC,	AAATTTTCCC.	

According to the preceding discussion, any arbitrary collection of strands that are made of the above code words will never form any of the hybridizations shown in Fig. 1 – 3.

The set of all multi-set systems over a large alphabet Σ is powerful enough to simulate any Turing machine [14]. By the results of [13], it is possible to encode an arbitrary multi-set system T with an appropriate K(n;m)-based system SYS, which uses the DNA alphabet {A, C, G, T}, such that the results of the computations of SYS are equivalent to those of the system T – see [13] for more precise explanations.

IV. Adding Error-detection Capabilities

In addition to hybridizations, random nucleotide errors might occur in DNA strands. These errors could be substitutions, insertions, and deletions. More specifically, the nucleotide A, say, in a strand xAy can be substituted by a different nucleotide, say T, resulting in the strand xTy. The most common types of substitution errors are transitions (C by T, T by C, A by G, G by A) and transversions (C by A, A by C, C by G, G by C, T by A, A by T, T by G, G by T) [18]. Another possibility is that the nucleotide A might be deleted from the strand xAy, which would result in the strand xy, or it might be inserted in a strand of the form xy resulting in the new strand xAy. Here we consider a channel model in which at most one substitution, insertion, or deletion error is permitted in any m consecutive nucleotides of a DNA strand, where m is a fixed but arbitrary parameter. We use the expression sid(1, m) to denote such a channel.

Suppose that the computation language of interest is L and is expected that only words in L can be decoded. Then any channel errors applied to the words of L should be detected. In general, if a language L is error-detecting for a given channel then no word of the language can be transformed to another word of the language using the errors permitted by the channel – see [17] for the property of error-detection for arbitrary channels. The problem of constructing languages capable of detecting various error combinations is, in general, non-trivial. Here we are interested in the case where the language L is of the form K+, where K is a comma-free code of the type considered in the previous section and satisfies additional properties that would ensure that K+ is error-detecting for the channel sid(1,m).

For any DNA word w, we define the parity symbols pc(w) and pg(w) as follows [16]:

- pc(w) = A or T, depending on whether w contains an odd or even, respectively, number of A's and C's.
- pg(w) = A or T, depending on whether w contains an odd or even, respectively, number of A's and G's.

Let x be any DNA word of the form CzG, where z contains only symbols from {C, G} (if any), with the property that x is equal to t(x) – CG and CCGG are examples of such words. Let i be the length of x. Consider the code K consisting of all the words of the form

$$x\ C\ w\ pc(w)\ pg(w)\ T$$

where w is any DNA word of length (m – i – 4) with the property that the pattern x does not occur in any position of xCw other than the first. In [16] it is shown that the language K+ is error-detecting for the channel sid(1,m), strictly t-free and strictly t-sticky-free, and F-invariant for any set F of K-delimited operations. A concrete example of the code K is the following.

CGCAATTT,	CGCTAAAT,	CGCCATAT,
CGCACTAT,	CGCTCATT,	CGCCATAT,
CGCAGATT,	CGCTGTAT,	CGCCTATT
CGCATAAT,	CGCTTTTT,	

According to the preceding discussion, the language K+ is error-detecting for the channel sid(1,8).

V. Discussion

In [16] we performed a few empirical tests for checking whether certain DNA languages posses the three properties considered in Section II. Here we present some of these tests on the DNA encoding used in Adleman's experiment [1] for computing a Hamiltonian path in a given directed graph. In this problem the question is whether there is a path starting at the input node, ending at the output node, and passing through all the nodes exactly once. In Adleman's DNA solution to the problem, each node and each edge was encoded using a 20-letter long DNA sequence. Table 1 shows the results of testing whether the set of nodes and the set of edges, taken separately and together, have the three encoding properties we have defined.

We also tested the same data for the modified properties of 0.85 strictly t-compliance and 0.85 strictly t-freedom.

TABLE 1

	Edges	Nodes	Both
strictly t-compliant	yes	yes	no
strictly t-free	yes	yes	no
strictly t-sticky-free	no	no	no

TABLE 2

	Edges	Nodes	Both
0.85 strictly t-compliant	yes	yes	no
0.85 strictly t-free	yes	yes	no

A language L is 0.85 strictly t-compliant if there are no two words in L of the form xuy and t(v) such that at least 85% of the corresponding nucleotides in u and t(v) are equal – see [16] for more details on the refined properties. The results of these tests are shown in Table 2.

The empirical tests suggest that our definitions of good encodings are quite promising. Directions for further research include a detailed investigation of the refined properties.

ACKNOWLEDGMENT

We thank Len Adleman for providing the DNA sequences that were used in his 1994 experiment.

REFERENCES

[1] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol 266, pp. 1021 – 1024, 1994.

[2] J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, Orlando, 1985.

[3] R. Deaton, R. Murphy, M. Garzon, D. R. Franceschetti, S. E. Stevens, "Good encodings for DNA-based solutions to combinatorial problems," in *Proc. of DNA-Based Computers II*, Princeton and in *AMS DIMACS Series*, vol. 44, L. F. Landweber, E. Baum, (eds.), pp. 247 – 258, 1998.

[4] R. Deaton, M. Garzon, R. Murphy, D. R. Franceschetti, S. E. Stevens, "Genetic Search of Reliable Encodings for DNA Based Computation," in *Proc. First Conference on Genetic Programming GP-96*, Stanford U., pp. 9 – 15, 1996.

[5] R. Deaton, R. E. Murphy, J. A. Rose, M. Garzon, D. R. Franceschetti, S. E. Stevens Jr., "A DNA based implementation of an evolutionary search for good encodings for DNA computation, " in *Proc. IEEE Conference on Evolutionary Computation ICEC-97*, pp. 267 – 271, 1997.

[6] U. Feldkamp, S. Saghafi, H. Rauhe. "DNASequenceGenerator: A program for the construction of DNA sequences, " in [12], pp. 179 – 189.

[7] M. Garzon, P. Neathery, R. Deaton, R. C. Murphy, D. R. Franceschetti, S. E. Stevens Jr., "A new metric for DNA computing, " in J. R. Koza, K. Deb, M. Dorigo, D. B. Vogel, M. Garzon, H. Iba, R. L. Riolo, (eds.), *Proc. 2nd Annual Genetic Programming Conference*, Stanford, CA, pp. 472 – 478, 1997.

[8] M. Garzon, C. Oehmen, "Biomolecular computation in virtual test tubes, " in [12], pp. 75 – 83.

[9] S. W. Golomb, B. Gordon, L. R. Welch, "Comma free codes," *Canadian Journal of Mathematics*, vol. 10, pp. 202 – 209, 1958.

[10] A. J. Hartemink, D.K. Gifford, J. Khodor, "Automatic constraint-based nucleotide sequence selection for DNA computations, " in *Proc. DNA-Based Computers IV*, Philadelphia and in *Biosystems*, vol. 52, L. Kari, H. Rubin, D. H. Wood, (guest eds.), pp. 227 – 235, 1999.

[11] T. Head, "Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors," *Bull. Math. Biology*, vol. 49, pp. 737 – 759, 1987.

[12] N. Jonoska, N. C. Seeman, (eds), "DNA computing: DNA-Based Computers VII, Tampa, Florida, 2001, " Lecture Notes in Comp. Science, vol. 2340, Springer, 2002.

[13] S. Hussini, L. Kari, S. Konstantinidis. "Coding properties of DNA languages, " in [12], pp. 107 – 118 and in *Theoret. Comp. Science*, vol. 290, pp. 1557 – 1579, 2003.

[14] L. Kari. "DNA computing: arrival of biological mathematics," *The Mathematical Intelligencer*, vol. 19, nr.2, pp. 9 – 22, 1997.

[15] L. Kari, R. Kitto, G. Thierrin. "Codes, involutions and DNA encoding., " in *Lecture Notes in Comp. Science*, vol. 2300, pp. 376 – 393, 2002.

[16] L. Kari, S. Konstantinidis, E. Losseva, G. Wozniak, "Sticky-free and overhang-free DNA languages," *Acta Informatica* (to appear).

[17] S. Konstantinidis, A. O'Hearn, "Error-Detecting Properties of Languages, " *Theoret. Comp. Science*, vol. 276, pp. 355 – 375, 2002.

[18] B. Lewin, *Genes VII*, Oxford Univ. Press, 2000.

[19] A. Marathe, A. Condon, R. Corn, "On combinatorial DNA word design, " in *Proc. DNA based Computers V*, pp. 75 – 89, 1999.

[20] G. Paun, G. Rozenberg, A. Salomaa, (eds), *DNA Computing: New Computing Paradigms*, Springer, Berlin, 1998.

[21] S. Wicker, "Deep space applications, " in *Handbook of Coding Theory vol. II*, chapter 25, pp. 2119 – 2169, Elsevier, 1998.