Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 171 (2007) 95–104

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Computing by Floating Strings

Giuditta Franco [1,2]

Department of Computer Science
University of Verona
Verona, Italy

Maurice Margenstern [3]

Laboratory of Theoretical and Applied Computer Science
Université Paul Verlaine - Metz
Metz, France

Abstract

We propose a biologically inspired system which computes on double structures of mobile strings by means of rewriting rules that have a biotechnological implementation, by resembling DNA computations. Its computational universality may be straight deduced from a few formal language theoretical results of one of the authors. Such a DNA-like computational device is naturally described by a membrane system which concludes the paper.

Keywords: DNA Computing, Double String Structure, Floating Strings, Membrane Systems, Register Machine Program.

1 Introduction

DNA molecules are composed of two paired strands, such that each symbol of a first one corresponds to its 'complementary' in the second one. The strands are oriented along opposite (called anti-parallel) directions. These three basic properties are respectively referred to as: bilinearity, complementarity and antiparallelism of DNA double strands, and they depend on some informational and computational aspects underlying the DNA autoduplication process [2].

The genome is the complete set of DNA contained in any cell of an organism, specifying how it will look like under a certain set of environmental conditions. It is

[1] The first author thanks Vincenzo Manca for giving her the inspiration to work on double structures by means of bio-inspired string operations. Both authors are very thankful to NATO grant PST.CLG.976912, the institution which kindly founded the presentation of this paper.

[2] Email: franco@sci.univr.it

[3] Email: margens@univ-metz.fr

doi:10.1016/j.entcs.2007.05.010

duplicated whenever the cellular division occurs, and the double stranded molecule is copied inside the cell in such a way that each strand works as template for one of the strands to assembly. Thus, nature employs a template driven algorithm to copy DNA strings, with a computational complexity linear in time and space. In a formal context of oriented symbols and strings floating in a fluid environment, the following (starting, doubling and separating) rules guide a *bilinear duplication algorithm* [2] where two copies of a given string are systematically obtained.

Let V be the alphabet of the string α to duplicate, while $\bullet$ being a meta-symbol describing the single step of computation, $\beta, \gamma \in V^\star$, λ denotes the empty string, and the symbol x in V is such that $x \neq \lambda$.

(i) *Starting*

$$\alpha \;\rightarrow\; \frac{\bullet\, \alpha}{\lambda}$$

(ii) *Doubling*

$$\frac{\alpha \;\bullet\; x\, \beta}{\gamma} \;\rightarrow\; \frac{\alpha\, x \;\bullet\; \beta}{\gamma\, x}$$

(iii) *Separating*

$$\frac{\alpha \,\bullet}{\beta} \;\rightarrow\; \alpha,\ \beta$$

If a string is used as a template, and a copy of each element is put 'face-to-face' to the template, along a line that goes 'parallel' to the original one, then a duplication algorithm based on a bilinear structure of strings and deterministic rewriting rules is obtained, in fact in the last step we have $\beta = \alpha$. In the case of DNA duplication, in the second step a symbol uniquely associated to x (the complementary one) is put 'face-to-face' to the template, and finally β results to be the Watson Crick complementary string of α.

If one wants to design an efficient duplication system for strings (finite sequences of symbols) represented as mobile polymers (floating in a fluid environment), then one needs 'symbol molecules' asymmetric with respect to the three space directions (as the molecules are from a chemical point of view), and arranged according to the three principles of bilinearity, complementarity and antiparallelism [2]. Moreover, some intrinsic features of DNA molecules, such as the geometry of the double helix, are proven to be implied by these principles [4], on which DNA processes are based on.

The rules of the bilinear duplication algorithm have a clear biotechnological implementation. In fact, the doubling step can be performed by a polymerase enzyme, and the separating step by a denaturation operation, due to an increase of the temperature, while the first step is just the formal passage from a single string to a string placed into a double structure. The biological interpretation of the symbol $\bullet$ is related to the orientation of the string: the computation starts only when it is at the beginning of a string with respect to the concatenation verse, enzymes can recognize this point from the chemical structure of the molecule.

One of the (first) natural questions to ask about DNA computing concerns its

potential to implement arithmetical operations, and, to this aim, the interest for building-in DNA Turing machines was widely addressed (see for example [11] and [12]). Along this direction, in [1] the implementation of a two states autonomous DNA automaton was shown, beside a logical design for a biomolecular Turing machine, unfortunately forced to wait until the parts needed to build it are invented. On the same research line, although still at a just theoretical level, in this paper we propose a universal DNA-like system which computes on double structures of mobile strings by means of rewriting rules that have a biotechnological implementation.

Essentially three types of operations are enough in order to obtain universal computations, as one can see from [6,7,8]; they are duplication, triplication, and division by six. Our system processes the information encoded on floating strings by simulating these three operations. It is able to execute any program of a register machine, since it has also a mechanism to switch from one instruction to another one and to perform jump instructions.

Finally, in Section 3, we present the natural description of our DNA-like system by means of a membrane structure. Indeed membrane systems are biologically inspired containers of floating strings. They are a discrete device (only discrete quantities are involved) which takes explicitly into account the notion of compartments [9], whereas the way how the evolution rules work recalls the computational strategies of formal grammars, that traditionally were applied to the analysis of biological structures [3]. In Nature eukaryotic cells keep together the genes, the RNAs and proteins that they encode, as well as the products of their activities, and cellular compartmentalization is the key of the organization of biomolecular systems, such as networks of proteins underlying the main cellular functions. It is essential for the evolution of all living organisms.

2 DNA-like system

We present a formal system based on the double string structure of mobile strings, that is able to duplicate, triplicate, and divide by six a number n, and to switch from performing one instruction to another one of a register machine program. By [5,6,7,8] we know that this suffices in order to perform universal computations, as all the basic instructions of a register machine can be reduced to "times 2" or "times 3" or "division by 6" instructions.

In a sense this system is an extension of the floating bilinear duplication algorithm introduced in [2], since (from a theoretical point of view) the computation can be implemented on DNA strings; the crucial difference though is the encoding used to represent numerical information by a floating string.

In fact, the output of the bilinear duplication is given by two copies of the initial string, and from the same viewpoint, the product of the triplication would be given by three copies of the initial string and the division by six would reduce six copies of the string to one copy. That is, the numerical information which is processed by the operations consists of the number of copies of the initial string. Here instead, we consider the information as completely carried by (one copy of) the current string,

regardless to the number of copies of such a string. It is transformed by means of operations in each step of computation, as usually on the tape of a Turing machine. The initial string encodes the given numerical input, and when one of the operations modifies it, the string encoding the number obtained after the application of that operation is produced.

Let V be the alphabet with four symbols $\{1, 2, 3, \#\}$, and n a numerical input. The factorization in primes of n can be seen as $2^k m 3^h$, with $m \in \mathbb{N}$ and non-divisible by 2 or 3. Since every m can be written in unary code, then every numerical input n can be considered as a particular string over V, with a number k of 2s as prefix and a number h of 3s as suffix that are uniquely implied by the value of n. In particular, we associate n to the string $2^k \# m \# 3^h$ belonging to $V^\star$.

With our encoding, in order to duplicate a number it is enough to juxtapose the symbol 2 at the beginning (with respect to the concatenation verse) of the string encoding the number, and in order to triplicate one has to concatenate the symbol 3 at the end. The division by six is performed by eliminating the first and the last symbol of the string if both of them are different from #, otherwise the number is not divisible by six.

The system works with deterministic rewriting rules over double string structures, where a special meta-symbol of kind $\bullet$ marks the point of progress of the computation. In particular, we consider different pointers corresponding to the p instructions of the program we want to simulate, they belong to the set Q of pointers and have an index representing an instruction label, $Q = \{\bullet_1, \bullet_2, \ldots, \bullet_p\}$. In the case of instructions which are decrements, that correspond to divisions [5], during the computation we use further (primed) indexes, among $\{1', \ldots, p'\}$. However the total number of the involved pointers is bounded by $2p$, where p is the number of the instructions of the given program.

In the following, we have three groups of rewriting rules, where λ denotes the empty string (in $V^\star$), $\alpha, \beta, \gamma, \xi \in V^\star$ with $\beta, \xi \neq \lambda$, whereas $a, b \in V$, $j \in \{1, \ldots, p\}$ and $j' \in \{1', \ldots, p'\}$. We refer to ξ' to denote the string obtained from ξ after having erased its first symbol.

(i) *Starting*

$$2^k \# m \# 3^h \longrightarrow \frac{\bullet_1\ 2^k \# m \# 3^h}{\lambda}$$

This step marks the start of the computation, by transforming the input from single string into double string structure. After such a step the first instruction is going to be executed, as required by the presence of the pointer with index 1.

(ii) This multiple step performs the operation indicated by the jth instruction of the program (whenever it can be performed), for $j \in \{1, \ldots, p\}$, that can be a duplication, a triplication, a jump, or a division by six.

(a) *Doubling* (for j belonging to the labels such that the corresponding instructions are duplications)

$$\frac{\bullet_j\ a\ \alpha}{\lambda} \longrightarrow \frac{a\ \bullet_j\ \alpha}{2a}$$

$$\frac{\gamma\ \bullet_j\ a\,\alpha}{2\gamma} \longrightarrow \frac{\gamma\,a\ \bullet_j\ \alpha}{2\,\gamma\,a}$$

(b) *Tripling* (for j belonging to the labels such that the corresponding instructions are triplications)

$$\frac{\alpha\ \bullet_j\ a\,\beta}{\alpha} \longrightarrow \frac{\alpha\,a\ \bullet_j\ \beta}{\alpha\,a}$$

$$\frac{\alpha\ \bullet_j\ a}{\alpha} \longrightarrow \frac{\alpha\,a\,\bullet_j}{\alpha\,a\,3}$$

(c) *Jumping from j to l* (if the label j corresponds to an instruction which is a jump to the l instruction)

$$\frac{\bullet_j\,\alpha}{\lambda} \longrightarrow \frac{\bullet_l\,\alpha}{\lambda}$$

(d) *Dividing by six* if possible, otherwise *jump from j to l* instruction (j belonging to the labels such that the corresponding instructions are divisions by six, which can be performed if, and only if, both the first and the last symbol of the current string are different from #)

$$\frac{\bullet_j\,2\,\alpha}{\lambda} \longrightarrow \frac{2\ \bullet_j\ \alpha}{\lambda}$$

$$\frac{\bullet_j\,\#\,\alpha}{\lambda} \longrightarrow \frac{\bullet_l\,\#\,\alpha}{\lambda}$$

$$\frac{\xi\ \bullet_j\ a\,\beta}{\xi'} \longrightarrow \frac{\xi\ a\ \bullet_j\ \beta}{\xi'\,a}$$

$$\frac{\xi\ \bullet_j\ 3}{\xi'} \longrightarrow \frac{\xi\,3\,\bullet_j}{\xi'}$$

$$\frac{\xi\ \bullet_j\ \#}{\xi'} \longrightarrow \frac{\xi\,\#\,\bullet_{j'}}{\xi'\,\#}$$

$$\frac{\xi\,\bullet_{j'}}{\xi'} \longrightarrow \frac{\xi\,\bullet_{j'}}{\bullet_l\,2\,\xi'}$$

(iii) *Switching and Separating*

(a)

$$\frac{\alpha\,\bullet_j}{\beta} \longrightarrow \frac{\alpha\,\bullet_j}{\bullet_{j+1}\beta}$$

(b)

$$\frac{\alpha\,\bullet_i}{\bullet_j\ \gamma} \longrightarrow \frac{\bullet_j\ \gamma}{\lambda}$$

(c)

$$\frac{\bullet_{p+1}\ \gamma}{\lambda} \longrightarrow \gamma$$

Whenever a string which is preceded by the pointer $\bullet_{p+1}$ is transformed in a single string (that is the step 3 part (c) is executed), then the program halts and the single string encodes the numerical result of the computation.

Given a register machine program, it may be simulated by a program with p labeled instructions that double, triplicate, or divide by six the numerical information, or represent a jump. In the above system the first step is performed only at the beginning of the program. Then, whenever the current label j points to a doubling or a triplicating instruction, the rewriting rules of the step 2, part (a) and part (b) respectively, are applied in a deterministic way. In both these cases the operation is completed when no other rule related to that operation can be applied, and a double structure of kind $\frac{\alpha\bullet}{\beta}$ is finally obtained. If label j refers to a jump, the rule of the step 2 part (c) is applied, so that the operation labeled by l can be performed. If j corresponds to a division by six, then the rewriting rules of the step 2 part (d) are applied. If the current string begins with the symbol 2 and ends with the symbol 3 (that means that it represents a number divisible by six, and the division can be performed), then the computational process leads to obtain a double structure of kind $\frac{\alpha\bullet}{\beta}$. Otherwise, a jump is performed whenever the first symbol is equal to #, and a double structure of kind $\frac{\alpha\bullet}{\bullet\beta}$ is obtained whenever the last symbol is equal to #.

The switching and separating rules of step 3 lead deterministically the computation to perform the next operation, or to give out an output. In particular, the (a) rule switches to the next instruction (the $(j+1)$th one), after that the operation j has been performed regularly, while the (b) rule allows the corresponding instruction to start even after an operation of division by 6 which was not performed. The (c) rule is applied at the end of the program, and it gives out the current string as output.

Note that we have an alphabet with exactly four symbols and the typical DNA string double structure. Moreover, the biochemical implementation of the main step (the second one) recalls the work of a special polymerase, since symbols are copied from a template by elongating a primer with concatenation, and the very last step recalls the annealing process.

In the case of duplication, a translation of the first symbol a into the string $2a$ has to be done (i.e., a symbol 2 and a symbol a are juxtaposed as lower string of the double structure) before performing the copying of the remaining string, and in the case of triplication, a translation of the last symbol a into the string $a3$ has to be done after a usual copying of the string. While in the case of division, the copying is performed after the elimination of both the first and the last symbol of the string.

There are some enzymes called exonuclease with the ability to cut away the extremal symbols of a string. Unfortunately, from an experimental point of view, it is really difficult to control these enzymes in such a way that only one letter is removed. At a theoretical level though, in all these cases, the natural orientation of the DNA strings is a mechanism to recognize which is the first symbol and which is the last one of a string, and enzymes as polymerase or exonuclease are able to

recognize these points.

The exchange of pointers and the appearing of a pointer in the lower string can be seen as indications corresponding respectively to what enzyme is going to read and copy and from where. In standard DNA computations, the enzymes are chosen by humans, and the point of progress of the computations is given by the natural orientation of DNA strings. At this stage, the autonomy of our computations basically depends on the actual ability to control the copying operation in a wet lab, i.e. it depends on the existence of suitable enzymes performing our operations.

Still from a biological point of view, finally we could read the two final DNA strings by means of the Sanger method, but it is enough (and easier) to perform an electrophoresis, because the output string can be discriminated by means of its length. In fact, in the case of duplication and triplication the output is represented by the longest string, and in the case of division by the shortest one. A little modification on the algorithm can easily be done in order to increase the difference of length among the final strings and to make the implementation by electrophoresis more realistic.

3 A membrane DNA-like system

Membrane systems are proposed as a model to represent various aspects of molecular localization and compartmentalization [10], including the movement of molecules between compartments, the dynamic rearrangement of molecular reactions, and the interaction between molecules in a compartmentalized setting [9].

We associate the DNA-like system introduced in the previous section to a membrane system in a very natural way. In fact, since the computation is guided by the value of the pointer index, then it can be performed in a single compartment related to the kind of index, while the rewriting rules are applied along the classic strategy, where "all objects which can evolve must do so" [9]. The difference here is that the objects are double string structures, and the rewriting rules in the membrane regions can be viewed as enzymes or biological agents able to read and to write on the DNA double structure.

Let P be a register machine program with p labeled instructions being duplications, triplications, divisions by six, and jumps. We call A the set of labels such that the corresponding instructions are duplications, B the set of labels corresponding to triplications, C the set of labels that correspond to jumps, and D the set of labels such that the corresponding instructions are divisions by six. Of course, A, B, C, D form a partition of the label set.

Starting from the same setting of the DNA-like system and the same encoding for the strings, but considering the pointers as special symbols of the alphabet, we introduce a membrane system that has four distinct membranes, called m_a, m_b, m_c and m_d, inside the skin membrane m_s (see Figure 1). Although the notation follows what was introduced in the previous section, it might be worth recalling that k, h are any natural numbers, m is a string over the alphabet $\{1\}$, $a \in V$, and l corresponds to the instruction to which j indicates a jump.

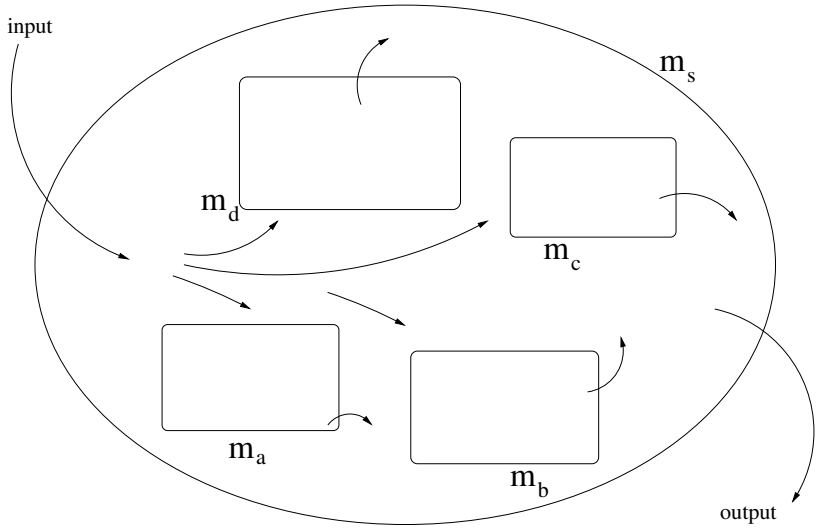

Figure 1. Membrane DNA-like system.

The rules related to the region of the skin membrane belong to the set

$$R_{m_s} = \{2^k\#m\#3^h \longrightarrow \frac{\bullet_1\, 2^k\#m\#3^h}{\lambda},\quad \frac{\alpha\,\bullet_j}{\beta} \longrightarrow \frac{\alpha\,\bullet_j}{\bullet_{j+1}\beta},$$

$$\frac{\alpha\,\bullet_i}{\bullet_j\,\gamma} \longrightarrow \frac{\bullet_j\,\gamma}{\lambda},\quad \frac{\bullet_{p+1}\,\gamma}{\lambda} \longrightarrow \gamma\}$$

for $i, j \in A \cup B \cup C \cup D$ and $\alpha, \beta, \gamma, \in V^\star$ with $\beta \neq \lambda$.

The rules related to the region of the membrane m_a belong to the set

$$R_{m_a} = \{\frac{\bullet_j\, a\, \alpha}{\lambda} \longrightarrow \frac{a\ \bullet_j\ \alpha}{2a},\ \frac{\gamma\ \bullet_j\ a\, \alpha}{2\gamma} \longrightarrow \frac{\gamma\, a\ \bullet_j\ \alpha}{2\,\gamma\, a}\}_{j\in A}$$

The rules related to the region of the membrane m_b belong to the set

$$R_{m_b} = \{\frac{\alpha\ \bullet_j\ a\,\beta}{\alpha} \longrightarrow \frac{\alpha\, a\ \bullet_j\ \beta}{\alpha\, a},\ \frac{\alpha\ \bullet_j\ a}{\alpha} \longrightarrow \frac{\alpha\, a\, \bullet_j}{\alpha\, a\, 3}\}_{j\in B}$$

The rules related to the region of the membrane m_c belong to the set

$$R_{m_c} = \{\frac{\bullet_j\,\alpha}{\lambda} \longrightarrow \frac{\bullet_l\,\alpha}{\lambda}\}_{j\in C}$$

Finally, the rules related to the region of the membrane m_d are:

$$R_{m_d} = \{\frac{\bullet_j\, 2\,\alpha}{\lambda} \to \frac{2\ \bullet_j\ \alpha}{\lambda},\ \frac{\bullet_j\,\#\,\alpha}{\lambda} \to \frac{\bullet_l\,\#\,\alpha}{\lambda},\ \frac{\xi\ \bullet_j\ a\,\beta}{\xi'} \to \frac{\xi\, a\ \bullet_j\ \beta}{\xi'\, a},$$

$$\frac{\xi\ \bullet_j\ 3}{\xi'} \to \frac{\xi\, 3\, \bullet_j}{\xi'},\ \frac{\xi\ \bullet_j\ \#}{\xi'} \to \frac{\xi\,\#\,\bullet_{j'}}{\xi'\,\#},\ \frac{\xi\,\bullet_{j'}}{\xi'} \to \frac{\xi\,\bullet_{j'}}{\bullet_l\, 2\,\xi'}\}_{j\in D}$$

Single strings α (on the given alphabet $V = \{1, 2, 3, \#\}$) enter the skin membrane from the environment, whereas a double structure $\frac{\bullet_i \alpha}{\lambda}$ enters membrane m_a whenever $i \in A$, enters membrane m_b whenever $i \in B$, enters membrane m_c whenever $i \in C$ and membrane m_d whenever $i \in D$. Double structures (or strings) exit from a membrane when no more rules can be applied in that membrane.

For any given register machine program we can make the above membrane system, the functioning of which is very simple: it receives from the environment a (single) string as input, processes it by performing the instructions of the program and gives out the (single) string encoding the output of the computation. In fact, whenever a string α enters skin membrane, the only rule that can be applied in the skin region on a single string is applied, and the structure $\frac{\bullet_1 \alpha}{\lambda}$ enters the membrane corresponding to the index set which 1 belongs to. Inside any internal membrane an operation is performed, while in the skin region the switching from one instruction to another happens.

This system is universal and turns out to be a model very close to biology, in fact any internal membrane represents the action of a special polymerase enzyme which copies on the double structure, and the rules of that region identify the kind of enzyme. Note that the membranes float in the skin region, in fact the enzymes can move inside a region but their different actions do not interfere with each other.

4 Future Work

We conclude by sketching a slight variant of this membrane system that can execute many programs in the same time. If we have k programs $P_1, \ldots, P_k$, with $p_1, \ldots, p_k$ number of instructions respectively (program P_i has p_i instructions), then we can take A, B, C, D as the union of the sets of the membrane labels for all the programs, that indicate duplication, triplication, jump, and division by six respectively. To perform k programs in parallel by our membrane system, it is enough that the instruction labels are represented by different kinds of symbols for different programs, for example $\{l_1^{(1)}, \ldots, l_{p_1}^{(1)}\}$ for P_1 and $\{l_1^{(i)}, \ldots, l_{p_i}^{(i)}\}$ for P_i with $i = 2, \ldots, k$. There are rules corresponding to each of these labels, but the kind of rules is the same of the above system, only the cardinality of the sets A, B, C, D (as well as that of $R_{m_a}, R_{m_b}, R_{m_c}, R_{m_d}$) increase, while the number of membranes being the same. In this way, no confusion arises among pointers indicating instructions of different programs, and a number k of programs can theoretically be executed simultaneously by a system with five membranes such as that in Figure 1.

References

[1] Benenson, Y., T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro, *Programmable and Autonomous Computing Machine made of Biomolecules*, Nature **414** (2001) 1, 430-434.

[2] Franco, G., and V. Manca, *An Algorithmic Analysis of DNA Structure*, Soft Computing - A Fusion of Foundations, Methodologies and Applications **9** (2005) 10, Springer, 761-768.

[3] Lindenmayer, A., *Mathematical Models for Cellular Interaction in Development. Part I and Part II*, Journal of Theoretical Biology **18** (1968), 280-315.

[4] Manca, V., *On the Logic and Geometry of Bilinear Forms*, Fundamenta Informaticae **64** (2005), 257-269.

[5] Margenstern, M., *Une machine de Turing universelle non-effaçante à exactement trois instructions gauches*, C. R. Acad. Sci. Paris, t. 320, Série I (1995), 101-106.

[6] Margenstern, M., *On quasi-unilateral universal Turing machines*, Theoretical Computer Science **257** (2001), 153-166.

[7] Margenstern, M., and L. Pavlotskaïa, *Deux machines de Turing universelles à au plus deux instructions gauches*, C. R. Acad. Sci. Paris, t.320, Série I (1995), 1395-1400.

[8] Margenstern, M., and L. Pavlotskaïa, *On the optimal number of instructions for universal Turing machines connected with a finite automaton*, Preproceedings of Cellular Automata Workshop, M. Kutrib and T. Worsch eds, 1996.

[9] Păun, Gh., "Membrane Computing. An Introduction", Springer-Verlag New York, 2002.

[10] Regev, A., E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro, *BioAmbients: An Abstraction for Biological Compartments*, Theoretical Computer Science – Special Issue on Computation Methods in Systems Biology **325** (2004) 1, 141–167.

[11] Rogozhin, Y., *Small Universal Turing Machines*, Theoretical Computer Science **168** (1996), 215-240.

[12] Shapiro, E., and Y. Benenson, *Bringing DNA Computers to Life*, Scientific American Inc., 2006.