DOI: 10.2478/awutm-2019-0009

Analele Universităţii de Vest,
Timişoara
Seria Matematică – Informatică
LVII, 1, (2019), 97– 121

Some Wonders of a Bio-Computer-Scientist

Gheorghe Păun

Dedicated to the memory of Professor Ştefan Măruşter

1 Preliminaries

The previous title takes some precautions, which are visible, but still I want to stress them.

First, the autobiographical character. The last two decades – to be precise, the last twenty two years – I was totally dedicated to bio-computability, one can even say that I was *confiscated* by this research area, so fascinating, promising, endless in possibilities of theoretical developments and of applications. The beginning is placed in the spring of 1994, when I read a paper by Tom Head, a wise American who later became a friend and a collaborator of mine, who, already in 1987, proposed a formal language theory model for the recombination operation of DNA under the influence of restriction enzymes and ligase. He has named it the *splicing* operation. I will shortly describe it later. I remember it exactly – I was in Austria, in Graz, participating in a conference. On the one hand, I became immediately enthusiastic about this idea – after two decades of research in formal language theory, I was subconsciously looking for areas to apply this theory, on the other hand, I was somewhat unsatisfied, because Head's formalization was complex, it remained, to my taste, too close to the biological reality. Right then, in the hotel in Graz, I have imagined a simpler definition, closer to the style I was used, I can call it *Salomaa-Marcus style.* Four years after that I have worked

This is the English version of a lecture delivered on December 8, 2016, on the occasion of receiving a Doctor Honoris Causa title from the West University of Timişoara, Romania.

in this area. Until 1998, when I had a second thrill, after defining a model inspired from the architecture and the functioning of the living cell, the first one from a families of models which continues to grow. After that, I have totally identified myself with *membrane computing*, a research area which transformed me in a letter and which will be the territory through which I will invite the reader in what follows. (I will sometimes use the abbreviation MC for *membrane computing.*)

I return to the title. Wonders-questions, not models-results-applications, although the former are connected to the latter. Places where the mathematician who I am by education and the computer scientist with twenty years of theoretical research – I have graduated mathematics in 1974 – remains still, with raised elbows, either not understanding, or expecting something different, or faced with objects coming from biology which, seen through the glasses of the theoretical computer scientist, suggest computability ideas-models totally new for theoretical computer science and, much more, for practical computer science, in the forms we know them from books and from implementations. During billions of years of evolution (I adhere here to the evolutionist paradigm, this is not a suitable place for an evolution version creation discussion, furthermore, for what follows *we do not need any creationistic hypothesis*, to quote a phrase assigned to Laplace, but also to other big names of physics), life has evolved many processes and supports for these processes of a high precision, remarkably subtly and efficient, which can be transferred, at least at the theoretic level, to computer science. How to do that, asks himself the mathematician-computer scientist? With what practical consequences, asks himself the computer scientist interested in applications? Why so many computability aspects, as they were developed up to now, look *non-natural*, far from the "real reality", from the biological one?

Some of these questions get close to the science fiction (SF) frontier of science, but I will not avoid them, as long as the science, physics in particular, does not totally refute them as non plausible. We are placed here under the flag of the possible, not of the probable...

There will also appear questions-wonders which are more precise, more localized. Only part of those which I have reached, significantly fewer than those which were possibly met by other colleagues of bio-computing. That is why the title starts with *some*...

A few remarks about the style. Although I will refer to many notions, from biology and from computer science, although in the end I will provide a pretty comprehensive bibliography (also this one will be mainly autobiographical), the discussion will be informal, without precise references.

In what concerns the bibliography, one of the items is my Reception Discourse in the Romanian Academy, delivered on October 24, 2014, and having the privilege to be followed by a response by Solomon Marcus, the Professor, always with capital initial letter. The present pages can be seen as a continuation of the Discourse, at a more informal-colloquial level, but also as an opportunity to pay my homage to my Professor, who constantly used to promote the *state of wondering, of questioning yourself*. Sadly, the Professor passed away in March 2016.

Above all these, this text is an invitation to the reader to wonder together...

2 How much biology? How much computability?

Before mentioning any reason to wonder-question ourselves, it could be useful to roughly specify the framework: what/how much biology, what/how much computability are necessary for what follows?

Mainly cellular biology, at a general level: membranes which enclose compartments, "protected reactors" where specific biochemical reactions take place, involving "reactants" (from ions to large macromolecules) which swim in an aqueous solution or they are bound on membranes or on the cytoskeleton, then protein channels which make possible the communication among compartments. Many other ingredients can be added, the biologists know a lot of things about the cell. The monograph *Molecular Biology of the Cell*, by Alberts, Johnson, Lewis, Raff, Roberts, Walter, has over 1500 pages of a large format. A true universe – of a nanometrical scale. The smallest entity about whose alive character there is no doubt (about viruses the opinions are mixed). A complex "factory", at the same time robust and fragile, very precisely and efficiently organized.

Already at this level we get an important question, but one mainly of a biological type: what is making the difference between *alive* and *non-alive*? Chemistry and physics cannot provide a convincing answer. The borderline seems to be defined in terms of organization, information, in terms of *something* which we cannot yet perceive, understand, and model. We may call it *soul*, or *vital breath*, just for moving the question away.

An important detail: the today biology, chemistry, physics are much developed in the study of their object of study in terms of substances and energy, but not also in terms of information. I do not enter into details, this opens the gate to many questions, starting with the definition itself of information, organization, complexity. Can we conceive information without any support, of a substance or energy type, the other two components of what we use to call *matter*? Two more remarks: (1) biology and physics seems to be on the verge of passing to "a new age" – for biology there were proposed some up-dated names, such as *infobiology* and *infobiotics*, and (2) computer science is a part of information theory, in the broad sense, therefore, the computational, algorithmic approach seems to be one of the ways through which the biology and the physics will evolve in the near future. Quantum computing seems to be one of the first steps for physics (Gruska), bio-computability seems to be half of the path for biology.

Let us return to the cell, with a slogan-"equation" launched by Solomon Marcus during one of the first MC meetings, in Curtea de Argeş, Romania, 2001:

Life = DNA software + membrane hardware.

It is stressed here the role played by membranes in the life of cells, thus also motivating the name which I gave to this research area, *membrane computing*; maybe a better name were *cellular computing.*

After the individual cell, I will also incidentally invoke populations of cells, tissues, organs, colonies of bacteria, in the end, also the "thinking cell", the neuron.

Details, in the next sections – here, the computer science basic prerequisites. First of all, Turing computability. The standard framework for most computability research. By Turing-Church Thesis, the highest level of algorithmic computability. At only 24 years, the genial Turing answered Hilbert's question on *what can be mechanically computed*, with a particular application to the decidability in arithmetics, by introducing what is now known under the name of *Turing machine*, the most general and the most convincing (for the contemporary researchers, Church, Kleene, Gödel etc., who tried in various ways to answer Hilbert's question) of the notion of an algorithm. Then, by the definition of the universal Turing machine (a machine able to simulate any particular Turing machine as soon as a code of the particular machine, together with an input for it, are placed as an input on the tape of the universal Turing machine), and by the theorem proving the existence of the universal Turing machine, Turing was signing the birth certificate of the computers as we have them today: when designing and constructing the first (programmable, as the existence of a universal Turing machine made possible) computers, at the beginning of forties, last century, John von Neumann was explicitly influenced by Turing ideas. From here, the name of Turing-von Neumann computers, from here their sequential character and their theoretical vulnerability to computer viruses (programs and data are placed in the memory, with the possibility of affecting programs by other programs in the same way as data are affected).

At the maximal level, the Turing machine, at the lowest level of computability, the finite automaton, a form of a Turing machine restricted in an extreme extent. These are the two poles of computability to which the bio-inspired computability is constantly referred to.

These are the two poles of the computability *competence*. From a practical point of view, at least equally important is the *performance*, the efficiency. In this way, it enters into the stage the famous distinction between the class of problems which can be solved in polynomial time (with respect to the size of the input) and the class of problems for which no polynomial algorithm is known, but, if somebody proposes a solution (if a solution can be “guessed”), we can check it in polynomial time. The famous complexity classes **P** and **NP**, the (in)famous problem whether or not $\mathbf{P} = \mathbf{NP}$, the first one in the list of “millennium problems” for whose solution the Clay Institute from USA offers one billion dollars.

I will also invoke grammars (especially Chomsky grammars); further details will be mentioned when they will be useful.

3 Starting from the DNA

Let us start with DNA, chronologically the first investigated and a relevant case study.

Today we are learning the language in which God created life, historically claimed Bill Clinton in the summer of 2000, when the reading of the human genome was completed. A language, indeed. Syntax, in the proper meaning of the term. Four “letters”, A (adenine), C (cytosine), G (guanine), and T (thymine), placed along two strands, with well specified pairs face to face, the so-called *Watson-Crick complementary pairs* (Watson and Crick have received the Nobel Prize, in 1962, for the discovery of the DNA structure, at the beginning of fifties): A is always in front of T, and C always forms a pair with G. This is the primary structure of DNA, the only one of interest here. The secondary structure, the double helix, and the ternary one, the complex folding, are not relevant for what it follows.

If we raise the temperature of the solution where the DNA molecules are placed, the two strands are separated. If we decrease back the temperature, then the nucleotides look for their Watson-Crick complementary pairs and the double stranded structure is rebuilt. Then, the *restriction enzymes* – somebody called them the most intelligent tools nature provided to the genetic engineers. They are proteins which recognize a short sequence of nucleotides, a pattern, a *context*, to come closer to linguistics, and cut the DNA molecules, in most cases in the middle of this pattern, in most cases, producing *sticky ends*, staggered strands: one of them is cut in a point, the other one some nucleotides away; the single strands of nucleotides are sticky, if they find their complements, then they get glued to them, completing the double strand.

A lot of further biochemistry can be found around (for instance, ligase, other enzymes, which help the reconstruction of the double strand, or the directionality of the two strands, whose ends are marked with $3'$ and $5'$, the biochemists know why, opposite on the two strands and preserved by the strands after getting separated), and still more computability. Also, similarly much linguistics – it was underlined by Solomon Marcus already in 1974, in a paper which appeared too early, so that it did not receive the audience it deserved.

Now, one first example – and the first surprise.

A result in formal language theory, published already in 1980 by Engelfriet and Rozenberg, says that any language which can be recognized by a Turing machine (equivalently: which can be generated by a Chomsky grammar; in the usual terminology: *recursively enumerable*), hence of the most general type among the computable languages, can be obtained starting from a certain fixed language, that is, independent of the starting language, applying to it a sequential transducer, the simplest type of a transducer, a finite automaton with output (one reads the input string, from left to right, without returns, and for each read symbol one outputs one or more symbols, also depending on a state from a finite set of states). The unique language from which we start, the one in which there are "hidden" all computable languages, is obtained as follows: consider the symbols 0, 1, consider their "complements" $0'$ and $1'$; take a string over the symbols 0, 1 and consider its *twin*, the string obtained by priming all symbols; mix the symbols of the string with the symbols of its twin, in all possible ways, preserving the ordering of symbols in each string – this operation is called *shuffle*. Finally, collect all strings obtained by shuffling all strings over 0, 1 with their twin strings. This is the language we are looking for – called the *twin-shuffle language* over 0, 1.

Four letters, $0, 1, 0', 1'$. Maybe this was the coincidence which suggested to Rozenberg and Salomaa the following operations: take a DNA molecule and "read" it, from left to right, on each strand, with random speeds on the two strands, writing 0 if we met A or T on the upper strand, $0'$ if we met them on the lower strand, 1 if we met C or G on the upper strand, and $1'$ if we met them on the lower strand. Doing this for all possible DNA molecules, we get... exactly the twin-shuffle language over 0, 1! According to Engelfriet and Rozenberg theorem, if we also apply a sequential transducer, then we obtain any computable language (and *all* of them, by varying the transducer). Otherwise stated, everything which can be computed is "codified" in the DNA molecule, what remains to do is to read the molecules as sketched above and then to rewrite these readings with a sequential transducer. A completely unexpected conclusion, difficult to be imagined (without having at hand the theoretical result from 1980).

In a DNA computing book written together with Rozenberg and Salomaa, published by Springer-Verlag in 1998, there are two results which complete the previous observation: on the one hand, we can consider only three nucleotide-symbols, $0, 0', 1$, with the last one being its own "complement" (nature uses to be redundant, nature also loves symmetries), and, on the other hand, we can also take into account the opposite orientation of the two strands, that is, we can read one strand in one direction and the second one in the opposite direction. In both cases, the result is the same, the characterization of the computing power of the Turing machine.

Later, Vincenzo Manca proved that, in a certain precise sense, the double stranded structure of the DNA molecule is *necessary* in order to obtain the computational universality.

We already have here a beautiful wonder – and a comforting example of using an old "purely theoretical" result in a new framework, much closer to the reality.

4 Splicing

Let us pass now to the splicing operation introduced by Tom Head (in 1987, hence, at the theoretical level, DNA computing started several years before the Adleman experiment, from 1994). Recombination, cut-and-paste, with a complete formal, syntactic formulation. If two restriction enzymes, each one with its own pattern, cut two molecules in such a way that the sticky ends they produce are identical, then the obtained fragments can be recombined, the prefix of the one molecule with the suffix of the other molecule. In this way, we pass from the two starting molecules to two new molecules.

The definition can be simplified, without losing too much from the biochemical significance. First, the Watson-Crick complementarity allows for passing from the double stranded molecules of symbols to usual strings. In this framework, each restriction enzyme recognizes a substring and cuts somewhere in its middle, in a specified position. Two enzymes which produce identical sticky ends lead to a *splicing rule*, which can be written as a quadruple of strings, with the first pair of strings indicating the pattern recognized by the first enzyme and the place where it cuts the string, and analogously the second pair of strings for the second enzyme.

This simplified version of the splicing operation was much investigated in DNA computing. In theory, as the starting point of defining a computing model called, in a paper written together with Rozenberg and Salomaa, *H system*, as a homage to Tom Head. In short, one gives a finite set of string-axioms and a set of splicing rules, one applies the rules to the axioms, then to the resulting strings, and so on, iteratively, possibly also selecting only certain strings, and in this way we generate a language. Like in a Chomsky grammar.

Two are the basic results of this area – and, between them the wonder!

On the one hand, we get a characterization of regular languages, those recognized by finite automata (the result got several proofs, the initial ones pretty complex), on the other hand, a characterization of the computing power of Turing machines (this result belongs to me and the proof, somewhat surprisingly, is simpler than the proofs of the first result mentioned above). The difference is that in the first result we use a finite set of splicing rules, in the second one we use an infinite set, but the set of rules is represented by a regular language. The universality can be obtained also for a finite set of rules, but with the use of rules controlled in various ways inspired by biology or by the formal language theory: promoters, inhibitors, using a priority relation, imposing a dependence on the sequence of rules used, and so on.

The technical details are not relevant, important is that we compute either at the level of one pole of computability, or at the level of the other pole. Everything or nothing, without equalizing any intermediate level – and there are several such levels, defined in terms of automata or grammars. It is worth noting that also in other frameworks, for instance, in MC, we obtain similar results. May we infer from these results that the intermediate levels are not *natural*? In a certain sense, this is the case, because, if we think to the classes of automata which characterize the context-free and the context-sensitive languages in Chomsky hierarchy, the push-down and the linear bounded automata, respectively, the former have a rather ad-hoc definition, the latter are inspired by complexity theory, the family of languages is defined by taking into consideration a property observed during the computation, not with respect to a static property, belonging to the automata architecture.

5 Computing in a natural way

The discussion can be generalized: what means to compute *in a natural way*? Which data structures to use and which operations on them? Theoretical computer science deals mainly with processing strings of symbols, especially using the *rewriting* operation, replacing a substring, short in comparison with the whole string which it is processed, with another string, also short. On the tape of automata of all types, from the Turing machine to the finite automaton, it is written a string of symbols. The Chomsky grammars, Lindenmayer systems, Marcus contextual grammars, Post systems, Markov algorithms, all these are processing strings. The same for Thue iterated morphisms, actually equivalent with Lindenmayer deterministic systems.

In comparison, what we find in biology? In the case of DNA, the double stranded molecule, and, as operations, the recombination, Tom Head's splicing, the separation of strands and their annealing, only rarely the point mutations, at the level of single nucleotides. Then, in the compartments of a cell, the *multiset*, the set with multiplicities associated with its elements. In biochemistry, like in chemistry, *the numbers matter.* While the DNA molecule can be considered a string, hence on the DNA string we have positional information, like in the case of numbers written in the Arabic style, in the aqueous solution from the cell compartments the numbers are expressed in unary. Base one is exponentially less efficient than base two – and still this was chosen by nature/biology. A biochemical reaction can be interpreted as a rewriting operation of a multiset: a sub-multiset is replaced by another multiset. However, in the functioning of a cell there appear many other operations, such that those of symport and antiport (I will return to them, as they deserve a more detailed discussion), or operations by which the membranes themselves evolve (division, dissolution, creation of membranes, exo- and endo-cytosis, and so on). Similarly in what concerns the functioning of neurons and of neural nets. I am careful not to say "the functioning of the brain", because, in spite of the fact that the neurologists know "everything" about the physiology of the brain, nobody can explain how the brain becomes *mind*, the organ where thinking, sensing, sentiments, conscience is (supposed to be) placed. Hypotheses, proposals, speculations there exist, significantly less certainty...

With all these data supports and operations with them one can define computing models, most of them equivalent in power with Turing machines, hence computationally complete, computing everything which can be computed (we take as valid the Turing-Church thesis). The proofs are often, if not always, constructive: we start from a Turing machine or from an equivalent model (normal forms for Chomsky grammars, classes of regulated grammars with context-free rules, register machines, other devices) and we construct an equivalent H system or, in MC, an equivalent P system. Starting the construction from a universal Turing machine (or an equivalent model), the biologically inspired computing model will implicitly be universal, hence programmable.

In theory, everything is wonderful. *In info.* At the practical level, there however appear two problems: the effective implementation, *in vitro*, of the theoretical model, and the question whether such a bio-computer can have also further motivations than proving that *it is possible* to compute in this way. Is such a computer useful, better than an existing electronic computer, at least from one point of view? Possible points of view are many. The possibility to solve problems of **NP** (or higher) complexity in a polynomial time is the first and the most attractive goal, but there are other ones related to the energy efficiency, the possibility to evolve in time, to learn, to self-repair, to deal with imprecise or incomplete data. All these are dreams for computer science, difficult to achieve, but they are common realities in the life of a cell, in biology in general.

Let us consider only one aspect, the massive parallelism, met ubiquitously in biology. The electronic engineers, even if they cannot equalize the biological parallelism, could put together a large number of processors, but, on the one hand, there would appear problems related to the heating of the computer, on the other hand, problems related to the synchronization, the coordination of the processors. It enters the stage the so-called *communication complexity*: the number of bits needed for coordinating the processors becomes comparable with the number of bits used in the computation itself. How the nature has solved these two problems we do not know precisely. Still less we know about how to imitate the nature's solutions.

6 Symport and antiport operations

Let us return to the *symport* and *antiport* operations. The membranes which provide the internal structure of the cell are constituted of so-called phospholipid molecules, which have a polarized "head" and a non-polarized "tail", consisting of two fatty acids, hydrophobic. Thrown in water, such molecules self organize spontaneously, forming a sphere with two layers, keeping the fatty acids separated from the water, opposing to the internal and the external water molecules the polarized heads. The common "enemy" and the electrical charges keep the molecules together, in a structure called *fluid-mosaic*, a model proposed by Singer and Nicolson, fully accepted only in 1972. The phospholipid molecules can move with respect to each other, but they do not allow to large molecules to pass between them from a side to the other side of the membrane; neither the ions can pass, because of their electrical charges. Nature has solved the problem of communication between the interior and the exterior of a vesicle of this type in a rather ingenious way: between the phospholipid molecules there are intercalated proteins which function as transmembrane channels – important to stress, of a *selective* type. Channels for water (the aquaporins of Gheorghe Benga, for which a Nobel Prize was awarded in 2003, but not to him...), natrium-potassium and natrium-calcium pumps, and so on and so forth.

Two special types of such channels are those which support the processes which the biologists call *symport* and *antiport*. In the first case, two molecules, let us identify them by M_1 and M_2, cannot pass separately through the symporter channel, but together they can do it, either entering the inner "reactor", or exiting it. In the second case, M_1 and M_2, placed in separated regions, one inside and the other one outside, cannot separately pass through the protein channel, but together they can do it, one in a direction and the other one in the opposite direction. The protein opens the channel until the molecules pass and then the channel is closed again.

In this manner, we obtain a way to "communicate" among the compartments of a cell – with the mentioning that I call *communication* the passage of objects (molecules, of any type) across a membrane. Starting from multisets of objects placed in the compartments of a cell and using given symport and antiport rules (pairs (M_1, M_2) associated with membranes, with specified directions to move each molecule), we can compute: we iteratively apply the rules, until reaching a configuration where no rule can be applied. (Because the number of objects cannot be modified by symport and antiport rules, we assume that the environment participates in the computation, as an exhaustible source of symbols.) Surprising at the first sight, but not so much at the second sight, we again obtain a characterization of the power of Turing machine.

Universal computation, by communication – here lies the surprise! We compute not by rewriting strings or multisets, but by moving objects across borders defined by means of membranes. Does it makes sense (to try) to construct a computer based on the symport and antiport operations? No, in what it concerns the computing power (Turing-Church Thesis), maybe, from the point of view of efficiency (provided that we will succeed to implement a significant level of parallelism), yes, from a point of view which is not so much taken into consideration by the current technology, although it refers to an important detail: the consumption of energy, heat dissipation. In computer science it is said that the dissipation of heat appears because of erasing. Rewriting means first erasing and after that writing. The operations of symport and antiport do not assume erasing, we only have moving, changing the place. Will these operations diminish/eliminate the energy loss during a computation? This is an appealing hypothesis – but the biologists warn us that, in order to function, many protein channels need chemical energy (obtained with the help of ATP, adenosine triphosphate, "the energy accumulator of the cell").

Why, at a closer examination, the universality of computing by means of symport/antiport is not so surprising? Theoretical computer science, old results in formal language theory, help us again. There are several characterizations of recursively enumerable languages of the following type: every language which can be recognized by a Turing machine can be obtained starting from a context-sensitive language (recognized by a linearly bounded automaton) and applying to it an operation which remove certain symbols (morphisms, left or right quotients, etc.). In short, context sensitiveness and erasing. Two features which the symport and antiport operations possess: the fact that two molecules evolve simultaneously means context sensitivity, while by "throwing" molecules in the environment or by collecting them in a corner of the cell and ignoring them there, we get the erasing. The only difficulty, for the theoretician, is the proof, the simulation of a computing mechanism equivalent with a Turing machine by means of certain symport and antiport operations associated with the compartments of a given arrangement of membranes. There are many proofs of this type in MC – that is, *in info*, but no *in vivo* or *in vitro* implementation.

7 Spiking models

Remaining close to biology, let us have a look to the way the neurons communicate to each other by means of electrical impulses of an identical shape, the *spikes*. Like any cell, a neuron has a body (soma), from which a "wire" starts, the axon, along which the spikes circulate. On the soma and in the end of the axon there are filaments, by the contact of which one obtains synapses, the links between neurons. We ignore here a lot of architecture and functioning details (some of them were already captured in MC models): the axon is covered by a protecting myelin layer, it is segmented by the so-called Ranvier nodes, a sort of relays amplifying the electrical impulses, the synapses correspond to symport/antiport operations, there also exists another class of cells, the astrocytes, which control/help the neurons activity, depending on the flow of spikes along their axons, and so on. We only keep in mind that we have only one type of "objects", the spike, and that the flow of spikes, their frequency, the distance in time between two consecutive spikes are very important. Of course, the functioning of a neuron, the emission of spikes depends on the neuron contents.

Abstracting all these details, one can define a computing model of the following type. Neurons are placed in the nodes of a graph whose arcs represent the axons/synapses. We start with a given number of spikes placed in each neuron. The neurons also contain *spiking rules*: depending on its contents, a neuron consumes a number of spikes and produces a number of spikes, which are sent to all neurons reached by a synapse which starts in the neuron where the rule was used. One obtains what is called in MC a *spiking neural P system*, in short, an SN P system. In the ten years since these devices were introduced, about 300 papers investigating them were published, from theory (computing power and efficiency) to applications (even in technological decision making, for instance, in a combination of these models with fuzzy reasoning models – the topic is mainly explored in China).

Like in many other places in bio-computability, also in this framework one obtains either a characterization of the power of finite automata, or of the power of Turing machines. Again, the intermediate levels of computability are not "natural".

Beyond the previous observation, two further reasons of wonder appear in this context.

The first one refers to the way the information is encoded/represented by means of spikes, in the distance between two consecutive electrical impulses. The time as a support of information... Similar to the Morse alphabet, but the goal there is only the transmission of information, here we deal with computations, even at the maximal level, that of Turing machines.

Can we use this observation for constructing computers? If data are encoded in time, using intervals, how will then function the classical time-space trade-off in this framework? Polynomial solutions to **NP**-complete problems are obtained, in theory and also in the DNA computing experiments and in MC theoretical approaches, by using an exponential workspace; in the spiking neurons case, the space and the time are "superposed", the time efficiency seems to be lost.

The second wonder reason is related to a result which looks unexpected at the first sight, namely, that there exist universal (in the universal Turing machine sense) SN P systems which have a relatively small number of neurons, at most of the order of hundreds. The result depends essentially on the type of spiking rules used in neurons and on the number of rules present in each neuron. In short, the number of neurons depends on their complexity, which is rather natural, but the fact that we can obtain a "programmable universal computer" with only about one hundred neurons is unexpected, at least with respect to the billions of neurons in the human brain and even with respect to the algorithmic computability. Clearly, the brain does not has as its goal (only) to compute at the algorithmic level (using the brain as a model for computers and using the computer metaphor in the study of the brain are useful strategies, but with soon to reach limits), but equalizing so easy the computability upper limit is not similarly easy to explain. Are the "neurons" of our models too complex, too powerful, or, on the other hand, the Turing computability is not too comprehensive? It is highly probably that both hypotheses are true. The second assumption is supported also by other characterizations of Turing computability in bio-inspired frameworks, using models which are simple "syntactically" or of small dimensions. The first hypothesis raises the question how to simplify the neurons in such a way not to lose the universality, even if this result is obtained by using a larger number of neurons. There are research efforts in MC following this idea.

8 Hypercomputability

While it is so easy, in DNA and cellular computing, to reach the level of algorithmic/Turing computability, it is much more difficult to pass beyond "the Turing barrier". This goal pertains to a branch of computability theory called *hypercomputability*, with publications, conferences, dedicated authors and also with skeptical or even acid comments. On the one hand, it is said/hoped that passing beyond "the Turing barrier" would have much more important practical consequences than a possible proof, even a constructive one, of the equality $\mathbf{P} = \mathbf{NP}$ (a result with a rather low plausibility). On the other hand, up to now, the ideas on which the models able of hypercomutability are based are not too many and not too realistic. In the domain literature there are about one dozen such ideas, but Martin Davis considers all of them *tricks*, supporting a *myth*.

Actually, Turing himself has proposed a version of his machine able of computing more than the Turing machine, by adding an *oracle*, which can answer (for free) questions, maybe questions which pass beyond the competence of the Turing machines. Several other ideas are based on squeezing infinity or real numbers in the construction or in the functioning of the machine. Because real numbers "are more than computable numbers" (more precisely: the set of computable numbers has a denumerable cardinality, while the cardinality of the set of real numbers is strictly larger), something uncomputable is introduced in advance in the model, hence there is no surprise to get a model which is more powerful than a usual Turing machine.

The same with the introduction of the infinity, even a denumerable one, with the following remark of interest for our discussion. Adleman experiment, of solving the problem whether a Hamiltonian path there exists in a graph, proceeds along the following main steps: one first generates all paths in the graph, then one eliminates all paths which do not pass through a number of nodes equal with the number of nodes in the graph (a selection according to the length of DNA molecules), then one removes the paths which are not passing through node 1, node 2, etc. What remains in the end are the DNA molecules which encode Hamiltonian paths. If no molecule survives, then this means that no Hamiltonian path exists. Also in other experiments one proceeds in a similar manner, that is, one starts with a large set, easy to be generated, and one removes iteratively elements which cannot be solutions. Otherwise stated, it is not constructed a solution, but one eliminates non-solutions. "Sculpturing!" Computing by carving! In terms of languages, we start from a general set of strings, maybe the total language over a given alphabet, and we eliminate repeatedly strings. *We remove the complement of the language we want to identify.* Not generating a language, as the grammars are doing, not accepting, as the automata are doing, but rejecting the strings which we do not want.

The family of Turing computable languages is not closed under the operation of taking the complement. Therefore, by "carving" we can "compute" languages which are not Turing computable. To this aim we need to eliminate an infinite set of strings. If in each step we remove a finite set of strings or even an infinite but regular one, in a finite number of steps we do not get out of the family of regular languages. Thus, either there is a step when we remove a complex language, or the "computation" lasts an infinite number of steps. Hypercomputability, but, agreeing with Martin Davis, the procedure does not look as an implementable (hyper)algorithm...

It is worth noting that the physicists are not refuting as totally infeasible the hypotheses on which the hypercomputability is grounded. Here is an SF scenario from which the physicists are not removing the S: let us assume that time has a 2-dimensional description, in spite of the fact that we, the humans, perceive only one dimension of it; let us assume that a human uses a computer which, after performing a number of steps along the dimension of the time which can be sensed by the user, starts to work on the orthogonal direction of time, performing an arbitrary number of steps, all of them invisible to the user, and then comes back and continues on the human visible dimension. Irrespective how many steps were made on the orthogonal direction of time, the user receives the result in a time which is essentially shorter for him.

We departed from biology, let us return to the cell. Nature creates membranes with two main goals: to localize the reactions and the reactants (enclosing them in "protected reactors"), and for creating reactors of a small size, where the reactants are sufficiently close to each other in order to collide and react, helped by the Brownian movement. Thus, smaller means faster. Let us generalize, exaggerating "a little bit", and assume that in a membrane placed inside another membrane the reactions are twice faster than the reactions in the membrane above. Let us create membranes inside membranes, repeatedly. (In MC there are rules of this type, for increasing the membrane architecture.) The reactions are accelerated from top down in an exponential way. This corresponds to the *accelerated automata*, already investigated in computer science since many years: the first step of a computation is performed in one time unit, the second one in half of a time unit, each step which follows in a time which is half of the time the previous step was performed. In this way, in two time units marked on a clock external with respect to the automaton, one performs an infinity of steps of the computation. The whole computation ends in at most two external time units. Accelerated automata of this type can solve problems beyond the power of Turing machines (such as, the halting problem, provided already by Turing as an example of a problem without an algorithmic solution). Exactly this can be done also by an accelerated P system – the result appears in a paper written together with Cristian Calude, now in New Zealand. Biological motivation, hypercomputability, but Martin Davis is again right: the membrane hierarchy should be arbitrarily deep in order to achieve an arbitrarily powerful acceleration...

9 Fypercomputability

9. Let us quit the hypercomputability and pass to... *fypercomputability*! This term occurred, as a game with letters-words, replacing the front *h* of hypercomputability with *f*, from *fast*. This is the main expectation from biocomputability, because it is one of the main "barriers" facing the "Turing-von Neumann" computers: the impossibility to solve **NP**-hard problems in a feasible (polynomial) time, not to speak also about **NP**-complete problems (a problem is **NP**-complete if each **NP**-hard problem can be reduced to it in a polynomial time; therefore, if an **NP**-complete problem could be solved in polynomial time, then all **NP**-hard problems would be solved in polynomial time, that is, $\mathbf{P} = \mathbf{NP}$).

However, the exponentials are not at all easy to handle... Here is a simple example, but with a strong didactic impact. Let us assume that we have a problem, for instance, dealing with graphs, of an exponential complexity, say, of the order of 3^n; let us assume that the existing computers can solve the problem in a reasonable time for graphs with 100 nodes. Assume now that the technology promises us or even offers us new computers which are 1000 times faster than the current ones. A spectacular progress, not at all easy to be obtained. A problem which is now solved in 1000 seconds will be solved with the new technology in one second. Good, but if now we can deal with graphs of 100 nodes, which will be the size of graphs which can be handled with the improved computers? The answer is disappointing – graphs with only 106–107 nodes – for the simple reason that 3^7 is greater than 1000. A great technological progress leads to a derisory advance in what concerns the size of the problems which can be solved. Not the technology is the way to cope with the exponential complexity, we need to look for something else, basically new. The parallelism is a path towards fypercomputability, and there are two main strategies: starting from the beginning of the computation with an exponential space (as in the Adleman experiment), or creating such an exponential space during the computation.

In MC, in most cases one follows the second strategy: using biological operations, such as the membrane division or the string replication, one creates an exponential workspace in a linear time, then, using this workspace, one solves in a polynomial time **NP**-complete problems. However, an exponential number of objects can also be obtained without involving the membrane division, but using only rules of the form $a \to aa$, repeatedly, in a parallel manner. Starting from one copy of a, in n steps we get 2^n copies of a. It is rather interesting to see that this manner of obtaining an exponential workspace is not sufficient in order to reach the desired efficiency: the so-called *Milano Theorem*, proved by Claudio Zandron in his PhD thesis (the first European PhD thesis in MC, presented in 2001; the first one in the world was presented one year before, in India), says that any P system, even using rules of the form of $a \to aa$, can be simulated in polynomial time by a Turing machine. Therefore, if such a system would solve **NP**-complete problems in polynomial time, then also the Turing machine would do it, which will imply that $\mathbf{P} = \mathbf{NP}$. Consequently (unless $\mathbf{P} = \mathbf{NP}$), the membrane division is necessary for fypercomputability.

Again, we have encountered here a subtle and surprising aspect. An exponential number of objects, placed in a single membrane, are not sufficient for exponentially speeding up the computation (in P systems), but if we separate these objects in different membranes, also exponentially many, the fypercomputation jump is obtained. The difference is made by the localization, the use of different rules in different membranes. In this way, there appears a parallelism of a more complex nature, with compartments which evolve in parallel, each of them processing, also in parallel, its own objects. I do not know whether this difference/phenomenon has appeared also in other frameworks – anyway not in the classic complexity theory, where the parallelism is not present, because the working model is the Turing machine, a sequential model.

10 Gaps in the computational complexity theory

Actually, the computational complexity theory, in its classic form, has several "gaps" from the point of view of bio-computability. I return to the way Adleman has solved the Hamiltonian path problem: he started from a given graph, hence from an instance of the problem, has constructed an ad-hoc "computer", using DNA molecules directly depending on the chosen graph, test tubes and other lab instruments, and he has found the solution (in a linear time, as the number of biochemical operations, some of them of a massive parallelism). In the complexity theory, such an approach is not allowed, it is required to start from the problem itself, and the algorithm/program should be written in a polynomial time, taking as parameters the size of the instances, not the instances themselves. In the general algorithm obtained in this way (one says that it is *uniform*) one introduces the instance to be solved. The idea is to prevent introducing the solution of an instance in the algorithm which pretends to solve the problem, but it simply provides the answer. Then, even for an uniform algorithm, one requests that the time of writing it is polynomial, so that it is not possible to work on solving the problem during the programming phase.

At the beginning of MC, a compromise between the two alternatives was chosen in the papers which proposed polynomial solutions to **NP**-complete problems: solutions which were not necessarily uniform, but at least "honest", starting from instances, but with the algorithm constructed in a polynomial time. They have been called *semi-uniform* solutions. And now the problems appear. How the complexity classes with respect to semi-uniform solutions look like and, more interesting, which are their relations with respect to the classic complexity classes? Of course, the semi-uniform classes are larger than the uniform ones. Are they strictly larger? Interesting enough, there were obtained results of both types, equality as well as strict inclusion, depending on the definition of the considered classes. I do not enter into further details, they become soon too technical.

Still further questions appear.

The brain (and the liver, and other organs and tissues) has a huge number of cells, not all of them working in each moment (at least this is what it was said some years ago). When a task appears, a problem for the brain, a food/beverage difficult to process for the liver, further cells are called to work. Can we use such a strategy in computer science, that is, to start from pre-computed resources, of an arbitrary size, without containing "too much" information, so that one cannot hide there a pre-computed solution of the problem, then to activate the "computer" according to the difficulty of the problem? This idea was followed in the framework of SN P systems, where the cell/neuron division does not seem to be biologically realistic, but it is natural to start from an arbitrarily large network of neurons, provided in advance, without containing spikes, and to introduce the problem to be solved, suitably encoded, in a small number of neurons, such that the network is activated as much as necessary, and it provides then the answer. Also in this area the theoretical developments are missing. When can we say that the initial resources are pre-computed in a honest way, they do not contain too much information? How the associated complexity classes can look like, which is their relation with the existing complexity classes?

Maybe even more interesting: can the internet be used as a pre-computed support for computations, of an arbitrary size? In a certain sense, internet has been already used in this way – it was the case of the SETI project, maybe of other projects, known or not.

Something more: the algorithms considered in the complexity theory are deterministic, what it is happening in a test tube or in a cell is far from that. However, the result is either "reliable", in the sense that it is obtained with a probability close to 1 (Adleman has used sufficiently many DNA molecules so that he was "sure" that all paths in the graph were constructed in the first phase of the experiment), either the computation, even if it is non-deterministic, is *confluent*, with two possibilities: *strongly confluent*, that is, after a non-deterministic phase, the computation converges to a unique configuration, and after that the continuation is deterministic, or *weakly/logically confluent*, that is, irrespective which was the followed path, the answer is the same. In MC, most of the solutions were initially semi-uniform and (strongly or weakly) confluent, after that for a while the solutions were uniform and confluent, while in the last time most of the considered solutions are uniform and deterministic. Once again, there appears here a challenge for the complexity theory, to clarify the relationships between complexity classes defined using non-deterministic solutions, confluent solutions (in the two versions), and the deterministic solutions.

11 Who is computing?

All these are challenges for the theory – also having some practical interpretations and consequences. However, let us refer also to some much more specific aspects, at least at the first sight closer to applicative computer science. Let us compare the computer programs, as they are usually conceived, as precise sequences of instructions, with a well specified order in which they are executed, maybe with the ordering controlled by using labels, *go to* instructions, conditional instructions, cycles, with the "programs" in a cell and the completely different way they are executed. In a cell, the molecules, present in the form of multisets, react/evolve by the means of *possible* reactions. Instead of a sequence of instructions, we have a set of reactions, a non-ordered set, without any organization, which are applied to "data" (to the molecules) in a concurrent manner, each possible reaction in competition with other possible reactions in finding molecules to process, with many reactions developing concomitantly if sufficient reactants are available and if the necessary conditions are met for the reactions to take place (temperature, salinity, acidity, the presence of certain catalysts and promoters, the absence of inhibitors, and so on). There appear probabilities, stoichiometric coefficients which predict-control the frequency of applying the reactions, depending on the population of reactants and the reaction conditions, there appears the dependence on promoters and inhibitors, but not the precise chain of instruction from a program in Algol-Fortran-Pascal-Basic or in any programming language closer to our days. In a large extent, the functioning of a cell is much more similar to the functioning of a Chomsky grammar than to the functioning of an automaton, of a Turing machine, where the states control the used instructions, maybe uniquely identifying them, in the case of deterministic automata.

Can we learn something at least interesting if not even useful for computability from these aspects?

Similarly in a tissue or in an organ, and still more in a colony of bacteria, there appears a functioning "in a community" which is not at all similar to the functioning of a set of processors put to work together. In biology, the parallelism is not total, the processes are not perfectly synchronized, in spite of the fact that there are rather precise "biological clocks", circadian cycles, annual seasons. For instance, the bacteriologists do not fully understand the way in which the bacteria realize the so-called *quorum sensing*, the "silent communication" before reaching a threshold after which the bacteria become aggressive. One discusses also in computer science about asynchronous computing, about amorphous computing, but nature is much ahead along these directions.

I end with another aspect of a practical interest: who is computing in Adleman experiment (and in many other similar experiments), who is the computer? The DNA molecules or... Adleman itself, the biochemist, the human? Without the human control, the molecules will not do what we expect from them to do, they will do almost nothing useful for us. It is true, a robotic system can replace the human operator. In this framework, "the computer" is a hybrid, and this is the plausible form of the computer of the future.

12 Other questions

Of course, there are many other wonders-questions which are worth mentioning. In what concerns the DNA, we have remained at the level of the primary structure, while neither for the protein channels we have mentioned the fact that *the shape* plays a fundamental role, the coupling between surfaces is crucial for the execution of the symport/antiport operations.

Actually, one can imagine computations based on the matching of certain shapes (*computing by shapes*), a sort of puzzle game with specific rules and restrictions, which is again universal and which can get (purely theoretical for the time being) "implementations" in terms of DNA biochemistry.

Similarly, at the level of the brain there are many things to be discussed. An old hypothesis claims that there are two parts of the brain, one which is conscious, controllable, and one which is sub/inconscious; the first one sends problems to the latter one, which proposes solutions, which are evaluated by the cortex and, if they are real solutions, then the problem is solved, otherwise the problem is pushed once again to the subconscious part – and the process is iterated. There is here a dialogue between a deterministic component and a non-deterministic one, again something new for computer science. Can such a process be modeled, for instance, in terms of SN P systems? Has computer science something to learn from this supposed functioning of the brain? (Let us remember that **NP** is the class of problems for which a solution proposed non-deterministically can be deterministically checked in polynomial time, without counting any time for the "guessing" of the solution, therefore a "bi-automaton, with one deterministic and one non-deterministic module", with the non-deterministic component working in no time, could be of a real practical interest.)

Then, concerning the learning: adaptation, evolution, learnability are common facts in biology, almost absent in the electronic technology, with the learning, at least, intimately present in the *neural computing*, a branch of bio-computing. Even if the type of learning here looks reductionistic, limited to the tuning of certain numerical coefficients (weights) on synapses linking "neurons" of a rather abstract form, this approach proves to be surprisingly efficient, having unexpectedly good results, comparable with the results which another area of bio-computing has, the evolutionary computing – without escaping the so-called *no free lunch theorems*. Recently, the neural computing has obtained a performance of a historical importance: one of the best GO players in the world was defeated, in March 2016, by *AlphaGO*, a..., a..., there is a problem here, as *AlphaGO* is not a program, it is not a computer (of the kind it was used twenty years ago in the game of chess, when Kasparov was defeated by *Deep Blue* computer). This time, billions of GO games among human players were stored on some thousands of computers, and a program based on neural computing has learned from these games and then from games played alone, the program against itself, so that, in the end, the South Korean GO player Lee Sedol, 9-dan, was defeated. The achievement belongs to a Google team, a detail of importance, because a routine to efficiently search through the thousand computers was also important.

The result is remarkable. For many years it was said that GO is the ultimate challenge for the artificial intelligence. Now, this frontier is already behind us. I confess that I have expected a bigger enthusiasm in the bio-computing community after this event.

Of course, now the challenge is to extend the strategy used by *AlphaGO* to other domains than GO, and I am sure that there are works in this direction.

I stop here, confident that the reader has his/her own wonders-questions in front of biology and in front of biology relations with other disciplines, from engineering to computer science. Anyway, the progresses in this area, of the collaboration of biology with computer science, should not be underestimated...

References

[1] **L.M. Adleman**, Molecular computation of solutions to combinatorial problems, *Science*, **226**, (1994), 1021-1024

[2] **B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter**, *Molecular Biology of the Cell*, 4th Edition, Garland Science, New York, 2002

[3] **P. Bottoni, G. Mauri, P. Mussio, and Gh. Păun**, Computing with shapes, *Journal of Visual Languages and Computing*, **12** (4), (2001), 601-626

[4] **C. Calude and Gh. Păun**, Bio-steps beyond Turing, *BioSystems*, **77**, (2004), 175-194

[5] **B.J. Copeland**, Hypercomputation, *Minds and Machines*, **12** (4), (2002), 461-502

[6] **M. Davis**, *The myth of hypercomputation*, In: Alan Turing: Life and Legacy of a Great Thinker Ed. by C. Teuscher, Springer-Verlag, Heidelberg, 2003, 195-211

[7] **J. Engelfriet and G. Rozenberg**, Fixed point languages, equality languages, and representations of recursively enumerable languages, *Journal of the ACM*, **27**, (1980), 499-518

[8] **J. Gruska**, *Quantum Computing*, McGraw-Hill, New York, 1999

[9] **T. Head**, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, **49**, (1987), 737-759

[10] **T. Head, Gh. Păun, and D. Pixton**, *Language theory and molecular genetics*, Chapter 7 in vol. 2 of Handbook of Formal Languages Ed. by G. Rozenberg and A. Salomaa, Springer-Verlag, Berlin, 1997, 295-360

[11] **A. M. Ionescu, Gh. Păun, and T. Yokomori**, Spiking neural P systems, *Fundamenta Informaticae*, **71**, (2006), 279-308

[12] **T.-O. Ishdorj and A. Leporati**, Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources, *Natural Computing*, **7** (4), (2008), 519-534

[13] **V. Manca**, On the logic and geometry of bilinear forms, *Fundamenta Informaticae*, **64**, (2005), 261-273

[14] **V. Manca**, *Infobiotics. Information in Biotic Systems*, Springer-Verlag, Berlin, 2013

[15] **S. Marcus**, Linguistic structures and genetic devices in molecular genetics, *Cahiers de Linguistique Théorique et Appliquée*, **11** (2), (1974), 77-104

[16] **L. Pan, T. Wu, and Z. Zhang**, A bibliography of spiking neural P systems, *Bulletin of IMCS*, **1**, (2016), 63-78

[17] **Ch. P. Papadimitriou**, *Computational Complexity*, Addison-Wesley, Reading, Mass., 1994

[18] **A. Păun and Gh. Păun**, The power of communication: P systems with symport/antiport, *New Generation Computing*, **20** (3), (2002), 295-305

[19] **A. Păun and Gh. Păun**, Small universal spiking neural P systems, *BioSystems*, **90**, (2007), 48-60

[20] **Gh. Păun**, On the splicing operation, *Discrete Applied Mathematics*, **70**, (1996), 57-79

[21] **Gh. Păun**, Regular extended H systems are computationally universal, *Journal of Automata, Languages, and Combinatorics*, **1** (1), (1996), 27-36

[22] **Gh. Păun**, Computing by carving, *Soft Computing*, **3** (1), (1999), 30-36

[23] **Gh. Păun**, Computing with membranes, *Journal of Computer and Systems Sciences*, **61** (1), (2000), 108-143, and Turku Centre for Computer Science-TUCS Report No. 208 (1998)

[24] **Gh. Păun**, P systems with active membranes: Attacking NP-complete problems, *Journal of Automata, Languages, and Combinatorics*, **6** (1), (2001), 75-90

[25] **Gh. Păun**, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002

[26] **Gh. Păun**, *Towards "fypercomputations" (in membrane computing)*, In: Languages Alive. Essays Dedicated to Jürgen Dassow on the Occasion of His 65 Birthday Ed. by H. Bordihn, M. Kutrib, and B. Truthe, LNCS 7300, Springer-Verlag, Berlin, 2012, 207-221

[27] **Gh. Păun**, *Looking for Computers in the Biological Cell. After Twenty Years (in Romanian)*, Reception Discourse in Romanian Academy, The Publ. House of the Romanian Academy, Bucharest, 2014

[28] **Gh. Păun, G. Rozenberg, and A. Salomaa**, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998

[29] **Gh. Păun, G. Rozenberg, and A. Salomaa, eds.**, *The Oxford Handbook of Membrane Computing*, Oxford Univ. Press, 2000

[30] **M. J. Pérez-Jiménez, A. Riscos-Núñez, Á. Romero-Jiménez, and D. Woods**, *Complexity – Membrane division, membrane creation*, Chapter 12 in The Oxford Handbook of Membrane Computing Ed. by Gh. Păun, G. Rozenberg, A. Salomaa, Oxford Univ. Press, 2010, 302-336

[31] **G. Rozenberg and A. Salomaa**, *Watson-Crick Complementarity, Universal Computations and Genetic Engineering*, Technical Report 96-28, Dept. of Computer Science, Leiden University, Oct. 1996

[32] **G. Rozenberg and A. Salomaa, eds.**, *Handbook of Formal Languages*, 3 vols., Springer-Verlag, Berlin, 1997

[33] **A. M. Turing**, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of London Mathematical Society ,Ser. 2*, **42**, (1936), 230-265; with a correction, **43** (1936), 544-546

[34] **J. D. Watson and F. H. Crick**, Molecular structure of nucleic acids, *Nature*, **4356**, (1953), 737-738

[35] **C. Zandron**, *A Model for Molecular Computing: Membrane Systems*, PhD Thesis, Universitá degli Studi di Milano, 2001

[36] **G. Zhang, J. Cheng, T. Wang, X. Wang, and J. Zhu**, *Membrane Computing. Theory and Applications (in Chinese)*, Science Press, Beijing, 2015

[37] The website of membrane computing: `http://ppage.psystems.eu.`

[38] The website of *Bulletin of IMCS* (International Membrane Computing Society): `http://membranecomputing.net/IMCSBulletin/.`

Gheorghe Păun
Romanian Academy, Bucharest, Romania
E-mail: curteadelaarges@gmail.com