Looking for Computers in the Biological Cell. After Twenty Years[*]

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
gpaun@us.es

1 Preliminary Cautious and Explanations

The previous title needs some explanations which I would like to bring from the very beginning.

On the one hand, it promises too much, at least with respect to my scientific preoccupations in the last two decades and with respect to the discussion which follows. It is true that there are attempts to use the cell as it is (bacteria, for instance) or parts of it (especially DNA molecules) to compute, but a research direction which looks more realistic, at least for a while, and which has interested me, is to look in the cell for *ideas* useful to computer science, for *computability models* which, passing from biological structures and processes to mathematical models, of a computational type, can not only ensure a better use of the existing computers, the electronic ones, but they can also return to the starting point, as tools for biological investigations.

Looking to the cell through the mathematician-computer scientist glasses, this is the short description of the present approach, and in this area it is placed the personal research experience which the present text is based on.

On the other hand, the title announces already the autobiographical intention. Because a Reception Speech is a synthesis moment, if not also a career summarizing moment, it cannot be less autobiographical than it is, one uses to say, any novel or poetry volume. And, let us not forget, the life in the *purity and signs world* (a syntagma of Dan Barbilian-Ion Barbu, a Romanian mathematician and poet)

[*] This is the English version of the Reception Speech I have delivered on October 24, 2014, at the Romanian Academy, Bucharest, and printed by the Publishing House of the Romanian Academy in December 2014. The answer to this speech was given by acad. Solomon Marcus. Some ideas and some paragraphs of the text have appeared, in a preliminary version, in the paper Gh. Păun "From cells to (silicon) computers, and back", published in the volume *New Computational Paradigms. Changing Conceptions of what is Computable* (B.S. Cooper, B. Lowe, A. Sorbi, eds.), Springer, New York, 2008, 343–371.

of mathematics assumes/imposes a great degree of loneliness, as acad. Solomon Marcus reminded us in his Reception Speech (2008), while the loneliness (it is supposed to) make(s) us wiser, but it also moves us farther from the ”world-as-it-is”, so that at some stage you no longer know how much from a mathematician belongs to the ”world” and how much belongs to mathematics. That is why we can consider that a mathematician is autobiographical both in his/her theorems and in the proofs of his/her theorems, as well as in the models (s)he proposes.

Looking back in time, I find that I am now at the end of two periods of two decades each, the second one completely devoted to ”searching computers in the cell”, while the first period was almost systematically devoted to preparing the tools needed/useful to this search. The present text describes mainly the latter of these two periods.

2 Another Possible Title

For a while, I had in mind also another title, much more general, namely, *From bioinformatics to infobiology.* It was at the same time a proposal and a forecast, and the pages which follow try to bring consistency to this forecast. Actually, the idea does not belong to me, in several places there were discussions about a new age of biology – the same was predicted also for physics – based on using the informational-computational paradigms, if not also based on further chapters of mathematics, not developed yet. The idea is not to apply computer science, be it theoretical or practical, to biology, but to pass to a higher level, to a systematic approach to biological phenomena in terms of computability, with the key role of information being understood. Attempts which illustrate this possibility, also advocating for its necessity, can be found in many places, going back in time to Erwin Schrödinger and John von Neumann. In a recent book, *Infobiotics. Information in Biotic Systems* (Springer-Verlag, 2013), Vincenzo Manca also pleads for ”a new biology”, which he calls *infobiotics*, starting from the observation that *the life is too important to be investigated only by biologists.* I would reformulate in more general terms: *the life is too important and too complex to be investigated only by the traditional biology* – with the important emphasis that exactly the biologists are called to not only benefit, but also to provide consistency to infobiology. Together with the computer scientists and, more plausibly and more efficient, borrowing from the computer scientists ideas, models, techniques, making them their own ideas, models and techniques and developing them. There is here also a plead for multi-trans-inter-disciplinarity (starting with the higher education), but also a warning: this is not only possible, but, it seems, this is also at the right time, on the verge to become urgent.

3 The Framework

Having in mind the title before and looking for an "official" enveloping area, the first syntagma which appears is *natural computing* – with the mentioning, however, that it covers a very large variety of research areas, including the bioinformatics and also moving towards infobiology. For an authoritative description, let us consider the *Handbook of Natural Computing*, edited by Grzegorz Rozenberg, Thomas Bäck and Joost N. Kok, published, in four large volumes, by Springer-Verlag, in 2012. From the beginning of the Preface, we learn that *Natural Computing is the field of research that investigates human-designed computing inspired by nature as well as computing taking place in nature, that is, it investigates models and computational techniques inspired by nature, and also it investigates, in terms of information processing, phenomena taking place in nature.* The generality is obvious, adding to the desire to identify in nature (important: not only in biology) ideas useful to computer science, a position which, as I have already said, although it is not completely new, if it is systematically applied, it can lead to a new paradigm in biological research and in other frameworks too: the informational approach, hence surpassing the traditional approach, the chemical-physical one.

The idea was formulated also in other contexts: the computational point of view (to the information processing one adds the essential aspect of computability) can also lead to a new physics – among others, this is the forecast of Jozef Gruska, an active promoter of quantum computing and a *pioneer of computer science*, rewarded with a diploma of this kind by the Computer Science Section of IEEE (let us remind the fact that also Grigore C. Moisil has been awarded such a diploma and title). On the same idea is grounded also the collective volume *A Computable Universe. Understanding and Exploring Nature as Computation*, edited by Hector Zenil and published by World Scientific in 2013. Many chapters have exciting-enthusiastic titles: *Life as Evolving Software, The Computable Universe Hypothesis, The Universe as Quantum Computer*, etc. There also is a chapter-long Preface, by sir Roger Penrose, not always fully agreeing with the hypotheses from the book.

Actually, also the *Handbook of Natural Computing* mentioned before includes the quantum computing among the covered domains. Here is its contents (the main sections, without specifying the chapters): *Cellular Automata, Neural Computation, Evolutionary Computation, Molecular Computation, Quantum Computation, Broader Perspective – Nature-Inspired Algorithms, Broader Perspective – Alternative Models of Computation.* There is some degree of "annexationism" here (for instance, cellular automata are not too much related to the biological cells), but let us mention that the section devoted to the molecular computation covers DNA computing, membrane computing, and gene assembly in ciliates, the former two areas being exactly what we are interested in here.

4 The Popularity of a Domain

Even remaining only at the editorial level and at the level of conferences (without considering also the research projects, hence the financial support), one can say that there is a real fashion of natural computing – more general, of unconventional computing, more restricted, of bioinformatics.

Here are only a few illustrations. Springer-Verlag has a separate series of books dedicated to natural computing monographs, named exactly in this way, it also has a journal, *Natural Computing.* There is an international conference, *Unconventional Computing*, which became, in the last year, slightly pleonastic, *International Conference on Unconventional Computation and Natural Computation. BIC-TA*, that is, *Bio-Inspired Computing – Theory and Applications*, is another conference of a real success, at least in what concerns the number of participants, a meeting whose format I has established, together with colleagues from Spain and China, in 2005, and which is organized since then each year, in China or in the neighboring countries – this can explain the massive participation, as the Chinese researchers are very active in this area.

We have reached the closest upper envelope of the area discussed here: the computability inspired from biology. It is important to note that the term "bioinformatics" (bio-computer science) has a double meaning, with, one can say, a geographical determination. In the "pragmatic West", it mainly covers the computer science applications to biology (in the "standard" scenario, one goes from problems towards tools, without too much theory). In Europe, both directions of influence are taken into consideration, from biology towards computer science and conversely. Although it is just natural that both these two research directions should be developed together, in collaboration, the reality is not always so. In search of solutions for current questions, some of them really urgent, for instance from the biomedical area, mathematics and computer science often provide tools prepared and developed in other areas. The typical example is that of differential equations, with a glorious history in physics, astronomy, mechanics, meteorology, and which are "borrowed" to biology, not always checking their adequacy. I will return to this issue, of a great importance for promoting new tools for biology.

"The European strategy", of constructing a mathematical theory which looks for applications after it is developed, has its appeal and advantages – but also its traps. Being an European, being a mathematician, I have been especially attracted by this strategy, but, in time, I became more and more interested by "reality", by applications.

5 What Means to Compute?

Let us come back to the title, with the fundamental question concerning the definition of the notions of computation and computer. This is a question of the same type as "what is mathematics?", with many different answers, none of them complete, none of them fully agreed. If information processing is a computation, then we can see computations everywhere. With a very important detail, hidden in the previous formulation: *we can see.* We, the human beings. Otherwise stated, an observer, which interprets a process as being a computation. I do not want to push the discussion as far as asking questions of the form "does a tree which falls in the water of a lake, in the middle of an uninhabited forest, produce any noise, taking into account that there is nobody there to hear it?" – I mention the fact that this question was the topic of a paper accepted some years ago by a conference on unconventional computing, that is why I recall it – and, on the other hand, I also do not want to involve God in this issue, the omnipresent, omniscient, omnipotent God, considered as an universal observer (at least, not for observing computations, maybe only for noticing noises in desert forests...).

A somewhat exaggerated but rather suggestive example is that of a drop of liquid which falls freely in the air. During its falling down, the drop instantaneously "solves" on its surface, by the form it takes, complex differential equations. Is this a computation? I would not go so far. Similarly with what happens continuously in the cells of a leaf or of the human body, at the biochemical or even at the informational level.

The idea of a computation as a process considered so by an observer is not at all new. One of the conclusions of the John Searle book *The Rediscovery of the Mind* (MIT Press, 1992), is exactly this – a computation is not an intrinsic property of a process, but it is *observer-relative.*

A very suggestive formulation of the role of the observer in considering a process as being a computation belongs to Tommaso Toffoli. The quotation which follows appears in a paper with a statement-title: "Nothing makes sense in computing except in the light of evolution" (*Journal of Unconventional Computing*, vol. 1, 2005, pages 3–29).

"We'we just seen that it is not useful to call *computation* just any nontrivial yet somewhat disciplined coupling between state variables. We also want this coupling to have been *intentionally* set up for the purpose of predicting or manipulating – in other words, for *knowing* or *doing* something. This is what shall distinguish bona–fide computation from other intriguing function–composition phenomena such as weather patterns or stock–exchange cycles. But now we have new questions, namely, 'Set up by whom or what?', 'What is it good for?', and 'How do we recognize intention?'

Far from me to want to sneak animistic, spiritualistic, or even simply anthropic considerations into the makeup of computation! *The concept of computation must emerge as a natural, well–characterized, objective construct, recognizable by and useful to humans, Martians and robots alike*" (my emphasis, Gh.P.).

Toffoli's questions should be remembered and discussed, but they move us far from our subject. Let us return to John Searle, namely, to a more technical reading of the idea of implying an observer in the definition of a computation. This was the approach of Matteo Cavaliere and Peter Leupold, both of them my students in the PhD school in Tarragona, Spain, the former one being my first PhD student there. They have published a series of papers with this subject, I cite here only a recent one, by Peter Leupold, "Is computation observer-relative?", presented at the *Sixth Workshop on Non-Classical Models of Automata and Applications*, Kassel, Germany, July 2014. Actually, in the Cavaliere-Leupold approach there appear two observers, one of them – we can call it observer of the first order – following a simple process and "translating" the steps of the process in an external language, and the second observer, closer to the Searle-Toffoli observer, interpreting as a computation the results of the activity of the first observer. Cavaliere and Leupold consider a series of process-observer (of the first order) pairs which, separately, have a reduced (computing) power, but which, together, lead to the computing power of Turing machines from the point of view of the external observer.

6 The Turing Machine

Let us start also from another direction, from the meaning given by mathematics to the notion of computation. Already from the thirties of the previous century we have a definition of what is computable, the answer Alan Turing gave to the question "what is mechanically computable?", formulated by David Hilbert at the beginning of the twentieth century. "Mechanically", i.e., "algorithmically" in our today reformulation. There were many proposed answers (I recall only the recursive functions and the lambda-calculus), given by great names of mathematics-computer science (I recall here only Alonzo Church, Stephen Kleene, Emil Post), but the solution given by Turing, what we call now *Turing machine*, has been accepted as the most convincing one (a fact certified even by the highly exigent Gödel). This is now in computer science the standard model of an algorithm (I have not said *definition*, because we have only an intuitive understanding of the idea of an *algorithm*, but we can say that in this way we have a definition of what is *computable*).

Without entering into details, I mention only that Hilbert's problem was more general. It started from the algorithmic resolution of diophantine equations, those with integer coefficients (the tenth problem in Hilbert's 1990 list), but in its later (in 1928) formulation Hilbert was saying that "the *Entscheidungsproblem* [the decision problem in the first order logic] would be solved if we would have a procedure which, for any logical expression we would decide through a finite number of operations whether it is satisfiable... *Entscheidungsproblem* should be considered the main problem of the mathematical logic". At this general level, Gödel theorems answer negatively Hilbert's program. Negative answers gave also Church and Turing, while Hilbert tenth problem was solved – also negatively – in 1970, by Yurii Matijasevich (after many efforts of several mathematicians: Julia Robinson, Hilary Putnam, Martin Davis). Turing not only gives a negative answer, moreover, he not only defines "the frontiers of computability", but he also produces an example of a problem placed behind these frontiers, a problem which is not algorithmically solvable, *the halting problem* (there is no algorithm, hence a Turing machine, which, taking as input an arbitrary Turing machine, can tell us, in a finite number of steps, whether the given input machine halts or not when starting from an arbitrarily given initial data). To the halting problems reduce, directly or indirectly, most if not all undecidability results obtained after that.

The Turing machine is so important for computer science, including the natural/unconventional computability, that it is worth discussing it a little bit more.

7 Some More Technical Details

It is interesting to note that when he defined his "machine", Turing explicitly started – he states this at the beginning of the paper – from the attempt to abstract the way a human being computes, reducing to the minimum the resources used and the operations made. In this way, in the end one obtains a "computer" which consists of a potentially infinite *tape*, bounded to the left, divided in *cells* where one can write *symbols* from a given finite *alphabet*; these symbols can be read and rewritten by a *read-write head*, which can "see" only one cell, can read the symbol written there, can change it, then it can move to the neighboring left or right cell or it can stay in the same place; the activity of the read-write head is controlled by the finitely many *states* of a *memory*. Thus, we get *instructions* of the form $s_1a \rightarrow s_2bD$ with the following meaning: in state s_1, with the head reading symbol a, the machine passes to state s_2, modifies a to b (in particular, a and b can be identical), and moves the read-write head as indicated by D. One starts with the tape empty, with the machine in a special initial state s_0; one writes the initial data on the tape (for instance, two numbers which have to be multiplied), one places the head on the first cell of the tape (the leftmost one), and one follows the instructions of the (e.g., multiplication) "program" until one reaches a *final state* and the machine *halts*, no further instruction can be applied. The contents of the tape at that moment is the result of the computation.

Extremely reductionistic, but this is the most general model of an algorithmic computation – because no previous definition of what is computable is known, this assertion is only a hypothesis, called the *Turing-Church thesis*. However, what made Turing machine so attractive were not only the *simplicity* of its definition and its *power* (it was proved that the Turing machine can simulate any other computing model), but also its *robustness* (the computing power is not changed if we add further ingredients to the architecture or to the functioning, such as further tapes, if we infinitely prolong the tape also to the left, if we consider non-deterministic computations, etc.), and, mainly, the existence of *universal Turing machines*: there exists a fixed Turing machine TMU which can simulate any particular Turing machine TU, in the following sense. If a code of the machine TM (let us denote it by $code(TM)$) is placed on the tape of TMU together with an input x of TU, then TMU will provide the same result as that provided by TU when starting from input x. A little bit more formally (but still omitting some details – e.g., codifications), we can write $TMU(code(TM), x) = TM(x)$. And Turing proved that there are universal Turing machines. This was done in 1936, in the paper "On computable numbers, with an application to the *Entscheidungsproblem*", published in *Proceedings of the London Mathematical Society*, Ser. 2, vol. 42, 1936, 230–265, with an erratum in vol. 43, 1936, pages 544–546.

This is the "birth certificate" of the today computers, consequently called of Turing-von Neumann type (in forties, when he has participate in the designing of the first programmable electronic computers, von Neumann was influenced by Turing ideas).

A couple of things deserve to be mentioned: the code of machine TM is the program to be executed/simulated on TMU, starting from the data x; the instructions of TMU form the "operating system" of our "computer"; the data and the programs are written in the same place, on the tape of the universal Turing machines (in the "computer memory") – from here it follows the possibility to process programs in the same way as we process data, hence the vulnerability of programs to computer viruses.

Several details are important from the point of view of natural computing. The work of the Turing machine is sequential, in each time unit one performs only one instruction. In many places in nature, if not in most of them, in particular, in biology, the processes develop in parallel, which is a very appealing feature for computer science, but these processes are not necessarily synchronized, which, in turn, raises difficulties for computer science.

There also are further differences between Turing machines, the "biological computers", and the electronic computers, but we will discuss these differences later.

For the time being we keep in mind that in what follows *to compute* has the meaning suggested by Turing machines: there are an input and an output, between them there is an algorithm which bridges inputs and outputs, and the result of a computation is obtained in the moment when the machine halts. Very restricted, but precise. With such a framework at hand, we can look around for computations, moreover, we can investigate them in a well developed context, the computability theory – actually, a set of several theories, such as automata theory, formal language (grammar) theory, complexity theory and others.

8 Computer Science and Mathematics

This is maybe the place to remind a debate which motivated many discussions and points of view, often biased, concerning the relation between computer science and mathematics. Discussions of this kind have appeared also in the Romanian Academy, they appeared in the higher education (in the sixties-seventies of the last century, at the time of sputniks and hydroelectrical plants, we had many faculties of "mathematics-mechanics", now mechanics was replaced by computer science), the issue is often debated in mass media. Actually, the context is larger, sometimes it is put in question the relation of mathematics with other sciences, with school education, with the society. There are persons who are proud of the fact that they "were not good in math". It was even expressed the opinion that mathematics is a luxury, a "national fetish" (this expression has recently appeared in the title of a Romanian newspaper article), in short, that one makes too much fuss of mathematics and one teaches too much mathematics. This opinion is getting more and more popular, supported also by the ubiquitous penetration of computers ("we no longer need to know the multiplication table, the computer knows it for us").

Of course, there is a problem with the mathematical education. *What, how much*, and, mainly, *how*? – and there also are further questions; we can find them, often also together with solutions, in the papers dedicated in the last years by professor Solomon Marcus to education. The problem cannot be solved from bottom up, the mathematicians involved in research and in higher education should consider it – this is, for instance, the opinion of Juraj Hromkovic, from ETH Zürich, formulated in an article published in the *Curtea de la Argeş* journal (`www.curteadelaarges.ro`, August 2014), based on the practical activity in this respect carried out in the institute where J. Hromkovic works (among others, this activity was materialized in mathematical school books of a new type). In general, the mathematicians should enter public debates and plead for their discipline, mainly they are guilty if the domain loses its popularity. It is true that for a mathematician mathematics is a great game, which, like any game, has an intrinsic rewarding, in the very development of the game, therefore it is natural that the interest for "popularization" is low among mathematicians, but the persons who are proud of their mathematical infirmity, be it real or only claimed, are always much more visible, more vocal, and the danger which comes from this is obvious.

Having in mind only the relation between mathematics and computer science, let us mention that the theoretical computer science, placed at the intersection of the two domains, is often considered by computer scientists as a part of mathematics, and by mathematicians as a part of computer science. Sometimes, theoretical computer science has problems even inside computer science – as it happens also with other theoretical branches of science with a strong practical dimension. Of course, all these are false problems by themselves, but they can have unpleasant practical consequences.

Being of the same opinion, I cite here an authoritative voice, that of Edsger W. Dijkstra, one of the classics of computer science, in fact, of the practical computer science: it is sufficient to remind that during sixties he has worked for implementing the Algol language in the Amsterdam Mathematical Center, and, furthermore, he was the promoter of structured programming, well-known among the software practitioners. (Maybe it is good to add here that the first four years after graduation I have intensively written computer programs, in Cobol and Fortran, realizing even the programs for computing the salaries of the workers in a large Bucharest factory – I remember, therefore, what practical computer science means...)

"The end of computer science?", asks Dijkstra, ironically-rhetorically, already in the title of a note published in *Communications of the ACM* (vol. 44, March 2001, page 92), which starts with the following phrase: "In academia, in industry, and in the commercial world, there is a widespread belief that computing science as such has been all but completed and that, consequently, computing has matured from a theoretical topic for the scientists to a practical issue for the engineers, the managers, and the entrepreneurs." Then, it adds: "This widespread belief, however, is only correct if we identify the goals of computer science with what has been accomplished and forget those goals that we failed to reach, even if they are too important to be ignored."

Much more explicit is Dijkstra in the speech he delivered in May 2000 at a symposium (*In Pursuit of Simplicity*) organized at the Austin-Texas University, on the occasion of his retirement. The title of the speech (published in *Information Processing Letters*, vol. 77, February 2011, pages 53–61) is relevant: "Under the spell of Leibniz's dream". I recall a couple of aphoristic phrases: "What is theoretically beautiful tends to be eminently useful." "In the design of sophisticated digital systems, elegance is not a dispensable luxury but a matter of life and death, being a major factor that decides between success and failure." "These days there is so much obsession with application that, if the University is not careful, external forces, which do make the distinction [between theory and practice], will drive the wedge between *theory* and *practice* and may try to banish the *theorists* to a ghetto of separate departments and separate buildings. A simple extrapolation will tell us that in due time the isolated practitioners will have little to apply; this is well-known, but has never prevented the financial mind from killing the goose that lays the golden eggs. The worst thing with institutes explicitly devoted to applied science is that they tend to become institutes of second-rate theory."

The plead to place us under the spell of Leibniz is obvious, because, Leibniz said it, "the symbols direct the reason", and, after having a language where "all reason truths will be reduced to a kind of calculus", "the errors will only be computation errors". (Leibniz program, continued and formulated in more precise terms by David Hilbert, cannot be realized, on the one hand, mathematics is too exact-rigorous while the reality is too complex and nuanced to can transform everything in formal computations, on the other hand, Gödel theorems proved that even the Hilbert program is not realizable.)

Of course, the mathematics-computer science relationship is much more complex, but we cannot explore it further here. I close the discussion returning to the starting point: the today computers, programmable, of Turing-von Neumann type, are born from the Turing universality theorem from 1936. It is interesting to note (and comfortable for Dijkstra position) that, by means of a vote through Internet, in 2013, looking for the most important scientific and technological British discovery, the first place was won, surprisingly for our pragmatic times, by the Turing machine and Turing universality theorem, which were placed ahead of the steam engine, the telephone, the cement, the carbon fiber and other similarly important things.

9 Does Nature Compute?

Having in mind the computability in the sense of Turing, the previous question becomes more restrictive, but the discussion above provides us the borderlines in between which we have to look for the answer: yes, nature computes at least at the level of... humans, and yes, nature computes whenever there is a process which can be interpreted as a computation by a suitable observer. Opinions which are placed closer to the former or the latter of these limits can be easily found, I cite here only one from the very permissive extreme, even passing over the borderline, because the observer is not mentioned anymore.

At the beginning of Chapter 2 ("Molecular Computation") of the collective volume *Non-Standard Computation* (T. Gramss, S. Bornholdt, M. Gross, M. Mitchel, Th. Pellizzari, eds., Wiley-VCH, Weinheim, 1998), M. Gross says: "Life is computation. Every single living cell reads information from a memory, rewrites it, receives data input (information about the state of its environment), processes the data and acts according to the results of all this computation. Globally, the zillions of cells populating the biosphere certainly perform more computation steps per unit of time than all man made computers put together."

In what follows, I adopt a more conservative and, at the same time, more productive position: bearing in mind the mathematical definition of computability, more precisely, the Turing approach, let us look around, especially in biology, in search of ideas, data structures, operations with them, ways to control the operations, "computer" architectures, which can suggest (1) new computability models, (2) ways to better use the existing computers, (3) possibilities of improving the existing computers at the hardware level, maybe even (4) new types of computers, based on biological materials. It should be noticed the increased ambition from a point to the next one. It is worth remembering that DNA computing started from the very beginning from the attempt to compute in a test tube, thus directly addressing the fourth goal in the list above.

We mainly had here in mind the goals of computer science, but the first objective also covers the second direction of research mentioned in the preface of the *Handbook of Natural Computing*, the investigation of processes taking place in nature in terms of computability, and this research direction should be explicitly and separately emphasized, especially for pointing to a "side effect" of this approach, namely the return to biology, delivering models useful to the biologist.

At this moment, DNA computing was not too much useful to practical computer science, it was useful to biology and much useful to nano-technology, suggesting new research questions. Membrane computing has significant applications to computer science and biology, with higher promises in the latter area, including biomedicine and ecology among the application directions.

A detail: "the goal of computer science" also covers the theoretical interest, which is not supposed to necessarily lead to applications, in the restricted meaning of the term. Let us think, for instance, to ciliates. In the division process, when passing from the micronuclear genes to the macronuclear genes, these unicellular beings complete complex operations of list processing, and they are doing this since millions of years, much before the computer scientists gave name and investigated these data structures. Of course, the ciliates are not thinking to computations when doing this, but we, the humans, can build beautiful theories starting from their activity, including computability models, sometimes equivalent in power with Turing machines. Detains and references can be found in the monograph *Computation in Living Cells. Gene Assembly in Ciliates* (Springer-Verlag, 2004), by A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, and G. Rozenberg.

10 An Eternal Dilemma

The previous discussion inevitably pushes us towards the long debate concerning the relation between invention and discovery. The bibliography is huge, I cite here only the book of acad. Solomon Marcus *Invention or Discovery*, Cartea Românească Publishing House, Bucharest, 1989 (in Romanian). How much is invention and how much is discovery in computer science – with particularization to natural computing? I do not try to provide an answer, there are as many answers as many view points, personal experiences, philosophical positions. The models we work with are of a mathematical nature, the Platonic point of view ensures us that everything is discovery, because mathematics itself is a revealed reality. Yes, but it is already agreed that notions, concepts, theories, and models are inventions, the theorems are discovered, the proofs are invented. We can continue the alternating sequence by adding that the applications are discovered. Therefore, the models are considered inventions.

However, I would like to introduce a nuance. The models are based on structures which already exists, but they have not received yet a name. Moreover, differently from a wall which can be discovered both by an archaeologist who knows what he is searching for, but also by a fruit trees farmer who digs the soil with other goals than finding the basement of an old church, a computability model can be "seen" in a cell only by a computer scientist who has already in mind computability models. For instance, the processes called by biologists symport and antiport exist, they function since long ages in their ingenious ways, but they *compute* only for a mathematician who is looking for a computing model based on passing "objects" from a cell compartment to another one. "Computing by communication" – I have searched for a while something like that, having the intuition that it exists, and I had the solution when a biologist (Ioan Ardelean) told me about symport and antiport operations. This was a model mostly discovered than invented. Actually, a discovery which was not done by bringing to light the discovered object, but by means of superposing the intuition of a model over a piece of reality. The imagined model, similar to other existing computability models, was actualized during the dialogue between reality and the formal framework. I can say that this is at the same time invention and discovery.

11 Another Endless Discussion

I am not continuing with other similarly delicate questions, always of interest in spite of any given answers. (For instance, providing us the opportunity to ask how much *art* and how much *science* is in computer science, Donald Knuth entitled an impressing editorial project, planned to have a dozen of volumes, *The Art of Programming.*) However, I touch here another very sensitive topic, with which I was confronted sometimes in the form of the newspaper question (but not completely nonsensical): ”During your research in the cell area, have you ever met God?” Of course, the expected answer is something different from ”yes” or ”no”, and similarly obvious is that, if we take the question seriously, we will get lost on the slippery sands of personal options, beliefs, metaphors.

If God is the order, the organization, the good and the beautiful, Spinoza's God, visible in the harmony of the Universe laws, as Einstein would say, then yes, I meet Him continuously, both in cells and outside them. Furthermore: in the title of a book originally published in 2009 by Simon & Schuster, and translated in Romanian in 2011, Mario Livio asks *Is God a Mathematician?* I answer in the style of Plato: no, God is not a mathematician, He is mathematics itself (the ”grammar of the world”) – hence, again, I meet Him every moment.

If, however, God is what the Book proposes to me, then I go in line with Galileo Galilei, who, in a letter sent to don Benedetto Castelli, on December 21, 1613, said (I recall it following Edmond Constantinescu, *God Does not Play Dice*, MajestiPress Publishing House, Arad, 2008; in Romanian): ”God has written two books, the Bible and the Book of Nature. The Bible is written in the language of men. The Book of Nature is written in the language of mathematics. That is why the language of the Bible is not suitable for speaking about nature. The two books must be studied independently from each other.” And Galileo added: the Book of Nature teaches us ”how the Sky/Heaven goes”, while the Bible teaches us ”how to go to the Sky/Heaven”.

After centuries of separation – mainly dogmatical, from both sides –, alternating with attempts, most of them pathetical, of reconciliation of science with religion, the words of Galileo can look too simple or opportunistic, but they cut in an efficient way a continuously regenerated Gordian knot. Let me mention also a more sophisticated, but somewhat symmetrical position, of Francis S. Collins, not only contemporary with us, but also connected to the topic of these pages, as he was the director of National Human Genome Research Institute, one of the leaders of the famous Human Genome Project. In 2006 he has published a book, *The Language of God. A Scientist Presents Evidence for Belief*, Simon & Schuster, translated in Romanian in 2009. The syntagma ”language of God” was used also by Bill Clinton, in 2001, when he has announced the completion of ”the most important, most wondrous map ever produced by humankind”, the map of the around three billions of ”letters” of the ”book of life”. Even if the title seems to suggest this, Collins is neither a creationist, nor an adept of the intelligent creation, but he is an ”evolutionary deist” and the conclusion of his book is that ”the God of Bible is also the God of the genome” (page 222 in the Romanian version), while ”science can be a form of religiosity” (page 240). This is a very comfortable positioning, but, in what follows, I remain near Galileo.

12 The Limits of Today Computers

The fashion of natural computing and especially of the computing inspired from biology does not have only the internal motivation, of the numerous research directions explored in the last decades and proved to be theoretically interesting and at least promising if not directly useful in practice, but it has also an external motivation, related to the limits of the current computers, some of them rather visible. Indeed, the computers are the twentieth century invention with the widest impact, with implications in all components of our life, from communication to the functioning of the financial system, from the health system to the army, from the numerous gadgets around us to Internet. In spite of all these – actually, just because of that – the computers which we have now have limits which we reach often (with the mentioning that also here, like in most things, there is something bad in the good and something good in the bad: powerful computers can be used both in positive ways, but also for bad goals, such as breaking security systems and cryptographic protocols on which, for instance, the protected communication is based.) Let us however think positively and note that there are many tasks which the today computers cannot carry out, but which we would like to have performed.

The processors become continuously faster and more compact, the memory storage larger and larger. Sure, but how much this tendency will last? It was much invoked the so-called *Moore law*, stated in 1965 by Gordon A. Moore, co-founder of Intel Corporation, with respect to the number of transistors which can be placed on an integrated circuit, extended then to the cost of information unit stored, formulated sometimes even in the form ”in each year, the computers become two times smaller, two times faster, and two times more powerful”. Exponential in all the three directions, thus tending fast towards the quantum limit in the dimension of processors. Even at the more technical level, confirmed for a couple of decades, of doubling the capacity of processors, the law – actually, only an observation, followed by a forecast – has been adjusted several times, with the doubling/halving moved first at one year and a half, then at two years, then at three years. Still, it is not too bad, but one cannot continue too much even at this pace.

In fact, the real problem is a different one. Progresses are made continuously at the technological level, but the current computers have intrinsic limits, which cannot be overcome only by means of technological advances. The computer recognizes fingerprints, but not human faces, it plays chess at the level of the world champion, but (on the standard board, not on reduced boards) it plays GO only at the level of a beginner, it proves propositional calculus theorems, but cannot go over this level (and definitely cannot distinguish trivial and non-interesting theorems from theorems which deserve to be collected). All these and many more, mainly because these computers are... of Turing-von Neumann type. That is, sequential. Uniprocessor. (It also has other weaknesses, less restrictive in the current applications – for instance, it is a considerable energy consumer.) It computes whatever can be computed, but this is true in principle, at the *competence* level. There is here also a historical aspect. In the beginning, we were interested in what it can be computed, in the frontiers of computability, of algorithmic decidability. All these are important mathematical questions, but in applications it is of a direct relevance the *performance*, the resources needed for a given computation, what we can compute now and here, in specified conditions. How much electricity consumes a computer and how much space it needs are no longer questions of current interest, as they were in sixties (and still are in special frameworks, such as in cosmos and robotics), but the time we have to wait before receiving the answer to a given problem or the result of a computation is a crucial aspect in any application. And, I already mentioned it, in this respect not the technological promises are crucial, but the mathematical limits, the borderline between feasible and non-feasible.

13 A Great Challenge: the Exponential Complexity

A powerful theory was developed dealing with this subject, the computational complexity theory. Since the very beginning, it has defined as tractable the problems which can be solved in a polynomial time with respect to the size of the problem. (An example: consider a graph – a map with localities and roads among them – with n nodes. Which is the time necessary for an algorithm to tell us whether or not the graph contains a Hamiltonian path, i.e., a path which visits all nodes, passing only once through each of them? If this time is bounded by a polynomial in n, then we say that the algorithm is of a polynomial complexity.) The problems of an exponential complexity, those which need a time of the type $2^n, 3^n$, etc. for an input of size n were considered intractable. The former class was denoted by **P**, the latter one with **NP**, with the abbreviations coming from "polynomial" and "non-deterministically polynomial", respectively: a problem belongs to **NP** if we can decide in a polynomial time whether a proposed solution for it is indeed a solution or not (otherwise stated, we "guess" a solution, then we check whether it is correct; more technically, the solution is found by a non-deterministic Turing machine, one which has several possible transitions at a computation step and we rely on the fact that it always chooses the right continuation, without exhaustively checking all possibilities). For precise details the reader can consult the monograph of C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

Let us recall that in the class **NP** there is a subclass, of "the most difficult problems in **NP**", the **NP**-*complete* problems: a problem is of this type if any other problem in **NP** can be reduced to it in a polynomial time. Consequently, if an **NP**-complete problem could be solved in a polynomial time, then *all* problems in **NP** could be solved in a polynomial time. The problems used in cryptography are in most cases **NP**-complete.

A beautiful theory, which, however, in its basic version has three weaknesses: (1) it cannot tell us yet whether or not **P** = **NP**, whether or not polynomial solutions can be found also for the problems which are now supposed to be of an exponential complexity, (2) the theory does not take into account such "details" as the coefficients and the degree of the polynomials and which, at the practical level, can have a crucial influence on the computation time, and (3) the theory takes into consideration the extreme cases, it is of the *worst case* type, it counts the steps of computations which solve the most difficult instances of a problem, while the reality is placed in most cases in the middle, near the "average". Here is an example with a practical relevance: the linear programming problem is in **P**, because the ellipsoid algorithm of Leonid Khachiyan (1979) solves the problem in a polynomial time, but this algorithm is so complex that practically, in most cases, it is less efficient than the old simplex algorithm, proposed during the Second World War, considered one of the most important ten algorithms ever imagined, but which is, theoretically, of an exponential complexity.

For these reasons, the complexity theory was refined and diversified (average complexity, approximate algorithms – these algorithms have a direct connection with natural computing), while the definition of tractability was carefully redefined.

Anyway, the general feeling was transformed in a slogan: *the Turing-von Neumann computers cannot solve in a reasonable time problems of an exponential complexity.*

The interest for the **P** = **NP** problem is enormous. On the one hand, most of the (no-trivial) practical problems are in the class **NP** and are not known to be in **P**, hence they are (considered) intractable, cannot be efficiently solved, on the other hand, most of the cryptographic systems in use are based on problems of an exponential complexity, hence solving them in a polynomial time would lead to breaking these systems. The problem whether **P** is or not equal to **NP** was already formulated in 1971 (by Stephen Cook), and in the year 2000 it was included by Clay Mathematical Institute, Cambridge, Massachusetts, in the list of the seven "millennium problems", with a prize of one million dollars for a solution.

While the importance of this problem for the theoretical computer science cannot be overestimated, it is not clear which would be the practical consequences of a solution, whichever this will be. There were many discussions on this topic – see, for instance, S. Cook, "The importance of the **P** versus **NP** question", *Journal of the ACM*, vol. 50, 2003, pages 27–29. If a proof of the strict inclusion of **P** in **NP** will be obtained, as most computer scientists (but not all of them!) believe, then almost nothing will be changed at the level of the practical computer science. If the equality will be proved in a non-constructive manner, or the proof will be a constructive one, but in a non-feasible manner (polynomial solutions to problems in **NP** will be found, but with polynomials of very high degrees or with very large coefficients), then the practical consequences will not be significant (but a race will start for ad-hoc solutions, having the time estimated by polynomials with reasonable degrees and coefficients). If, however, a "cheap" passage from **NP** to **P** would be found, then the consequences for the practical computer science will be spectacular – in the good sense, excepting the cryptography, where the consequences will be dramatic.

At the level of the software there is one further problem, which I recall here in the formulation of Edsger W. Dijkstra (from "The end of computer science?", the above mentioned paper): "Most of our systems are much more complicated than can be considered healthy, and are too messy and chaotic to be used in comfort and confidence. The average customer of the computing industry has been served so poorly that he expects his system to crash all the time." The lack of robustness of the complex software systems is today a concern of the same interest as it was in the year 2000.

In order to illustrate the fact that not by means of technological progresses one can face the exponential complexity, let us examine a simple case: let us consider a problem of exponential complexity of the range of 2^n, for instance, a graph problem, which can be solved on a usual computer, say, for graphs with 500 nodes, in approximately one quarter of hour; let us suppose that the technology provides us a computer which is 1000 times faster than the ones we have, which is a totally nontrivial advance, not very frequently met. Using the new computer, we will solve the same problems as before in about one second (around 15 minutes means approximately 1000 seconds), but if we try to address the same problem for graphs with more than 500 nodes, the progresses are negligible: with the new computer we will solve in a quarter of hour only problems for graphs with at most 510 nodes. The simple reason for that is the fact that 2^{10} is already bigger than 1000. If the problem were of complexity 3^n, then we will stop around 506 nodes...

14 Promises of Natural Computing

In order to cope with the exponential complexity, but also for other reasons which I will mention later, computer science has imagined several research directions, most of them also related to the natural computing, even to bio-inspired computing: (1) looking for massively parallel computers, (2) looking for non-deterministic computers/computations, (3) looking for approximate/probabilistic solutions to computationally hard problems.

All these three research directions were explored already in the framework of the "standard" computer science, both at the theoretical level and at the technology level, the electronic one. Multiprocessor computers are available since several years – but without reaching the massive parallelism which is supposed to solve complex problems. If a large number of processors are put together, there appear other problems, some of them technological (e.g., high temperature dissipation), others, maybe more important, theoretical, concerning the synchronization of the processors. A distinct research area deals with the synchronization complexity – see, for instance, Juraj Hromkovic monograph *Communication Complexity and Parallel Computing*, Springer-Verlag, 1997. One of the conclusions of this theory says that, for a large number of processors, the synchronization cost (measured by the number of bits necessary to this aim) becomes larger than the cost of the computation itself, which suggests to get rid of synchronization, but then other problems appear, as we are not accustomed to use asynchronous computers.

Even less used we are to construct and utilize "non-deterministic computers". In exchange, the last of the three ideas mentioned above is rather attractive, and in this respect of a great help is the "brute force" of existing computers. The approach is useful especially in addressing complex optimization problems: exploring randomly the candidate solutions space, for a large enough time, with a sufficiently high probability we will reach optimal or nearly optimal solutions. Approximate solutions, possibly found with a known probability of being optimal.

Here it enters the stage, with great promises, the natural computing. From now on I will only refer to the one having a biological inspiration.

In a cell, a huge number of "chemical objects" (ions, simple molecules, macromolecules, DNA and RNA molecules, proteins) evolve together, in an aqueous solutions, at a high degree of parallelism, and, at the same time, of non-determinism, in a robust manner, controlled in an intricate way, successfully facing the influences coming from the environment, and getting in time very attractive characteristics, such as adaptation, learning, self-healing, reproduction. Many other details are of interest, such as the reversibility of certain processes or the energy efficiency, with the number of operations per Joule much bigger than in the case of the electronic processing of information (erasing consumes energy, that is why the reversible computers are of interest; see, e.g., R. Landauer, "Irreversibility and heat generation in the computing process", *IBM Journal of Research and Development*, vol. 5, 1961, pages 183–191, and C.H. Bennett, "Logical reversibility of computation", *Idem*, vol. 17, 1973, pages 525–532).

It seems, therefore, that during millions of years of evolution nature has polished many processes (and material supports for them) which wait to be identified and understood by the computer scientists, in order to learn new computability methods and paradigms, maybe for constructing computers of a new kind. And, the computer scientists have started to work singe a long time...

Here are a few steps on this road, very shortly: *Genetic algorithms*, as a way to organize the search through the space of candidate solutions, imitating the Darwinian evolution, in order to solve optimization problems. Generalization to *evolutionary computing* and *evolutionary programming. Neural networks*, trying to imitate the functioning of the human brain, also used for finding approximate solutions, especially for pattern recognition problems. A little bit later, *DNA computing*, which has proposed a new hardware, massively parallel, based on using the DNA molecules as a support for computations. Even younger, *membrane computing*, taking as the starting point the biological cell itself and cell populations.

In turn, the evolutionary computing, in general, the area of approximative algorithms inspired from biology, is spectacularly ramified, in the most diverse (in certain cases, also picturesque) directions: *immune computing, ant colony algorithm, bee colony algorithm, swarm computing, water flowing computing, cultural* *algorithm, cuckoo algorithm, strawberry algorithm* – and it is highly probable that in the meantime further algorithms have been proposed...

It is important to note that all the above mentioned branches of natural computing, with the exception of DNA computing, are meant to be implemented on the usual computer, in the aim of having a better use of it; one proposes new types of software/algorithms, not to change the computers architecture or new types of hardware.

15 Everything Goes Back to Turing

In a certain sense and in a certain extent, the whole history of theoretical computer science is related to biology, it has searched and has found inspiration in biology. I have already mentioned that, in 1935-1936, when he has defined the machine which bears now his name, Turing tried to imitate the way the humans are computing.

After one decade, McCullock, Pitts, Kleene have founded the theory of finite automata starting from the modeling of neurons and of neural networks. Later, the same starting point led to what is called today neural computing.

It is interesting to note that the beginnings of this research area can be identified in unpublished papers of the same Allan Turing. We have here an interesting case which can illustrate the influence of psychology and sociology on the development of science, telling about uninspired group leaders and about researchers interested more in their research than in the publication of the obtained results. Specifically, in 1948, Turing has written a short paper, called "Intelligent machinery", which has remained unpublished until 1968, because his boss from the London National Physical Laboratory, ironically, named sir Charles Darwin, the grandson of the famous biologist with the same name, has written on the corner of the first page of the paper "schoolboy essay", thus preventing the publication.

"In reality, this farsighted paper was a manifesto of the field of artificial intelligence. In the work (...) the British mathematician not only set out the fundamentals of connectionism but also brilliantly introduced many of the concepts that were later to become central to AI, in some cases after reinvention by others." – I have cited from B.J. Copeland, D. Proudfoot, "Alan Turing's forgotten ideas in computer science", *Scientific American*, April 1999, pages 77–81. Among others, Turing paper introduces two types of "neural networks", with the neurons randomly connected. This was proposed as a first step towards an intelligent machine, one of the key features of these networks being that of learning, of getting trained for solving problems. This is neural computing *avant la lettre*, with the main ideas rediscovered later, without referring to Turing. Details about Turing "unorganized machines" can also be found in C. Teuscher, ed., *Alan Turing. Life and Legacy of a Great Thinker*, Springer-Verlag, 2003, and in C. Teuscher, E. Sánchez, "A revival of Turing's forgotten connectionist ideas: exploring unorganized machines", from *Proc. Connectionist Models of Learning, Development and Evolution Conf.*, Liége, Belgium, 2000 (R.M. French, J.J. Sougne, eds.), Springer-Verlag, 2001, pages

153–162. Furthermore, at the address `http://www.AlanTuring.net` one can find details about Turing unpublished manuscripts and about the recent efforts to reintroduce them in circulation.

The same Turing, in the same year 1948, has proposed the "genetic or evolutionary search", the first ideas of the evolutionary computing developed later, a domain which contains now several powerful branches, (re)launched during the years: evolutionary programming (L.J. Fogel, A.J. Owens, M.J. Walsh), genetic algorithms (J.H. Holland), evolutionary strategies (I. Rechenberg, H.P. Schwefel), all three initiated in the sixties, genetic programming (J.R. Koza, the years 1990). The first experiment of computer "optimization through evolution and recombination" was carried out in 1962, by Bremermann. Details can be found in A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*, Springer-Verlag, 2003.

It would not be completely surprisingly if among Turing manuscripts we would discover also ideas related to DNA computing – let us remember that Turing died in June 1654, and the *Nature* paper where J.D. Watson and F.H.C. Crick described the double helix structure of the DNA molecule was published one year before ("A structure for deoxyribose nucleic acid", vol. 171, April 25, pages 737–738).

It is worth mentioning that two other concepts with a high career in computer science come from Turing, thus supporting the assertion that "everything starts with Turing". First, Turing himself raised the question whether or not one can compute... more than the Turing machine, imagining Turing machines with oracles, which is a much investigated topic in the current computer science. Then, Turing can be considered not only a founder of artificial intelligence, but also a forerunner of what is called now *artificial life*: in the last years of his life, Turing was interested in morphogenesis, in modeling the evolution from the genes of a fertilized egg to the structure of the resulting animal.

16 An Encouraging Example: The Genetic Algorithms

Before passing to the DNA and membrane computing, topics which I will describe in more details, let us spend some time discussing a branch of natural computing inspired from biology which is, at the first sight, surprisingly efficient. This is the *genetic algorithms* area, used for solving complex optimization problems for which there do not exist deterministic optimal algorithms or these algorithms are not efficient. The implicit slogan can look confusing: *if you do not know where to go, then go randomly* – with the mentioning that the "randomness" here is directed, the "random walk" is done "like in nature, in species evolution".

Everything is a metaphorical imitation of some elements from the Darwinian evolution. Let us assume we have a two variables function (we can suggestively represent it as a ground surface, with valleys and hills) for which we need to find the maximum (one of them, if there are several). If we cannot analytically address the problem, then we can choose to walk randomly through the definition domain, looking for the greatest value of the function. To this aim, we represent the domain points as "chromosomes", binary strings of a constant length, we choose (randomly or through other methods) a given number of starting points, and we compute the function value for all of them. Then we pass to "evolution": we take two by two the "chromosomes" and we recombine them (by crossover), that is, we cut them at a specified position and then we recombine the fragments, the prefix of one "chromosome" with the suffix of the other one and conversely. In this way, we obtain two new "chromosomes", describing two new individuals of the next "generation". We repeat this procedure for a specified number of times, we select the best solution obtained so far, and we stop.

Nothing guarantees that in this way we reach the solution of the problem, that, for instance, we do not get stuck in a local maximum, without being able to escape, but, and this is the (pleasant) surprise, in a large number of practical applications, this strategy works. Sure, there are a lot of variations of the previous scenario, it is even said that the monographs in this area are a sort of "cooking books", collections of recipes, lists of ingredients and suggestions of improvements of the algorithms: besides recombination, similar to the biological evolution, one also uses the local mutation operation, the passing from a generation to another one can be done in many ways, the "chromosomes" population can be distributed, we can evolve it locally, communicating in a way or another among regions, there are several halting criteria, and so on and so forth.

We have here at work the brute force of the computers and the evolutionary metaphor – with results, I repeat and stress it, unexpectedly good: non-intuitive solutions, rapid initial convergence, in many cases succeeding to avoid local maxima. The only "explanation" for these good results is the "bio-mystical" one: genetic algorithms are so good because they involve ingredients which nature has polished for many millions of years in the species evolution.

All these induce, at the same speculative level, a rather optimistic conclusion: if the genetic algorithms are so useful, in spite of the lack of any mathematical argument for their usefulness, let us try to imitate biology also in other aspects, with a great probability that, if we are similarly inspired to extract the right ideas, to obtain other fruitful suggestions for improving the use of the existing computers and, maybe, for imagining computers of other kinds, more efficient.

However, this optimism should be cooled down by the observation that a famous result in the area of evolutionary computing, in the area of approximate optimization algorithms in general, is the so-called *no free lunch theorem* of David Wolpert and William Macready (1997), which, informally, says that any two methods of approximate optimization are equally good, in average, over all optimization problems. "Equally good" can be also read "equally bad", for each method there are problems for which the method does not provide satisfactory solutions.

17 A Coincidence

Before passing to DNA computing, an autobiographical intermezzo. In April 1994 I was in Graz, Austria, attending a conference, and there I got a copy of the paper by Tom Head, from State University of New York at Binghamton, USA, soon after that a friend and collaborator, "Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors", published in *Bulletin of Mathematical Biology*, vol. 49, 1987, pages 737–759. It was a revelation. I was then after twenty years of formal language theory research and I immediately felt that it is open there a large area of application of what I have done before. It is true, I should had a similar revelation earlier, namely when I have read professor Solomon Marcus paper "Linguistic structures and generative devices in molecular genetics", from *Cahiers de Linguistique Thèorique et Appliquée*, vol. 11, 1974, pages 77–104, but probably it was too early, at that time I had not passed yet through the twenty years of preparation for natural computing which will be shortly described in a forthcoming section. In his paper, Tom Head introduces a formal operation with strings which formalizes the operation of recombination of DNA molecules. He calls it *splicing*, and I will call it in the same way, thus distinguishing it from the recombination operation from the genetic algorithms. The two operations are related, but they are not identical. Still being in Graz, I have imagined a sort of grammar based on the splicing operation, in fact, a variant simpler than that of Tom Head and closer to the string operations in language theory. The paper emerged in this way was published in *Discrete Applied Mathematics* and it consecrated the splicing version I have proposed. After a few weeks, I was in Leiden, The Netherlands, where I have written a paper together with Grzegorz Rozenberg and Arto Salomaa, the latter one from Turku, Finland, the place where I have spent after that several years, initially devoted to DNA computing and then to membrane computing. As usually well inspired, G. Rozenberg gave to our paper the title "Computing by splicing". Because, starting then, we have named H systems the computing devices based on splicing, thus reminding the name of the one who has introduced (invented of discovered?...) the respective operation, we have sent the paper, in manuscript, to Tom Head. He has immediately replied, by phone, asking us rather excited: have you known that right now it was carried out a successful experiment of computing with DNA?! No, we did not know – this was only a coincidence, which I place in the category of significant coincidences.

18 The First Computation in a Test Tube

Tom Head was talking about Leonard Adleman experiment, published in the autumn of 1994 in *Science*, nr. 226, November 1994, pages 1021–1024: "Molecular computation of solutions to combinatorial problems". Speculations about the possibility of using DNA molecules for computing were made already in seventies of the last century (Ch. Bennett, M. Conrad, even R. Feynman, with his much invoked phrase "there is plenty of space at the bottom", referring to physics but also extended to biology). Adleman has confirmed these expectations, solving in a laboratory the problem whether a Hamiltonian path exists or not in a given graph (I have mentioned it in a previous section). The problem is known to be **NP**-complete, hence among the most difficult intractable problems, of an exponential complexity (we assume that **P** is not equal to **NP**), but Adleman solved it in a number of steps which is linear with respect to the size of the graph. It is true, these steps are biochemical operations, performed by making use of a massive parallelism, even of non-determinism, all these made possible by the characteristics of the DNA molecules and the related biochemistry.

In short, millions of copies of one stranded sequences of nucleotides, codifying the nodes and the edges of the graph, were placed in an aqueous solution. Then, by decreasing the temperature of the solution, these sequences annealed, forming double stranded molecules, corresponding to the paths in the graph. Because there were used sufficiently many copies of the initial sequences, with a high probability we obtain in this way *all* paths in the graph. From them, the paths were selected which pass through all nodes, and this was done by usual laboratory procedures: gel electrophoresis for separating the molecules according to their length, then selection through denaturation and amplification by PCR of the paths passing through all nodes (hence Hamiltonian).

This procedure assumes a number of biochemical operations which is linear with respect to the number of the nodes in the graph. The problem is **NP**-complete, hence this is an extraordinary achievement – and the consequences were accordingly sound. Already in the next year, 1995, it was organized in Princeton a meeting with the title "DNA Computing", which became an international conference which is still continuing. However...

19 Pro and Against Arguments

Adleman experiment was a historical achievement, the proof that *it is possible.* However, the experiment has considered a graph with only 7 nodes, for which the problem can be solved by a simple visual inspection. In comparison, at the beginning of the nineties, the computers were already able to solve the Hamiltonian path problem for graphs with several hundred nodes, sufficient for current practical applications (in the meantime, the progresses continued).

Moreover, the solution was obtained by means of a space-time trade-off, the number of molecules used was exponential with respect to the number of nodes. Juris Hartmanis, an authoritative name in computer science, after expressing his enthusiasm (Hartmanis compares computer science with physics: while the latter progresses by means of crucial experiments, the former progresses by means of proofs that something can be done, by *demos*; Adleman has produced such a *demo*!), has computed the quantity of ADN which is necessary in order to apply Adleman's procedure for a graph with 200 nodes and he has found that the weight of the ADN would be greater than the weight of the Earth...

From a practical point of view, DNA computing is, in a certain extent, in the same point even now. Numerous experiments, but all of them always dealing with "toy problems", a lot of theory, a lot of lab experience gained in dealing with DNA molecules, with results of interest for the general lab technology (just one example:

an improved version of PCR, the Polymerase Chain Reaction, called XPCR, was proposed; one of the inventors is a mathematician, Vincenzo Manca, mentioned already in the first pages of this text), but the domain has moved towards nanotechnology, no computability practical applications were reported (unless if, and this is plausible, there were applications in cryptography which are still classified).

However, the list of possible advantages of using DNA molecules for computing is large: a very good efficiency as a data support, with one bit at the level of a nucleotide; energy efficiency; parallel and non-deterministic behavior, two dreams of computer science (with the mentioning that the non-determinism also brings problems, for instance, providing false solutions); a very developed laboratory technology; robustness, predictability, reversibility of certain processes.

20 The Marvelous Double Helix

The DNA molecule has surprising properties at the informational and computational level. Let us remind that, formulated in "syntactic" terms, we have two strings of letters A, C, G, T, the four nucleotides, placed face to face, in Watson-Crick complementary pairs, always A being paired with T and C with G. The two strings are oriented, in opposite directions with respect to each other; the biochemists indicate the directionality by marking one end of a string with $3'$ and the other end with $5'$. There already appear here a surprise, first pointed out by G. Rozenberg and A. Salomaa in *Technical Report* 96-28 of Leiden University, The Netherlands (October 1996), "Watson-Crick complementarity, universal computations, and genetic engineering": the structure of the DNA molecule "hides", in a codified manner, the computing power of Turing machines! The formulation above is not precise, it however corresponds to the following observation. Already in 1980, it was proved (J. Engelfriet and G. Rozenberg) that *any language whose strings can be recognized by a Turing machine can be written as the image of a specified fixed language, let us denote it with $TS(0,1)$, by means of a sequential transducer.*

The previous language is the so-called "twin-shuffle" over 0, 1 (hence the used notation). Shuffle is the operation of mixing the letters of the two words, without changing their ordering (exactly as in the case of shuffling two decks of playing cards of different back colors). Here we shuffle the letters of two "twin" words, one string of symbols 0, 1 and the second string identical with this one, but changing the "color" of each symbol (for instance, we can add an upper bar or a prime to each symbol in order to get the twin string). In turn, the sequential transducers are the simplest transducers, with a finite memory and with a head which scans the string from left to right. Let us note that we work with four symbols, let us say 0, 1 and their pairs $0', 1'$. Exactly the number of the nucleotides, four. Let us also note that $TS(0,1)$ is a fixed language. Given an arbitrary language, if it is recognized by a Turing machine, then it can be obtained from this unique language $TS(0,1)$, only the transducer depends on the language.

The nice and significant surprise is that the language $TS(0,1)$ can be obtained by means of "reading" the DNA molecules, in the following way: let us walk along the two Watson-Crick complementary sequences, from the left to the right, advancing randomly along the two strands, and associating with the four nucleotides A, C, G, T symbols 0, 1 according to the following rule: $A = 0, G = 1, T = 0', C = 1'$. Collecting all these strings over $0, 1, 0', 1'$, for all readings of all DNA molecules, we get a set which is exactly $TS(0,1)$!

Consequently, any language which can be defined by a Turing machine can be obtained by translating these readings of the DNA molecules by means of the simplest transducer, the sequential one, with a finite memory. The transducer depends on the language, it "extract" from $TS(0,1)$ the result of the computations of a Turing machine. The power is there, what we have to do is only to make it visible. (In a certain sense, we have again the coupling of a simple process, the "reading" of the DNA molecule, and an observer of the first order, a simple one, the sequential transducer, like in the papers of M. Cavaliere and P. Leupold mentioned before, with the result reaching the highest level of computability, the power of Turing machines.)

Two questions arise in this framework. For instance, we mentioned the different orientation of the two strands of the DNA molecule, but in the previous reading we pass along the two strands in the same direction, from the left to the right. There is no problem, the reading of the double stranded DNA molecules can proceed in opposite directions and the result is the same. Second: nature is redundant, are all the four nucleotides (the four symbols $0, 1, 0', 1'$) necessary in order to cover, in the sense discussed above, the power of Turing machines? No, three symbols are sufficient – but not two! Proofs for all these results can be found in the monograph (translated in Japanese, Chinese, and Russian) Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, 1998.

Speaking about computations and redundancy, let us remember that a large part of the DNA molecule is "residual", it does not codify genes and we do not exactly know what it is used for. We can then speculate: if in the cell, at the genetic level, one performs computations (the viruses are strings of nucleotides, hence their identification is a parsing operation, hence a computation), and these computations are supposed to be complex, why not?, even of the level of Turing machines, then we need a "workspace", a "tape" which in the end remains empty in most of its length, with the result placed in a finite part of it (at the beginning in the case of the Turing machine tape). Can then the DNA without an apparent usefulness be the workspace for complex computations, which we cannot yet understand?

21 Computing by Splicing

In his experiment, Adleman has not used the splicing operation, but the biochemical ingredients specific to the splicing have been used in many other cases:

restriction enzymes, which cut the DNA molecules in well specified contexts, ligases which glue back the nucleotides thus repairing the strands, recombination on the basis of the ”sticky ends” of the molecules with the strands of different lengths, hence with nucleotides which do not have their Watson-Crick pairs.

I do not recall biochemical details or mathematical details concerning the splicing operation. In short, two molecules (represented as simple strings, because the nucleotides of a strand are precisely identified by their complementary nucleotides placed on the other strand) are cut in two parts each, in the middle of a context specified by a pair of substrings, and the fragments obtained are recombined crosswise, thus obtaining two new strings. Starting from an initial set of strings and applying this operations repeatedly (with respect to a given finite set of contexts, hence of *splicing rules*), we obtain a computing device, a language generator, similar to a grammar. We obtain an *H system.* A large part of the monograph *DNA Computing. New Computing Paradigms* cited before is dedicated to the study of these systems: variants, extensions, generative power, properties.

Always when a new computing model is introduced, the first question to clarify concerns its power, in comparison with the automata theory and language theory classifications – the Turing machine and its restrictions, the Chomsky grammars, the Lindenmayer systems. Let us only note that the two ”poles” of computability are the power of Turing machines, through the Turing-Church thesis the maximal level of algorithmic computability, and the power of the finite automata, the minimal level. In terms of grammars and languages, the maximal class is that of unrestricted Chomsky grammars and of recursively enumerable languages, while the minimal one corresponds to regular grammars and languages.

The H systems with a finite number of starting strings and a finite number of splicing rules generate only regular languages. This is not sufficient as computing power, moreover, a ”computer” of this level cannot have (convenient) universality properties, hence it cannot be programmable.

Interesting and attractive enough is the fact that, adding a minimal control on the splicing operation, with many controls of this kind suggested by the area of regulated rewriting or coming from biology (example: associate a promoter, a symbol, with each splicing rule and the rule is applied only to strings which contain that symbol; a variant – the symbol does not appear, it acts as an inhibitor), then we obtain H systems which are equivalent with the Turing machine. The proof is constructive, therefore we ”import” in this way from the Turing machines the existence of the universal machine, which means that we get an universal H system, a programmable one.

Unfortunately, so far, no such universal ”computer” based on splicing was realized in a laboratory. The passage from the natural case, with an uncontrolled splicing operation (thus with the power under the power of the finite automaton), to the controlled case was not yet done in a laboratory and it is not clear whether it can be realized in the near future. The construction of the universal computer based on splicing has to still wait...

22 An Important Detail: The Autonomous Functioning

Let us not forget that a universal, programmable computer should work autonomously, that is, after starting a program, the computer continues without any external control. This is completely different from the usual DNA computing experiments, where the human operator (or a robotic operator) controls the whole process. For instance, in the case of the 1994 experiment, Adleman was, in fact, the "computer", he has only used the DNA molecules as a support for the computation, while the computation complexity counted the lab steps performed by the biochemist, not the internal steps, the DNA operations, performed in parallel.

There are, however, promising progresses towards the implementation of autonomous computations, the key-word, very much promoted in the last years, being *self-assembly.* Remarkable achievements in this direction has obtained Erik Winfree and his group from Caltech, Pasadena, USA, and his approach is worth mentioning also because it starts (pleasantly enough for the discussion concerning the usefulness of mathematics for computer science) from an old chapter of theoretical computer science, the domino calculus of Wang Hao, developed in the beginning of sixties of the last century. In short, square dominos, with the edges colored (marked), can be used for computing (by placing the dominos adjacently, in such a way that the neighboring dominos have the contact edges of the same color), thus simulating the work of a Turing machine. We obtain once again a computing model which is universal.

Erik Winfree has constructed "dominos" from DNA molecules, with the edges marked with suitable sequences of nucleotides, he has left them free in a solution, such that the dominos glued together according to the Watson-Crick affinity of the nucleotides "coloring" the edges. The approach worked well, the experiments were successful – but everything has remained once again, in Hartmanis terms, at the level of a *demo.* It is important to underline that this time it was not addressed a given problem, as in Adleman case and as in most of the experiments reported in the DNA computing literature, but it was implemented in a laboratory a Turing machine, hence an universal computing model – that is why this *demo* is perhaps farther reaching than that of Adleman (however, Adleman was the first one...).

There also are other attempts to obtain autonomous "computers" in a laboratory. I mention here only the simulation of a finite automaton, an achievement of a team from Weizmann-Rehovot and Tehnion-Haifa, Israel: Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro, "Programmable and autonomous computing machine made of biomolecules", *Nature*, vol. 414, November 2001, pages 430–434, with the mentioning that one deals with a finite automaton with only two states. Again, only a *demo...*

23 What Means to Compute *in a Natural Way*?

The *DNA Computing* monograph has also chapters dedicated to other ways of computing, inspired from the DNA biochemistry, for instance, by insertion and deletion of substrings (in given contexts), by means of a "domino game" with DNA molecules which are coupled on the basis of the Watson-Crick complementarity, a model different from the Wang Hao one.

Splicing, insertion-deletion, prolongation of strings. In membrane computing we use the multiset processing. The evolution itself is mainly based on recombination/splicing, the local mutations appear only accidentally. In contrast, the existing computers and almost all theoretical models of computing use the string rewriting operation. One works locally, on strings of arbitrary length. This observation is valid for automata, grammars, Post systems, Markov algorithms. All these operations, both the rewriting and the "natural" ones (the splicing only with an additional control), so different among them, lead to computability models of the same power, that of the Turing machine.

The question is obvious: what means to compute *in a natural way*? With many continuations: Why computer science has not considered (with rare exceptions) also other operations different from rewriting? Can we devise (electronic) computers based on "natural operations" (for instance, using the splicing or other forms of recombination)? When Hilbert has formulated the question "what is mechanically computable?", he probably had in mind formal logical systems, where the substitution is a central inference rule, and Turing has proposed an answer in the same language. Were we influenced in this way to think in the same terms when we have designed the first computers? I have never heard that the engineers have said that we cannot imagine, maybe also construct, computers based on different operations.

It remains the question whether or not such new types of computers would be better than the existing computers or not. Theoretically, they will have the same power, hence the differences should be looked for on different coordinates: computational efficiency, easiness of use, learning possibilities and so on.

I said above that the H systems are either of the power of finite automata or equivalent with Turing machines. Similar situations are met in the membrane computing. Can we then say that the classes of automata and grammars which lie in between finite automata and Turing machines – and there are many such classes investigated in the theoretical computer science – are not "natural"? In some sense, this is the case. For instance, the context-free languages have a definition which has a mathematical-linguistic motivation, while the context-sensitive languages have a definition with a motivation coming from the complexity theory (it refers to the space needed for generating or recognizing the strings of a language).

24 Let Us Pass to the Cell!

In spite of the theoretical achievements, of numerous successful experiments (however, dealing with problems of small dimensions) and of the continuous progresses in what concerns the lab techniques, the DNA computing has not confirmed the enthusiasm of the twenty years ago, after the announcement of the Adleman experiment – if not having, as I have suggested before, application in cryptography which will be declassified only after several decades. There are elements which can support this assumption. For instance, during the first DNA Computing Conference, Princeton, 1995, a communication was presented (D. Boneh, C. Dunworth, R. Lipton: "Breaking DES using a molecular computer") which described a possibility to break Data Encryption Standard, DES, the system used by the American administration, using DNA, in four months. Next year, the subject was discussed by a team containing also Adleman, and the proposed DNA experiment was supposed to can break DEA in five days, provided that the lab operations would be done by robots. A further paper of this kind was presented in 1997, the year when DES was broken also with electronic computers and then abandoned.

Anyway, at some years after Adleman experiment it was clear that one cannot go essentially further, it was necessary to have one more innovative idea, one more "breakthrough" in order to make an essential step towards applications (towards a "killer-app", as the Americans use to say), and one of the "explanations" of this situation was the fact that DNA molecules behave better *in vivo* (more predictable, more robustly) than *in vitro*. The suggestions is just natural: let us go to the cell!

At the personal level, this moment coincided with the writing of the *DNA Computing* monograph, a fact which repeated almost systematically in the first two decades of my research career: after approximately five years of work in a branch of theoretical computer science, I have put together, alone or in collaboration, the results, publishing a monograph, and after that I have passed to another topic – still remaining in the framework of theoretical computer science, especially of formal language and automata theory. A lack of perseverance or an excess of curiosity? Maybe a part of each of them, but a lucky combination: all chapters of theoretical computer science which I have explored before passing to the membrane computing area were used, sometimes in a decisive extent, in this last domain – with which I have discontinued the tradition of a change at each five years: after sixteen years dedicated almost exclusively to membrane computing, in spite of the fact that I have written, as usually, a monograph after about five years from the first paper, there is no sign of decreasing the interest for this area.

25 The Fascinating Cell

The cell is really fascinating for a mathematician-computer scientist. I am sure that this is true also for biologists. The smallest entity which is unanimously considered *alive*. The topic is not trivial: at the middle of years 1980, at the Santa Fe Institute for complexity studies a new research vista was initiated, under the name of *artificial life*, as an extension of artificial intelligence, aiming to investigate the life per se, to simulate it on non-biological supports, on computer and in mathematical terms. The starting point was, of course, the attempt to have a definition for what we intuitively call *life*, but the progresses have not went too far: all definitions either left out something alive, or they ensured that, for instance, the computer viruses are alive (they have "metabolism", self-reproduction etc.).

Let us also remember that already Erwin Schrödinger has a book whose title asks *What is Life?* (Cambridge Univ. Press., 1967, translated in Romanian in 1980).

The cell passes this test. It is an extraordinarily small "factory", with a complex, intricate and efficient internal structure, where an enormous number of agents interact, from ions to large macromolecules like that of ADN, and where informational processes are carried out at each place and in each moment. Some cells live alone (I am not saying "isolated"), as unicellular organisms, other cells form tissues, organs, organisms.

It is a topic of interest the one concerning the role of the cells in making possible the life itself. I am only citing the reference book B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002, the paper of Jesper Hoffmeyer "Surfaces inside surfaces. On the origin of agency and life", *Cybernetics and Human Knowing*, vol. 5, 1998, pages 33–42, also important for what follows because it proposes the slogan "life means surfaces inside surfaces", referring to the membranes which define the inner structure of the cells, and I end with a paragraph from the book of S. Kauffman, *At Home in the Universe*, Oxford University Press, 1995: "The secret of life, the wellspring of reproduction, is not to be found in the beauty of Watson–Crick pairing, but in the achievements of collective catalytic closure."

I am adding also a suggestive equation-slogan, which acad. Solomon Marcus has launched during one of the first Workshops on Membrane Computing, the one in Curtea de Argeş, 2002:

Life = DNA software + membrane hardware.

26 The Membrane. From Biology to Computability

We have thus arrived to a fundamental ingredient – the membrane. One can speak very much about it, and the biologists and the experts in bio-semiotics have done it. The cell itself exists because it is separated from the neighboring environment by a membrane. Not only metaphorically, any entity exists because it is delimited by a "membrane", actual or virtual, from the world around.

The (eukaryotic) cell also has a number of membranes inside: the one which encloses the nucleus, the complicated Golgi apparatus, vesicles, mitochondria. From a computational point of view, the main role of these membranes is to define "protected reactors", compartments where a specific biochemistry takes place. There also are other features-functions of the biological membranes which are important for membrane computing: in membranes are placed protein channels which allow the selective communication among compartments; on membranes are bound enzymes which control many of the biochemical processes which take place around them; the membranes are useful also for creating reaction spaces small enough so that the molecules swimming in solution can get in contact so that they can react. It is said that when a compartment is too large for the local biochemistry to be efficient, nature creates new membranes, in order to obtain small enough "reactors" (so that, by Brownian movement, the molecules collide sufficiently frequent and react) and for creating new "reaction surfaces".

I stress the fact that I look here to the cell, its structure, and processes inside it through the glasses of the mathematician-computer scientist, ignoring many biochemical details (for instance, the structure itself of the membranes) and interpreting the selected ingredients according to the goal of this approach: to define a computing model.

Let us give some details, starting with the essential role of membranes in communication. If, in the biological cell or in the model we are going to define, the compartments delimited by membranes evolve separately, then we will not have one "reactor", but a number of neighboring "reactors", evolving independently. However, the membranes ensure the integration. The polarized molecules or those of great dimensions cannot pass through the (phospholipid, with a polarized "head" and two hydrophobic "legs") molecules of the membranes, but they can pass across a membrane through the protein channels embedded in it. This passage is selective and sometimes it is done against the gradient, from a smaller concentration to a higher concentration. A very interesting case is that of the simultaneous passage through a protein channel of two or more molecules: the respective molecules cannot pass separately, but they can do it together, either in the same direction (*symport*) or in opposite directions, one molecule entering the respective compartment and the other one going out, simultaneously (*antiport*). An important chapter of membrane computing is based on these operations and the interest comes from the particularity of this process: there is no rewriting, but only object transport across the borders defined by the membranes, there is no erasing, but only communication. *Computing by communicating* (objects). We can formulate also in this context the question *what means to compute in a natural way?*

We can read in many places about the informational processes taking place in a cell, in most cases with the involvement of membranes, too.

"Many proteins in living cells appear to have as their primary function the transfer and processing of information, rather than the chemical transformation of metabolic intermediates or the building of cellular structures. Such proteins are functionally linked through allosteric or other mechanisms into biochemical 'circuits' that perform a variety of simple computational tasks including amplification, integration, and information storage."

This is the abstract of the D. Bray paper "Protein molecules as computational elements in living cells", published in *Nature*, vol. 376, July 1995, pages 307–312. In their turn, S.R. Hameroff, J.D. Dayhoff, R. Lahoz-Beltra, A.V. Samsonovich, S. Rasmussen, in a paper from *Computer*, November 1992, pages 30–39, interpret the cytoskeleton as an automaton, while W.R. Loewenstein, in *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford University Press, 1999, constructs a whole theory starting from the informational aspects of the cell life. About the bio-semiotics of the cell has elaborated in many places Jesper Hoffmeyer, already mentioned at the previous pages. I am citing here only his paper "Semiosis and living membranes", presented at *Seminário*

Avançado de Comunicaçao e Semiótica. Biossemiótica e Semiótica Cognitiva, Sao Paolo, Brasil, 1998.

In this context we can also remember the essential role of the water in the life of the cell, as well as the processes of moving water molecules across membranes through the dedicated channels, the *aquaporines*, in whose discovery our colleague Gheorghe Benga had pioneering contributions.

27 A Terminology-History Parenthesis

Before passing to a quick description of membrane computing, let me point out a few preliminary things.

First, about the name of the domain. I have called it *membrane computing*, starting from the position of the membrane in the life of the cell, in its architecture and functioning, but the choice was not the best one. "Cellular computing" was probably the most "marketable" choice, but I have discarded it as being too comprehensive.

Then, the name of the models: in the first papers, I used "membrane systems", but soon those who started to investigate these models have called them "P systems", continuing the line of other computing devices having a name (H systems are the closest ones). In the beginning this induced to me some public discomfort, for instance, during conferences, but the letter P soon became autonomous, completely neutral to me. What is interesting is that there are papers which use the syntagma "P system", sometimes even in the title, without citing any paper of mine. Of course, it would be a great success if this syntagma will became largely folkloric...

The domain has grown very rapidly and it is still active after more than sixteen years since its initiation. I have sometimes asked myself which were the explanations, and what I can do for enhancing the growth. Several aspects concurred to the interest for the membrane computing: the favorable context (the natural computing "fashion" mentioned in the beginning); the right moment, on the one hand, with respect to the DNA computing (which, in some sense, is covered and generalized by membrane computing), on the other hand, with respect to the theoretical computer science in general and the formal language theory in particular.

There are several things to be mentioned here. After four decades since the introduction of Chomsky grammars, the formal language theory became "classical" enough and got somewhat retired from the front research (almost completely in USA), even if there still exist specialized conferences (for instance, about finite automata and their applications) or more general conferences (DLT – Developments in Language Theory). Membrane computing appeared as a continuation and an extension of formal language theory: the main investigation objects are no longer the strings of symbols and the languages, but (I anticipate) the multisets of symbols and the sets of multisets. Strings, without taking into account the ordering of the symbols, more technically speaking, strings "seen" through the Parikh application, the one which tells us the number of occurrences of each symbol in a given string. The consequence was that a large number of researchers in formal language theory became interested in the new research area. Among them, since the very beginning, important names, such as Arto Salomaa (Finland) and Grzegorz Rozenberg (The Netherlands), Oscar H. Ibarra (USA), Sheng Yu (Canada), Kamala Krithivasan (India), Takashi Yokomori (Japan), Mario J. Pérez-Jiménez (Spain), as well as very active researchers from my generation, such as Jürgen Dassow (Germany), Erzsébet Csuhaj-Varjú (Hungary), Jozef Kelemen (The Czech Republic), Rudolf Freund (Austria), Gheorghe Marian and Gabriel Ciobanu (Romania), Yurii Rogozhin (Republic of Moldova), Linqiang Pan (China), many of them coagulating around them research groups dedicated to membrane computing.

Somewhat surprising was the rapidly growing number of PhD students – now doctors – who have presented theses in membrane computing. There are over 50 at this moment. I mention only the first two, Shankara Narayanan Krishna (India) and Claudio Zandron (Italy), with the theses presented in 2001, respectively, in 2002. Starting with the summer of 2014, C. Zandron is the chairman of the steering committee of membrane computing.

A comprehensive information about the membrane computing area can be found at the domain website from the address `http://ppage.psystems.eu`, hosted in Vienna (it is the successor of a page which has functioned for many years in Milan, Italy, at the address `http://psystems.disco.unimib.it`).

Of course, it has counted very much the "sociology" of the domain. A *community* was soon created, and this is very important, not only in science, but in culture in general. They have contributed to that the seniors mentioned above, the yearly conferences (started in 2000, with the first three editions organized in Curtea de Argeş, Romania, where the meeting returned for the tenth edition and where I intend to also organize the twentieth edition) as well as a series of meetings which I would like to specially emphasize, one of a unusual format, which I have organized for the first time in Tarragona, Spain, in 2003. After that, it took place every year in Seville, also in Spain. Because it had to have a name, I called it "Brainstorming Week on Membrane Computing". One week when researchers interested in membrane computing work together, far from the current preoccupations, teaching or bureaucratic tasks. A very fruitful idea was to collect in advance open problems and research topics and to circulate them among the participants before the meeting in Seville, then addressed, in collaboration, during the Brainstorming. Very useful meetings – in the website of membrane computing one can find the yearly volumes, with the papers written or only started during the Brainstorming.

Very useful was, of course, the Internet. The first paper, "Computing with membranes", has waited more than one year before it was published in *Journal of Computer and System Sciences* (vol. 61, 2000, pages 108–143), but, because I was in Turku, Finland, in the autumn of 1998, I made the paper available on Internet, in the form of an internal report of TUCS, Turku Center for Computer Science (*Report No.* 208, 1998, www.tucs.fi). Until 2000, when the journal paper has appeared, there were written some dozens of papers, making possible the organization of the first meeting dedicated to this topic, in Curtea de Argeş.

28 A Quick View on Membrane Computing

Let us not forget: we want to start from the cell and to construct a computing model. The result (the one proposed in the fall of 1988) is something of the following form. We look to the cell and we abstract it until we only see the *structure* of the hierarchically arranged *membranes*, defining *compartments* where *multisets of objects* are placed (I am using a generic term, abstract, free of any biochemical interpretation); these objects evolve according to given *reactions*. A multiset is a set with multiplicities associated with its elements, hence it can be described by a string; for instance, *aabcab* describes the multiset which contains three copies of a, two of b, and one of c. All permutations of the string *aabcab* describe the same multiset. The reactions, in their turn, are described by multiset "rewriting" rules, of the form $u \to v$, where u and v are strings which identify multisets. Initially (in the beginning of a computation), in the compartments of our system we have given multisets of objects. The evolution rules start to be applied, like biochemical reactions, in parallel, simultaneously, making evolve all objects which can evolve – and thus the multisets change. Using a rule $u \to v$ as above means to "consume" the objects indicated by u and to introduce the objects indicated by v. We have to notice that the objects and the rules are localized, placed in compartments, the rules in a given compartment are applied only to objects from that compartment. Certain objects can also pass through membranes. We proceed by applying rules until (like in the case of a Turing machine) we get stuck, no rule can be applied, and then the computation halts. The result of a halting computation is "read", for instance, in the form of the number of objects placed in a compartment specified in advance.

Processing of multisets (of symbols), in parallel, in the compartments defined by a hierarchical structure of membranes – this is the short description of a "P system". A distributed grammar, working with multisets of symbols – this is the direct connection with the formal language theory.

The working site starting here looks endless.

First, one can introduce a large number of variations of P systems, with a mathematical, computer science, biological motivation, or motivated by applications.

From the point of view of mathematics, the models should be minimalistic, they have to contain the smallest number of ingredients. For computer science, a computing model is good to be as powerful as possible, in the best case universal, equivalent with the Turing machine, and as efficient as possible, in the best case able to solve **NP**-complete problems in polynomial time.

Biology and applications provide a long list of alternatives, starting with the way of arranging the membranes (hierarchical, as in a cell, or placed in the nodes of an arbitrary graph, as in tissues and other populations of cells), the types of objects (symbols as before, strings or even more complex data structures, such as graphs or bidimensional arrays), the form of the evolution rules (also dependent on the type of objects), the strategies of applying them, the way of defining the result of a computation.

I have mentioned before the multiset rewriting rules. They can be *arbitrary, non-cooperative* (with the left hand multiset consisting of a single object, which corresponds to context-free rules in Chomsky grammars), or, an intermediate case, *catalytic* (of the form $ca \to cv$, where c is a catalyst, an object which assists object a in its transformation to the multiset v). Then, we have the *symport* and *antiport* rules, which move objects from a compartment into another one (example: the antiport rule $(u, out; v, in)$, associated with a membrane, moves the objects indicated by u from this membrane to the surrounding compartment and the objects indicated by v in the opposite direction). Very important are the rules which *divide* membranes, because they increase, even exponentially, the number of membranes in the system. Many other types of rules were investigated (for instance, with a control on their application – with promoters, inhibitors, etc.), but I do not mention them here, the presentation would become too technical for the intentions of this text.

If the objects in the compartments of a system are strings, then they evolve by means of operations specific to strings: rewriting, insertion and deletion, or, in order to make the model more uniform from a biological point of view, by the splicing operation from the DNA computing.

An interesting situation is that when we work with symbol objects, hence with numbers, but the result of a computation is "read" outside the system, in the form of the string of the objects which are expelled from the system. It is worth noticing the qualitative difference between the internal data structure, the multiset, and the external one, the string, which carries out positional information.

In turn, the applications need a completely different strategy of constructing the models – far from minimalistic, but adequate to the modeled piece of reality; this time not the computing power is of interest, but the evolution in time of the system. I will come back to applications.

Over this small jungle of models one superposes the investigation program of the classic computer science: computing power, normal forms, descriptional complexity, computational complexity, simulation programs, etc., etc.

29 Classes of Results (and Problems)

Of course, I will not recall precise theorems, but I will only mention the two main classes of results in membrane computing and their general form.

Computational completeness/universality: most of the classes of P systems considered so far are equivalent with Turing machines, they are computationally complete. Because the proofs are constructive, in this way one also brings to membrane computing the universality property in the sense of Turing (that is why we speak about computational completeness and universality as they would be synonymous). In most cases, this result is obtained for systems of a reduced, particular form, with a small number of membranes. For instance, cell-like P systems with only two membranes, using catalytic rules (hence not of the general form) can compute whatever the Turing machines can compute.

An important detail: *two* catalysts are sufficient. It is an open problem whether the P systems with only one catalyst are universal. The conjecture is that the answer is negative, but the proof still fails to appear. This is one of the most interesting types of open problems in membrane computing (many of them still open): identifying the precise borderline between universality and non-universality.

Efficiency: the classes of P systems which can grow (exponentially) the number of membranes can solve **NP**-complete problems in a polynomial time. The idea is to generate, in a polynomial time, an exponential working space and then to use it, in parallel, for examining the possible solutions to a problem. Membrane division helps, similarly the membrane creation, similarly other operations. Like in the case of the Adleman experiment, we have again a space-time trade-off, but in our case the space is not provided in advance, but it is created during the computation, through "mitosis" or by means of other "realistic" biological operations.

There are also in this area open problems concerning the borderline between efficiency and non-efficiency, but more difficult to be stated in plain words.

Interesting is a somewhat unexpected fact. Using rules of the form $a \to aa$, applied in parallel, we can produce an exponential number of copies of a in a linear number of steps. (In n steps, we get 2^n copies of a.) However, such an exponential working space is not of any help in solving high complexity problems in a feasible time– this is what the so-called *Milan theorem*, from Claudio Zandron PhD thesis, says. If these objects are localized, placed in an exponentially large number of membranes, then the situation is different. Otherwise stated, not only the size of the working space matters, but also its structure, the possibility to apply different rules in different compartments. This is a subtle aspect, which I do not know whether it has been met also in other frameworks.

For details, the reader is refereed to the monograph *Membrane Computing. An Introduction*, published by Springer-Verlag in 2002 (and recently translated in Chinese) and, especially, to *The Oxford Handbook of Membrane Computing*, edited by Gh. Păun, G. Rozenberg, and A. Salomaa and published by Oxford University Press, in 2010.

30 Significations for Computer Science and for Biology

A computing model which has the same power as the Turing machine is a good thing, such a computer is universal not only in the intuitive sense, but it is also programmable. Moreover we have here a distributed, parallel computer, with a great degree of non-determinism, controlled in various biologically inspired ways.

Let us, however, observe the similarities and the differences between a usual computer program, a set of instructions of a Turing machine, and a set of evolution rules of a P system. In the programming languages, the programs consist of precisely ordered instructions, perhaps labeled and addressed by means of these labels. In the case of the Turing machine, the sequence of instructions to be applied is determined by the states of the machine and by the contents of the tape. In the cell case, the reactions are potential, their set is completely unstructured, and their application depends on the available molecules. The evolution rules are just waiting for the data to which they can be applied, there is a competition between rules with respect to the objects to process.

The differences are visible and they suggest once again the question *what means to compute in a natural way?*, adding now the question whether we can work with programs in the form of completely unstructured sets of instructions.

On the other hand, in the first moment, it is expected that the biologist reaction to results of the type of the equivalence with the Turing machine is indifference, a raising of the shoulders. Another domain, another language, another book... But: if the cell is so powerful from a computational point of view, then, according to an old result, the Rice theorem ("all nontrivial problems – having both instances with a positive answer and instances with a negative answer – about a computing model equivalent with Turing machines are algorithmically undecidable"), no nontrivial question about the cell can be solved in an algorithmic way, by means of a program. The biologists formulate every day such questions: How a cell, a cell population, an organ or an organism evolves in time? Is there a substance which gets accumulated over a given threshold, in a given compartment? What happens if we add a multiset of molecules (a medicine), does the state of an organ improves (from specified points of view)? – and so on. If a model of the cell would be decidable, then we could find the answer to such questions by (algorithmically) examining the model, at a given initial state. But, because this is not possible (cannot be done in principle, not only we cannot do it now, here), what remains to do are the laboratory experiment (expensive and time consuming), the computer experiment (cheap, fast, but with the relevance depending on the quality of the model), and, theoretically, the non-algorithmic, ad-hoc, approach.

The previous paragraphs can be seen also as a plead for biology to learn new languages, in particular, the language of theoretical computer science, thus having the possibility of raising problems and of finding solutions which cannot appear, cannot be even formulated in the previous language. This would be an essential step towards infobiology.

31 Three Novel Computer Science Problems

In the continuation of the discussion about the significance for computer science, let us point out a remarkable fact: natural computing in general and membrane computing in particular raise theoretical questions which were not considered in the framework of the classical computer science. Here are three questions of this kind, all three pertaining to complexity theory.

Like in the case of Adleman, most experiments of DNA computing started from an instance of a problem and constructed a "computer" associated with that instance. The standard complexity theory does not allow such an approach, it asks for *uniform* solutions, for programs/algorithms which start from the problem (and its size) and solve all instances of the problem. The idea is that during the programming stage one can already work on solving the problem, so that one can then pretend that the solution was found faster than it was the case in reality. That is why, also for the uniform solutions one limits the time allowed for programming, for constructing the algorithm. Let us then place a bound also on the programming time in the case when we start from an instance, so that we cannot cheat here either. The relationship between uniform solutions and semi-uniform (with a limited time for programming) solutions is not clarified yet, in spite of its importance for the natural computing. In membrane computing there were reported a series of related results – see, for instance, recent papers by Damien Woods (Caltech, USA), Niall Murphy (Microsoft Research, Cambridge, UK), Mario J. Pérez-Jiménez (Seville University, Spain).

Second: in DNA computing and in many models in membrane computing, at least part of the steps of a computation are of a non-deterministic type, but in the end the experiment/computation provides a unique result. The idea is to organize the computation in such a way that it is *confluent*, with two variants: either the system evolves non-deterministically for a while, then it "converges" to a unique configuration and then it continues in a deterministic way, or the system "converges logically", it gives the same result irrespective how it evolves. Again, the complexity theory lacks a study of these situations, of the cases intermediate between determinism and non-determinism.

Finally, the biology provides situations where extended resources wait for external challenges which activate a suitable portion of the resource. The examples of the brain and of the liver, from which we use at any given time only part of the huge number of available cells, are the most known. We can then imagine "computers" – for instance, neural networks – with an arbitrarily large number of cells/neurons, but containing only a limited quantity of information (not to hide there the solution of a problem); after introducing a problem in the system, one activates the necessary number of cells/neurons for solving it. There is no theory dealing with this strategy (of using *pre-computed resources*). How the pre-computed working space should look in order to contain only "a limited quantity of information", how this information can be defined and measured, when a system with pre-computed resources is acceptable/honest, it cannot hide the solution of a problem in its structure?

Natural computing not only motivates the improvement of old results in computer science, but it also makes necessary new developments, which were not imagined before.

32 About the Tools Used in Membrane Computing

In order to stress once again the relationships between various branches of theoretical computer science which, at the first sight, look far from each other, and the fact that membrane computing, the natural computing in general, use many old techniques and results, let me remind some details from my personal experience.

In the first universality proof for P systems I have used the result of Yuri Matijasevich mentioned also before, of characterizing the sets of numbers computed by Turing machines as solutions of diophantine equations. I have, however, soon realized that a simpler proof can be obtained starting from the characterization of the same sets of numbers with the help of the matrix grammars. The initial paper was published in this form. In this context it appears the necessity of improving some old results in this area. After a while, also the matrix grammars were replaced, the proofs are now based mainly on register machines, investigated already in the sixties.

A technique even older was useful in the first universality proof for H systems, namely the way of functioning of Post systems, which were introduced at the beginning of the years 1940. Adapted to the splicing operation, this has led to a technique called *rotate-and-simulate*, which has become almost standard for H systems and their variants.

In the first years of my research activity, I was much interested in matrix grammars and I have concluded this research with a monograph (published in Romanian, in 1981), extended after a while to a book (published by Springer-Verlag, in 1989), in collaboration with Jürgen Dassow, from Magdeburg, Germany, dedicated to all restrictions in the derivation of context-free grammars. The same happened with other domains which were useful in the membrane computing; the Marcus contextual grammars and the grammar systems are the most important of them.

In mathematics and computer science it is not possible to say in advance whether and when a subject or a result will be useful...

33 Spiking Neural P (SNP) Systems

A class of P systems inspired from the brain structure and functioning deserves to be separately discussed. It was introduced later than other models (M. Ionescu, Gh. Păun, T. Yokomori: "Spiking neural P systems", *Fundamenta Informaticae*, vol. 71, 2006, pages 279–308), but it seems that it will get earlier hardware implementations useful to computer science (details about this possibility can be found in the paper "The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits", by Zihan Xu, Matteo Cavaliere, Pei An, Sarma Vrudhula, published in *Fundamenta Informaticae*, vol. 134, issue 1-2, January 2014, pages 183–200).

In a few words, such a system consists of ”neurons” linked through ”synapses” along which circulate electrical impulses, produced in the neurons by means of specific rules. Like in the case of the real neurons (see, for instance, W. Maass: ”Networks of spiking neurons: The third generation of neural network models”, *Neural Networks*, vol. 10, 1997, pages 1659–1671), the communication among neurons is done by means of identical electrical impulses, *spikes*, for which the frequency is relevant, codifying information. Otherwise stated, important is the distance in time between spikes. In each moment, the axons are a sort of ”bar codes”, sequences of 0 and 1 which move from a neuron to another one. Obviously, the model ignores many neuro-biological details, but even at this reductionistic level we can formulate a series of questions concerning the relevance for computer science. In a certain sense, the SNP systems use the time as a support of information. The distance between two events, two spikes here, codifies a number. Can we construct a computer with such a ”memory”? I mention the question only as a speculation – provocative at the theoretical level.

A result which deserves to be recalled refers to the search of SNP systems which are universal in the Turing sense, that is, they can be programmed in such a way to simulate any other SNP system. From the equivalence with the Turing machine, it follows immediately that such a system exists. The problem of interest concerns the number of neurons of an ”universal brain” of this kind, able to simulate any computation in any particular system. This number is not at all too large. In the paper ”Small universal spiking neural P systems”, *BioSystems*, vol. 90, 2007, pages 48–60, by Andrei Păun and Gh. Păun, one uses 50 – 80 neurons, depending on the type of rules for producing spikes, but these numbers were subsequently decreased. In newspaper terms, we can say that ”there are computationally universal brains consisting of only a few tens of neurons”. From here we can either infer that a computing model of the form of SNP systems is very powerful, actually, that the neurons of these systems are *too powerful*, or that the Turing computability level is not very high – or both these conclusions. Of course, the human brain does not function as a Turing machine – but the computational paradigm was useful, in a certain extent, in modeling the brain functioning.

34 About Implementations

The DNA computing started by the definition of the splicing operation, in 1987, but about the possibility of using DNA molecules for computing there were discussions already one decade before. However, the domain became popular after Adleman experiment in 1994. An example was thus created, so that the question whether or not there are implementations of P systems is both natural and frequent. It is understood that one speaks about implementations on a biological substrate. The answer is negative. There were some attempts, but no successful experiment was reported.

An experiment of this kind was designed in the group of professor Ehud Keinan (with well known research both in chemistry and biology) from the Technion Institute in Haifa, Israel, where I have spent one week in 2006, exactly with this purpose. Two main related problems were identified from the beginning: finding a P system plausible to be implemented in a laboratory and, of course, finding the biochemical techniques necessary. We did not intend to solve an **NP**-complete problem, we have not found a reasonable one, but we have looked for a system whose behavior was illustrative for membrane computing (compartments, multisets, parallel processing), and we have chosen a system generating numbers in the Fibonacci sequence. The lab implementation seemed to be only a time issue – as well a question of money, for buying the laboratory equipments and the... DNA molecules. The plan was to simulate the membranes by means of the micro-chambers of a reconfigurable lab installation, with the objects being DNA molecules.

The first experiments did not succeed, then the... sociology of science struck again: the two PhD lady students who were in charge with this experiment moved to USA. In the meantime, an USA patent has appeared, on the name of Ehud Keinan, for implementing a P system, but using another technique, based on three non-miscible liquids placed in a common space. As far as I know, it is about a "theoretical implementation", no successful experiment was reported.

The question which naturally arises is whether or not such an experiment would bring something useful from the point of view of applications. Recalling a saying of Benjamin Franklin, "it is impossible to say what will become a newborn baby", but, having in mind the case of DNA computing, it is highly possible that this will only be a *demo*, at the level of simple calculations.

Completely different is the situation of implementations on an electronic hardware. There are several promising implementations on a parallel hardware (on NVIDIA graphic cards, in Seville, Spain), on a hardware especially designed for membrane computing (Madrid – Spain and Adelaide – Australia), on networks of computers, even on web. All these succeed in a great extent to capture the essential characteristics of P systems, the parallelism. Having in mind the parallelism, I do not call implementations, but *simulations* the cases when one uses standard sequential computers.

On the other hand, both the simulation programs and, still more, the implementations are useful in applications.

35 Applications

Membrane computing confirms an observation already made in several situations: when a mathematical theory, starting from a piece of reality, is sufficiently developed at the abstract, theoretical level, there are high chances to find applications not only in the domain which has inspired it, but in other areas too, some of them far away, at the first sight, from the reality from where the theory emerged (but having a common deep structure). It is, very convincingly, the present case.

It was just natural to return to the cell. Biology needs tools and models, the cell is not easy to model. It was stated that, after completing the human genome reading, the main challenge for the bioinformatics is the modeling of the cell (M. Tomita: "Whole-cell simulation: A grand challenge of the 21st century", *Trends in Biotechnology*, vol. 19, 2001, pages 205–210). I have already mentioned that many of the models currently used in biology are based on differential equations. In many cases they are adequate, in many cases not. Differential equations belong to the mathematics of the continuum, they are appropriate to very large populations of molecules, uniformly stirred. However, in a cell, many molecules can be found in small numbers, therefore the approximation of the finite through the infinite, as necessary for applying differential equations, can lead to wrong results. This makes necessary the discrete models, in particular, the P systems, which also have other characteristics which are attractive for the biologist: they come from biology, hence they are easily understandable, which is an aspect which should not be underestimated; furthermore, P systems are algorithmic models, directly programmable in order to simulate them on the computer; can be easily extended, are scalable, adding new components, of any type, does not change the simulation program; their behavior is emergent, cannot be predicted by just looking to the components.

There are many applications of membrane computing in biology and biomedicine. From the individual cell, the applications passed to populations of cells (e.g., of bacteria) and then to... ecosystems. Here is only one title, a suggestive one: "Modeling ecosystems using P systems: The bearded vulture, a case study", by Mónica Cardona, M. Angels Colomer, Mario J. Pérez-Jiménez, Delfi Sanuy, and Antoni Margalida, the last two being biologists, experts in the ecology of the bearded vulture and animal protection from Lleida, Spain. Of course, the ecosystem is a metaphoric cell, while the "molecules" are the vultures, goats, wolves, hunters, all these in discrete quantities, small known numbers, with no possibility to be modeled with the instruments of the continuous mathematics. Other ecosystems which were investigated concern Panda bears in China and the zebra mussel from the water basins of the Spanish hydroelectrical plants.

So far, plausible applications. Not so expected are the applications in computer graphics (but in this respect we have a previous example, that of Lindenmayer systems), cryptography (in the organization of the attack against certain cryptographic systems), approximate optimization (distributed evolutionary computing, with the distribution organized like in a cell; the number of papers in this area is very large, the topic being popular in China, and the results are surprisingly and pleasantly good – with the mentioning that the famous no free lunch theorem should cool down also here the enthusiasm), economic modeling (a metaphorical extension similar to that to ecosystems), robot control.

These two last areas of applications are part of a potentially larger one, based on the use of the so-called *numerical P systems*, where, in a cell-like framework there evolve numerical variables, not molecules; the evolution is done by means of certain *programs*, consisting of a *production function* and a *repartition protocol*. The inspiration comes from economics (Gh. Păun, Radu Păun: "Membrane computing and economics: Numerical P systems", *Fundamenta Informaticae*, vol. 73, 2006, pages 213–227). The systems of this kind compute functions of several variables, in a parallel way, and this computation is rather efficient, that is why it is expected that this somewhat exotic class of P systems will find further applications.

Details about applications can be found in the webpage of membrane computing, in the mentioned *Handbook*, as well as in the collective volumes *Applications of Membrane Computing* (edited by G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez) and *Applications of Membrane Computing in Systems and Synthetic Biology* (edited by P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez), both of them published by Springer-Verlag, in 2006 and 2014, respectively.

36 Doubts, Difficulties, Failures

During ceremonies like the today one [delivering a Reception Speech in the Academy] or with the occasion of periodical reports, it is not usual, even not appropriate, to also speak about difficult moments, even if this would be instructive for the reader and useful for the domain.

On the other hand, the hesitations and the doubts are continuously a component of the researcher life. For instance, I can compile a long list of moments where my expectations were of a certain type and the results were different.

This happened starting with the mathematical results. For instance, in the beginning I did not believe that the catalytic P systems are universal, furthermore, that they are universal even in the case of using only two catalysts. Similarly, for a while I have expected to find a class of systems for which the number of membranes induces an infinite hierarchy (of the classes of sets of computed numbers). In exchange, almost always the universality is obtained with only one or two membranes. One membrane means no structure of the system, a trivial architecture. Of course, we can see here the positive fact: the (catalytic) processing of multisets is powerful enough in order to simulate a Turing machine.

Because I have in mind the case of the DNA computing, I do not count as a failure the fact that there are no biological implementations of P systems (although such an event would have a great publicity impact), but I still wait for an implementation on a parallel or a dedicated hardware having a "commercial" value. Such an implementation is necessary and, I believe, it is also possible. For instance, some years ago, a team of biologists and computer scientists from Nothingham, Sheffield (UK), and Seville have tried to simulate on a computer the communication among bacteria, modeling the so-called *quorum sensing.* The simulation programs were able to deal with hundreds of bacteria, the biologists wanted to pass to populations of thousands of bacteria. My expectation is that the implementations, for instance, on NVIDIA cards, will reach soon this level of magnitude requested by the biologists.

Concerning the applications in general, although they were not of interest at the beginning of membrane computing, at some moment it was clear that the domain cannot pass over a certain level of development and notoriety without "real" applications. For a while, there were applications, but of the *postdiction*, not of the *prediction* type. The frequent scenario is the following: we take a biological phenomenon, discussed in a paper or in a book, we formalize it as a P system, we write a simulation program (or we take one available – at this moment we also have a specialized programming language, *P-lingua*, realized in the Seville University), we perform experiments with data from the paper or the book, and if the results are similar to those obtained in a laboratory or through other methods, we are happy. Postdiction, nothing new for the biologists, we only get more confidence in the new model and we can tune it with real data. In order to pass over this stage it is necessary to have a biologist in the team, who should come with a research question, with hypotheses which need to be checked. In turn, the computer scientist should come with sufficiently versatile models and with sufficiently efficient programs, in order to cope with the complexity of biological processes. After sixteen years, the bibliography of membrane computing applications is rather large – see the references from the previous section – although still we need biologists who have to come towards the computer scientists, maybe to learn membrane computing or, at least, to learn to use the instruments which the computer scientists have already realized (and tested).

I said before that I was continuously interested in forming a community – initially, this was an intuitive desire, later it became conscious, as this was a way to stabilize the domain against the dynamics of the groups. This looks as an external aspect, but we do not have to ignore the influence of the psycho-sociology on science, especially in the case of young branches. A group which is broken can mean a group less (it depends where its members land, whether they continue or not the research activity) or the apparition of several new groups, in new places. I have been the witness of both these two types of consequences. Fortunately, at the present time, the membrane computing community has dimensions which provide it with a comfortable inertia – which, however, does not mean that membrane computing will not get dissolved into infobiology, it already works for that...

37 At the Frontier of Science-Fiction

The main promise of natural computing is a better use of the existing computers, pushing forward the frontier of feasibility, by providing solutions, perhaps approximate, to problems which cannot be solved by means of traditional techniques. The DNA computing came with a more ambitious goal, that of providing a new type of hardware, of "biological chips", "wet processors", efficient not only in computational terms, but also in what concerns the energy consumption, or making plausible very attractive features, self-healing, adaptation, learning. Biology can suggest also new computer architectures or ideas for implementing other dreams of computer science, such as the parallel computation, the unsynchronized one, the control of distributed processes, the reversible computation and so on.

All these are somewhat standard expectations, but there also are some ideas which point out to the science of tomorrow, if not directly to science-fiction.

One of these directions is that which aims to *hypercomputability*, to "compute the uncomputable", to pass beyond the "Turing barrier". The domain is well developed, there are over one dozen basic ideas which lead to computability models stronger that the Turing machine – while physics does not forbid any one of these ideas, moreover, it even suggests ideas which look genuinely SF, like, e.g., the use of an internal time of the model which contains cycles or of a bidimensional time. It is true, Martin Davis ("The myth of hypercomputation", in *Alan Turing: The Life and Legacy of a Great Thinker*, C. Teuscher, ed., Springer, 2004, pages 195-212) considers all of them tricks by which the computing power is introduced in the model from the very beginning, in disguise, and then one proves that the model passes beyond the Turing machine (for example, one considers real numbers, which can codify, in their infinite sequence of decimals, all possible computations), but there also are some ideas which look more realistic that others.

One of them is that of *acceleration*, already discussed several decades ago, not only in computer science: R. Blake (1926), H. Weyl (1927), B. Russell (1936), have imagined processes which need one time unit (measured by an external clock) for performing the first step, half of a time unit for the second step (the process "learns"), and so on, at every step, half of the time needed by the previous step. In this way, in two time units (I insist: external, measured by the observer) one performs infinitely many (internal) steps. Such an accelerated Turing machine can solve the halting problem, hence it is more powerful than a usual Turing machine.

Let us now remember the observation that nature creates new membranes in order to get small reactors, where the reactions are enhanced, because of the higher possibilities of molecules to collide. Consequently, *smaller is faster.* The biochemistry in an inner membrane is faster than in the surrounding membrane. Let us push the speculation to the end and assume that the "life" in a membrane is twice faster than in the membrane containing it. Exactly the acceleration we have mentioned above. One can prove (C. Calude, Gh. Păun: "Bio-steps beyond Turing", *BioSystems*, vol. 77, 2004, pages 175–194) that, exactly as in the case of the accelerated Turing machine, an accelerated P system (able to repeatedly create inner membranes) can decide the halting problem.

Hypercomputability can seem to be only a mathematical exercise, but it is estimated that passing beyond the Turing barrier could have more important consequences than finding a proof, even an efficient one, of the $\mathbf{P} = \mathbf{NP}$ equality; see, for instance, B.J. Copeland: "Hypercomputation", *Minds and Machines*, vol. 12, 2002, pages 461–502.

Let us get closer to the laboratory. I have mentioned the lab implementation of a finite automaton with an autonomous functioning. A finite automaton can parse strings. The genes are strings, the viruses are strings (of nucleotides). A hope of medicine is to cure illnesses by editing genes, to eliminate viruses by identifying them and then cutting them in pieces. A more efficient idea than to introduce medicines in out body is to construct a "machinery" which can recognize and edit the necessary sequences of nucleotides, genes or viruses. To this aim, we need a carrying vector, to bring the gene editor in the right place. The identification of that place can be done by an automaton, possibly a finite one, while the vector can be a sort of nano-carrier which can be also built from DNA molecules. In short, un nano-robot, suitably multiplied, which can move from a cell to another one, curing what it is necessary to be cured. A pre-project of such a nano-robot was presented in 2004, by Y. Benenson, E. Shapiro, B. Gill, U. Ben-Dor, R. Adar ("Molecular computer. A 'smart drug' in a test tube"), to the tenth edition of the DNA Computing Conference organized in Milan, Italy. In a great extent, it was the same team which has implemented the autonomous finite automaton mentioned before.

There still are many things to be done, the possibility to have our body continuously scanned by a gene repairing robot is not at all close to us. (Such a robot can also have malevolent tasks, it can be a weapon – one can open here a discussion about the ethics of research, but there are sufficiently many debates of this type, even in bio-computer science. Also Francis S. Collins speaks about bioethics in *The Language of God*, the book mentioned several pages before.) On the other hand, there are numerous nano-constructions made of DNA, "motors", "robots", etc. The nano-technology based on DNA biochemistry is spectacularly developed. I cite, as a reference, the paper J.H. Reif, T.H. LaBean, S. Sahu, H. Yan, P. Yin: "Design, simulation, and experimental demonstration of self-assembled DNA nanostructures and motors", *Proceedings of the Workshop on Unconventional Programming Paradigms*, UPP04, Le Mont Saint-Michel, September 2004.

It is worth mentioning here also an observation made by Jana Horáková and Jozef Kelemen in "Capek, Turing, von Neumann, and the 20th century evolution of the concept of machine", from *Proceedings of the International Conference in Memoriam John von Neumann*, Budapest Polytechnic, 2003, pages 121–135, with respect to the evolution of computers, somewhat in parallel with the evolution of the idea of a robot: from organic to electromagnetic, then to electronic, and in the end tending to return to organic.

Further speculations? Without any limits, starting from facts with a solid scientific background. In the extreme edge, one can mention Frank Tipler, with his controversial eternal life, in informational terms, which is nothing else than artificial life at the scale of the whole universe (F. Tipler: *The Physics of Immortality*, Doubleday, New York, 1994). In any case, we have to be conscious that all these are plans for tomorrow formulated today in the yesterday language, to cite a saying of Antoine de Saint-Exupéry. The progresses in bioengineering can bring surprises which we cannot imagine in this moment.

38 Do We Dream Too Much?

Let us come down on the Earth, to the reality, to the natural computing as we have it now and how it is plausible to have it in the near future, adopting a lucid position, even a skeptical one, opposed to the enthusiasm from the previous section and to the enthusiasm of many authors. (I am not referring here to newspaper authors, which too often use big words when talking about bioinformatics.)

In order to promote an young scientific branch, the enthusiasm is useful and understandable – but natural computing is no longer an young research area. Let us oppose here to the previous optimism a more realistic position, starting from the differences, many and significant, between computer science and biology, from the difficulties to implement bio-ideas in computer science and computations in cells: the goal of life is life, not the computations, we, the computer scientists, see everywhere computations and try to use them for us; in a certain sense, life has unbounded time and resources, it affords to make experiments and to discard the results of unsuccessful attempts – all these are difficult to extend to computers, even if they are based on biomolecules. Similarly, life has a great degree of redundancy and non-determinism. Then, the biological processes have a high degree of complexity, moreover, they seem to mainly use the mathematics of approximations, probabilities, fuzzy sets, all of which are difficult to be captured in a computing model, not to speak about the difficulty to implement them.

Still more important: we perhaps dream too much even from the theoretical point of view. First, the space-time trade-off does not redefine the complexity classes, at most it can enlarge the feasibility space (see again Hartmanis remarks about Adleman experiment).

Then, there is a theorem of Michael Conrad (”The price of programmability”, in the volume *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Kammerer and Unverzagt, Hamburg, 1988, pages 285–307) which says that three desired characteristics of a computer, *programmability* (universality), *efficiency*, and *evolvability* (the capacity to adapt and learn), are contradictory, there is no computer which can have all these three features at the same time. We can interpret this result as a general *no free lunch* theorem for the natural computing.

A similar theorem of limitation of ”what can be done in principle” belongs to Robin Gandy, a student and collaborator of Turing, which offers general mathematical arguments to Martin Davis: the hypercomputability is a difficult thing to reach (see, for instance, the paper by R. Gandy ”Church's thesis and principles for mechanisms”, in the volume *The Kleene Symposium*, J. Barwise et al., eds., North-Holland, Amsterdam, 1980, pages 123–148). Gandy wanted to free the Turing-Church thesis of any anthropic meaning (in Turing formulation, the thesis says that ”everything which can be computed by a human being can be computed by a Turing machine”). To this aim, he has defined a general notion of a ”computing machine”, described by four properties formulated mathematically and which any ”computer”, an actual or a theoretical one, should possess. Then, Gandy proved that any machine having these properties can be simulated by a Turing machine.

Passing from theoretical computer science to applications, let me notice that there are visible limitations also in this respect. I am even convinced that, if one will make lists with the properties the models and the simulations we would like to have (adequacy, relevance, accuracy, efficiency, understandability, programmability, scalability and so on), then impossibility theorems similar to Arrow, Conrad, Gödel theorems will be proved concerning the modeling and the simulation of the cell – the very task which M. Tomita formulated.

39 Everything is New and Old All Are...

(The title of this section reproduces a verse from a poem by Mihai Eminescu, the national poet of Romania.)

In spite of what was said above, there is a more and more visible interest in the modeling of the cell. Actually, a dedicated research direction was proposed, the *systems biology*, with several programmatic papers, published in high visibility journals, such as *Science* and *Nature*. The main promotor was H. Kitano ("Systems biology: A brief overview", *Science*, vol. 295, March 2002, pages 1662–1664, "Computational systems biology", *Nature*, vol. 420, November 2002, pages 206–210), which has in mind a general model of the cell, meant to be simulated on a computer and then used, in relation also with other computer science and biological instruments, in such a way "to transform biology and medicine in a precise engineering". The goal is important and probably feasible in a medium-long term, but the insistence with which one speaks about "systems biology" as about a novel idea made Olaf Wolkenhauer to ask already in the title of his paper from *Bioinformatics* (vol. 2, 2001, pages 258–270) whether this is not only "the reincarnation of systems theory applied in biology". The paper recalls the efforts in this respect made in the years 1960, with the disappointments appeared at that time, due, among others, to the limits of the computers (but also to the limits of biology: let us remember that the Singer-Nicolson model of the membrane as a "fluid mosaic" dates only from 1972). But, besides the computing power, it is possible that something else was missing, which is perhaps missing even today, both in computer science and in biology. The last paragraph from Olaf Wolkenhauer paper invokes the name of Mihailo Mesarovic, a classic of systems theory, which, in 1968, said: "in spite of the considerable interest and efforts, the application of systems theory in biology has not quite lived up the expectations... One of the main reasons for the existing lag is that systems theory has not been directly concerned with some of the problems of vital importance in biology". His advice for biologists, continues Olaf Wolkenhauer, is that such a progress can only be obtained by means of a stronger direct interaction with the systems theory researchers. "The real advance in the applications of systems theory to biology will come about only when the biologists start *asking questions* which are based on the system-theoretic concepts rather than using these concepts to represent in still another way the phenomena which are already explained in terms of biophysical or biochemical principles... then we will not have *the application of engineering principles to biological problems*, but rather a field of *systems biology* with its own identity and in its own right." (M.D. Mesarovic: "System theory and biology – view of a theoretician", in *System Theory and Biology*, M.D. Mesarovic, ed., Springer, New York, 1968, pages 59–87)

Mesarovic words can be taken as a motto of infobiology in favor of which the whole present text pleads.

The transformation of biology and medicine in "a precise engineering" can be also related with the current difficulties to understand what is life, materialized, among others, in the current limits of the artificial intelligence and artificial life. One says, for instance, that up to now the computers are good in IA, the intelligence amplification, but not equally good in AI, artificial intelligence. Still less progresses were made in what concerns the artificial life. In terms of Rodney Brooks ("The relationship between matter and life", *Nature*, vol. 409, January 2001, pages 409–411), this suggests that "we might be missing something fundamental and currently unimagined in our models of biology". Computers are good in crunching numbers, but "not good at modeling living systems, at small or large scale". The intuition is that life is more than biophysics and biochemistry, but what else it is can be something which we cannot imagine today, "some aspects of living systems which are invisible to us right now". "It is not completely impossible that we might discover some new properties of biomolecules or some new ingredient". An example of such a "new stuff", R. Brooks says, can be the quantum effects from the microtubules of the neural cells, which, according to Penrose, "might be the locus of consciousness at the level of the individual cell" (citation from R. Brooks).

A similar opinion was expressed by another great name of the artificial intelligence, John McCarthy ("Problems and projection in CS for the next 49 years", *Journal of the ACM*, vol. 50, 2003, pages 73–79): "Human–level intelligence is a difficult scientific problem and probably needs some new ideas. These are more likely to be invented by a person of genius than as part of a Government or industry project".

Anyway, the progresses related to the collaboration between computer science and biology should not be underestimated. If we do it, then we take a risk which has struck big names of science and cultures. I close with a funny example of this kind, some statements (dated around 1830) of the French philosopher Auguste Comte: "Every attempt to employ mathematical methods in the study of biological questions must be considered profoundly irrational and contrary to the spirit of biology. If mathematical analysis should ever hold a prominent place in biology – an aberration which is happily almost impossible – it would occasion a rapid and widespread degeneration of that science."

Thanks to God, the philosopher was wrong – but we needed about two hundred years to see that...

40 (Provisory) Last Words

I hope that this quick description was convincing in showing that the way from biology to computer science and back to biology is intellectually fascinating and useful to both sciences.

A few things should be remembered: (i) in all its history, computer science tried to learn from biology, (ii) and this effort brought important benefits to computer science and equally to biology; (iii) the progresses in this area should not be underestimated, (iv) but, in general, it is plausible that we expect too much (and too fast) from the computer science-biology symbiosis, (v) because we ignore the essential differences between the two universes, the inherent limits of computability and the fact that biology is not a mathematically formalized science, (vi) with the mentioning that it is possible to need a new mathematics in order to model and simulate life and intelligence; finally, (vii) let me anticipate a new age of biology, beyond the today bioinformatics and the today natural computing, and let me also propose a name for it, *infobiology*.

Should we wait two further decades in order to see it taking shape?

From an intellectual point of view, during the forty years which I have told about here I have lived around academician Solomon Marcus, a "big tree" which invalidates the phrase ("In the shadow of big trees not even the grace is growing.") by which Constantin Brancusi motivated his decision to refuse to work under the guidance of Rodin: professor Solomon Marcus never puts shadow on his numerous students and collaborators, but on the contrary. I repeat, in order to stress it: on the contrary. I witness this and I dedicate to him this discourse, thanking him once again.