Available online at www.sciencedirect.com

SCIENCE DIRECT®

BioSystems 77 (2004) 175–194

www.elsevier.com/locate/biosystems

Bio-steps beyond Turing

Cristian S. Calude[a,*], Gheorghe Păun[b,c]

[a] Department of Computer Science, The University of Auckland, Bag 92019, Auckland, New Zealand
[b] Institute of Mathematics of the Romanian Academy, P.O. Box 1-764, 014700 Bucuresti, Romania
[c] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Sevilla University, Avenida Reina Mercedes s/n, 41012 Sevilla, Spain

Received 2 December 2003; received in revised form 10 May 2004; accepted 10 May 2004

Abstract

Are there 'biologically computing agents' capable to compute Turing uncomputable functions? It is perhaps tempting to dismiss this question with a negative answer. Quite the opposite, for the first time in the literature on molecular computing we contend that the answer is not *theoretically negative*. Our results will be formulated in the language of membrane computing (P systems). Some mathematical results presented here are interesting in themselves. In contrast with most speed-up methods which are based on non-determinism, our results rest upon some universality results proved for deterministic P systems. These results will be used for building "accelerated P systems". In contrast with the case of Turing machines, acceleration is a part of the hardware (not a quality of the environment) and it is realised either by *decreasing the size of "reactors"* or by *speeding-up the communication channels*. Consequently, two acceleration postulates of biological inspiration are introduced; each of them poses specific questions to biology. Finally, in a more speculative part of the paper, we will deal with Turing non-computability activity of the brain and possible forms of (extraterrestrial) intelligence.

Keywords: Turing; Bio-steps; P systems

1. Introduction

For more than 50 years the Turing machine model of computation has defined what it means to "compute" something; the foundations of the modern theory of computing are based on it.

Furthermore, as it has been noted by various authors (see, for example, Calude and Casti (1998)), the (silicon) computer, whose capacity for handling information has been growing at a rate 10 million times faster than information handling did in our nervous system during the more than 600 million years of evolution, seems to be the only important commodity ever to become exponentially better as it gets cheaper. However, this exponential race is essentially "Turing-bounded", it cannot produce feasible solutions to many intractable problems and, more importantly for our investigations here, it cannot solve Turing unsolvable problems.

Hypercomputation or super-Turing computation is a "computation" that transcends the limit imposed by Turing's model. For a recent perspective one can consult the special issues of the journal *Minds and Machines*, (Copeland, 2002); see also Ord (2002) for a lucid analysis, hypercomputation website for a comprehensive bibliography and the section 'Computation

* Corresponding author.
E-mail addresses: cristian@cs.auckland.ac.nz (C.S. Calude), george.paun@imar.ro, gpaun@us.es (G. Păun).

and Turing machines' in Teuscher (2003) (especially the critical paper by Davis (2004)).

Are these studies just mere idle speculations, pure theoretical abstractions studied for their own sake, with little or no regard to the 'real world'? We dare to answer this rhetorical question in the negative. First, according to Casti (2003), "*If* science really is essentially the carrying out of a calculation, then the limits of science are necessarily extended whenever we extend our computational capabilities". Promises are very high: if materialised, science will reach points that have never been seen before. But, what about the case of failure? Even in this case the gain will be also immense (and, arguably, scientifically more interesting), as negative results will reveal new *limits*, with far-reaching implications in mathematics, computer science, physics, biology, and philosophy.

To date, all suggestions to transcend the Turing barrier are, as far as we know, either purely mathematical, or based on (quantum or relativistic) physics – some details can be found below – and no serious attempt has been made to propose a 'biological computing agent' having super-Turing capability. This is one of the aims of this paper.

We will formulate our problems and solutions in the language of *membrane computing*, a branch of molecular computing (initiated in Păun (2000)) whose goal is to abstract computing models from the structure and functioning of the living cell; see Păun and Rozenberg (2002) for an introduction and Păun (2002), Calude and Păun (2001) for more comprehensive presentations. The basic idea is to find classes of membrane systems (P systems) which (1) allow deterministic characterisations of Turing computability, and (2) "support" in a natural way *acceleration postulates* with a direct biological inspiration[1].

The paper is organised as follows. In the next section we will present the motivation, the models and results in an informal way. The third section is devoted to the main results and complete proofs. The fourth section speculates on some implications of our results. We conclude the paper with some provisional conclusions and open problems. Readers less interested in all technical details may skip Sections 3.1 and 3.4.

[1] We will systematically avoid saying that one model or another, one feature or another from a given model is more or less "realistic" from a biological point of view, but rather we will try to get *inspired* from biology, or to have our speculations *motivated* by biology.

2. Problems and solutions: an informal discussion

There are various computing models whose capabilities go beyond Turing's, that is, they have super-Turing power: they exploit time acceleration ((Russell, 1936; Blake, 1926; Weyl, 1927; Stewart, 1991; Copeland, 1998; Svozil, 1998)), physics ((Etesi and Németi, 2002; Calude and Pavlov, 2002; Kieu, 2001, 2002; Adamyan et al., 2004) use quantum theory, Etesi and Németi (2002), Wiedermann and Leeuwen (2002) use relativity theory), neural networks (Siegelmann, 1995, 2003) and many other methods (see Ord (2002) for an overview), but *there are no biological models*. See Casti (1997), Calude and Casti (1998), Chown (2002), Teuscher and Sipper (2002), Copeland (2002), Delahaye (2003) for more general discussions.

2.1. Acceleration

In all approaches, the main problem is *to carry on an infinite computation in a finite amount of time*. The general scenario is suggested in Fig. 1: we have two scales of time, an *external*, *global* one, of the "user" of the accelerated device (the black box in the figure), and the *internal*, *local* time of the device. The problem is formulated in global time, at some moment t, and introduced into the accelerated device, which is able to perform an 'inner' infinite computation in a finite number, T, of external time units, when the "user" gets the answer to the problem.

The idea came from Russell (1936), Blake (1926), Weyl (1927) who observed that a process that performs its first step in one unit of (global) time, the second

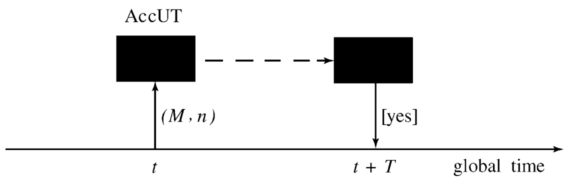

Fig. 1. The interplay between local and global time used for solving the Halting Problem by means of an accelerated device.

step in 1/2 unit of (global) time, and, in general, each subsequent step in half the (global) time of the step before, is able to complete an infinity of steps in just two global units of time since $1 + (1/2) + (1/4) + \ldots = 2$. A universal Turing machine working in this kind of accelerated manner is capable of deciding the Halting Problem (i.e., the problem whether an arbitrary Turing machine halts on an arbitrary input), the most (in)famous Turing undecidable problem, in the way suggested in Fig. 1: at some (global) time t one introduces (the code of) any particular Turing machine M and an input n into the accelerated universal Turing machine (AccUT), and in two (global) time units we have the answer, yes or no, whether $M(n)$ halts or not (here, $T = 2$).

We note that acceleration does not conflict with the Turing model of computation as the mathematical definition of a Turing machine does not specify how long does it take to perform an individual step. Even more, Svozil (1998) has shown that no known physical law forbids such an acceleration[2] (a quantum analysis of the phenomenon reveals some type of uncertainty which, in a sense, diminishes the value of the model, see Svozil (1998)). We also mention that time acceleration is qualitatively different from ‘time travel’, which is an ingredient not allowing super-Turing capabilities, cf. Calude et al. (2000). An accelerated Turing machine is simply a classical Turing machine (a piece of classical hardware) working in an environment allowing acceleration. Whether this is possible or not is a matter of physics!

We take here a different approach, grounded on suggestions coming from cell and brain biology: acceleration is a part of the hardware (not a quality of the environment) and it is realised either by *decreasing the size of “reactors”* (thus making possible that reactants find each other and react in a shorter time), or by *training*, by *speeding-up the communication channels*[3].

[2] Special relativity does not forbid such a scenario as it allows objects to exceed the speed of light as long as they never slow down below it.

[3] Ultimately, these hypotheses should be validated or not by biophysics. However, here we place ourselves in an idealized framework, where, for instance, there is no limit of acceleration, of decreasing the size, hence we can perform arbitrarily many steps of acceleration.

2.2. A glimpse to membrane computing

The above two hypotheses can find a natural framework for a formulation and (mathematical) investigation in terms of *membrane computing*.

Membrane computing is a branch of natural computing whose aim is to abstract computational models, ideas, paradigms from the living cell (neurons as particular cases) structure, functioning, and inter-relationships in tissues, organs, organisms. The approach is very general, and starts from two (optimistic) observations: (i) there are several biologically inspired branches of computer science (for example, genetic algorithms, or more generally, evolutionary computing) which prove to be unexpectedly efficient, and the most plausible “explanation” of this efficiency seems to be the fact that they use – at an abstract, computational level – operations, tools, processes which have been used and improved by nature for billions of years; (ii) the cell is a marvellous tinny machinery, the smallest living thing, with a complex structure,an intricate inner activity, and an exquisite relationship with its environment. In this context, the challenge is to find in the cell the ingredients/features which could be useful to computing.

The proposed answer of membrane computing to this challenge starts from the fundamental fact that cells are “defined” by a hierarchical structure of membranes, from external plasma membrane to many inner vesicles. The main role of biological membranes is to *separate and protect an ‘inside from an outside’*, making possible the communication between the two regions in certain circumstances only. Membranes are at the same time *borders* and *selective channels of communication*. For instance, it is the cell external membrane which simply gives identity to the cell, keeps nutrients inside and ejects waste products, protects the cell against “wrong chemicals” (by closing the protein channels towards the environment), and so on. The internal compartmentalisation of a cell, through the many membranes existing there, is crucial for the cell functioning. For Kauffman (1993), “the secret of life” is to be found “in the achievement of collective catalytic closure”, while for Hoffmeyer (1998) “membranes are the primary organisers of multi-cellular life”. Marcus (2004) synthesises all these in a slogan: “Life is DNA software + membrane hardware.”

For our aim it is relevant to note that one of the functions of membranes is to separate compartments which are small enough for the chemicals which swim there in water solution to find each other in a small enough time and react. The reactions taking place in a cell compartment depend on many local conditions (promoters, inhibitors, acidity, temperature, and so on), but the number of collisions per time unit of reactants (collisions driven by Brownian motion), which is, roughly speaking, directly proportional to the number of copies of each reactant present in each volume unit (hence inversely proportional to the volume) has a direct influence on the reaction rate. Hence, *smaller* (compartments) *means faster* (reactions).

Let us now pass from single cells to multi-cellular structures. Two neighbouring cells can "communicate" either directly, or through the environment. The main way of carrying the direct transfer of chemicals between adjacent cells is through common protein channels; in general, the passage of materials is made/controlled by proteins embedded in membranes which let chemicals to pass from a region to another one selectively, depending on size, polarisation, type, etc. A special type of trans-membrane transport appears when two, or several, chemicals pass only together through a protein channel, either in the same direction (such a process is called *symport*) or in opposite directions (called *antiport*).

Communication among compartments and cells is crucial for the functioning of cells and multi-cellular structures; it makes the difference from a population of separated agents to a *system*. That is why we may speculate that, at least in certain circumstances, *nature is interested in speeding-up the communication among compartments of the same cell and among different cells*. In the special – and rather important – case when cells are neurons, and communication is done via synapses, this is no longer just a speculation: *more frequently used sequences of synapses become faster and more efficient*.

These hypotheses are essential for the anatomy and behaviour of three classes of membrane systems (P systems) called *cell-like*, *tissue-like*, and *neural-like* P systems. We briefly introduce them here, for specifying the framework in which we work in; a more technical presentation will be given in Section 3.3.

The main ingredients of a cell-like P system are (1) the hierarchical arrangement of *membranes*, which delimit compartments (also called *regions*), where (2) *multisets of objects* evolve according to (3) *evolution rules*. The compartments are uniquely associated with membranes; Fig. 2 introduces the relevant notions related to a cell-like membrane structure. The objects and evolution rules are localised, associated with compartments. The objects correspond to chemicals and the rules correspond to the reactions which can take place in cell compartments. The objects are unstructured (atomic, in the etymological sense), hence they are represented by symbols from a given alphabet. In each compartment, we have a given number of copies

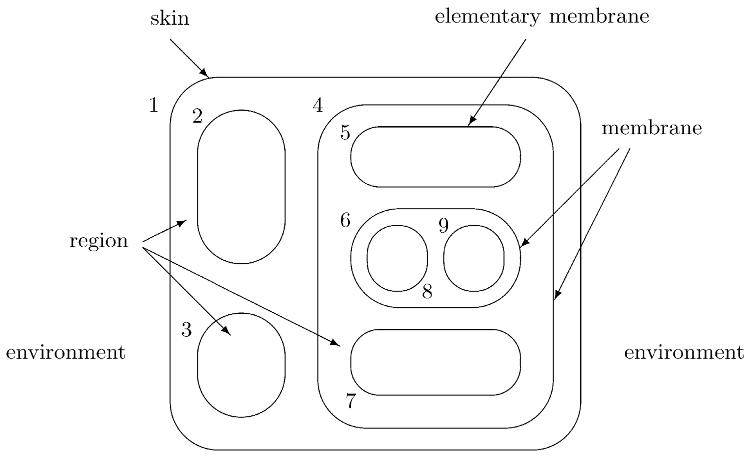

Fig. 2. A cell-like membrane structure.

of each object, hence a multiset (a set with multiplicities assigned to its elements). Thus, the evolution rules are multiset processing rules. The basic type of rules, modelling the chemical reactions, are of the form $u \to v$, where u and v are strings of symbols representing multisets of objects (the number of occurrences of a letter in a string gives the multiplicity of the respective object in the multiset). These rules can be classified according to the number of objects from their "antecedent" u in *cooperative* rules, those having at least two objects in u, and *non-cooperative* rules, those with u consisting of a single object. Applying such a rule means to "consume" the objects specified by u and to produce the objects specified by v. The objects from v can also be associated with target indications; for instance, an object which appears in v in the form a_{out} leaves the region where the rule is applied and becomes an element of the multiset placed in the surrounding region – which is the environment in the case of the skin region; similarly, an object a_{in} has to enter any one of the immediately inner membranes, non-deterministically chosen. In this way, compartments communicate to each other, and the system can send objects (hence signals) outside. The rules in each compartment are used in the maximally parallel manner: in each step of a computation, all objects which can evolve together will evolve. The objects and rules are non-deterministically chosen (observing the maximality restriction). In this way, we get transitions from a configuration of the system to the another. A sequence of transitions is a computation. A computation which reaches a configuration where no rule can be applied is said to be halting.

Basically, such a computing machinery can be used in two ways: (1) as a *generative device* (start from the initial configuration and collect all outputs – defined in a suitable manner – associated with all halting computations), and (2) as a *functional device* (introduce an *input* in a specified membrane and associate with it an *output*, in the end of a halting computation). An important particular case of the latter possibility is to use a P system as an *accepting* device, computing the characteristic function of a set (introduce an input in the system and say that the input is accepted/recognised if the system eventually halts and a special object, `yes`, is ejected from the system).

There are many variants of such systems, containing further features inspired from biology: priorities among rules, promoters or inhibitors for rules, membrane creation or membrane division rules, etc. Most of these types of P systems are equal in power to Turing machines; when possibilities to generate an exponential workspace in linear time (e.g., by membrane division) are provided, then computationally hard problems (typically, NP-complete) can be solved in polynomial (often linear) time.

2.3. Computing by communication

A rather important class of P systems uses only communication (based on symport/antiport rules) for computing. A symport rule is of the form (x, in) or (x, out), with the meaning that the objects specified by x enter, respectively exit, the region with which the rule is associated; an antiport rule is of the form $(x, \text{out}; y, \text{in})$, specifying that the objects of x exit and those of y enter the respective membrane.

The symport/antiport rules can be used both for cell-like and tissue-like systems. In the latter case, the membranes are not hierarchically arranged – hence in the nodes of a tree - but they are placed at the same level, like in a tissue, and establish a communication graph depending on the available communication rules. Fig. 3 suggests the shape of a membrane structure in the case of a tissue-like P system; note that some cells (always having only one membrane) communicate with each other directly, in a one-way or a two-way manner, while one cell (namely, 4) can communicate with the other three only through the environment.

Taking into account that synapses are one-way channels of communication, the systems where the communication among the cells is one-way (if a cell i

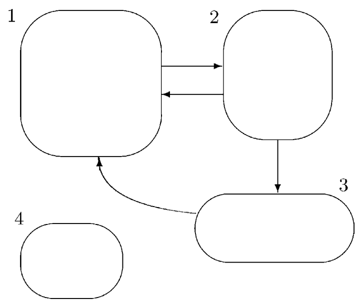

Fig. 3. A tissue-like membrane structure.

sends objects to cell j, then never cell j sends objects to cell i) are called neural-like.

Restricted (in the number of membranes and/or the size of multisets x, y from the symport/antiport rules) systems of both types, cell-like and tissue-like, computing only by communication are *universal*, i.e., equal in computational power to Turing machines. From the proofs of such results, one can see that the same assertion holds for neural-like systems as well[4].

Details about various classes of P systems can be found in Păun (2002); a comprehensive image of the domain can be found at P systems website.

Most, if not all, universality results known in the membrane computing literature deal with non-deterministic systems. Actually, deterministic systems, those where from each configuration there is at most one possible transition, do not make much sense for systems which work in the generative style starting from a given initial configuration: they can generate at most singleton sets of numbers. Because we want to solve decision problems, in particular the Turing Halting Problem, we have to deal with deterministic accepting systems.

2.4. Sources of acceleration

To transgress the Turing barrier we will use two bio-inspired sources of acceleration.

Specifically, starting from the postulate that *smaller is faster*, we may imagine that each membrane of a cell-like P system has its local time, and *the time unit is strictly smaller from a level of the membrane structure to the next level below*. Namely, the relation between the time unit t on any level i to the time unit $f(t)$ on level $i+1$ (hence to $f^2(t)$ on level $i+2$, etc.) is given by a Turing computable function f such that $\sum_{i=1}^{\infty} f^i(t) \leq T$, for a given constant T. Such a system is called (f, T)-*accelerated*. For example, we can take $f(t) = t/2$, or $f(t) = t/q$, where q is a rational number greater than 1, or $f(t) = t^{1/2}$ with $t > 1$. The rate of change $\lim_{t\to\infty}(f^i(t))/(f^{i+1}(t))$ has to be small (as close to 1 as possible). Actually, the rate is 2 if $f(t) = t/2$ and 1 if $f(t) = t^{1/2}$. If $f(t) = t^{1/2}, t = 2$, then the ratio $f^i(t)/f^{i+1}(t)$ decreases from 1.18 at $i = 2$, to 1.04 at $i = 4$ to reach 1.0001 at $i = 12$; quite soon the acceleration becomes extremely small.

Under the above assumption we will show that we can construct a cell-like P system Π, with multiset rewriting-like rules, which can solve the Halting Problem. More precisely, we will construct an (f, T)-accelerated Π system which in time $9T + 3$ determines whether a register machine (an equivalent way to present a Turing machine) stops on an arbitrary input. The P system Π works as follows: Given any register machine M and input n, the (f, T)-accelerated recognising P system Π (using membrane creation in order to increase the depth of the membrane structure – as well as other ingredients, such as membrane dissolution), will simulate the computation of M on n for at most $9T + 3$ units of global time: if M halts on n, then Π sends to the environment the special object `yes` (in global time at most $9T+3$), but if M does not stop on n, then Π sends nothing outside, so in global time $9T + 3$ we know that M will never stop on n.

The postulate *faster by training/repeating* is closer to the style of accelerated Turing machines: we simply assume that the step t takes a time $f(t)$, for a function f as above, and then, with the scenario illustrated in Fig. 1, we can solve the Halting Problem in T units of global time. This holds both for tissue-like (two-way communication) and for neural-like (one-way communication among cells) P systems.

What is crucial in all these constructions – and not very common in membrane computing up to now – is that we need to have a *universal P system of the given type which is deterministic*.[5]

The class of problems solved by these three types of accelerated P systems is exactly the class Σ_1, the class of predicates of the form "there exists x such that $R(x, y)$", where R is a Turing computable predicate. Is it a big step? Yes, and here are two examples which justify our answer: (a) such a P system is capable to decide whether an arbitrary Diophantine equation has

[4] Note that the above notion of "neural-like" P system is different from that in Păun (2002), where also states are associated with the "neurons", and both the processing of objects and the communication is done in a multiset rewriting style. There is also a clear and major difference between what we call here "neural-like systems" and the neural networks from neural computing.

[5] For the reader familiar with the current proofs in membrane computing, this is a strong restriction, because the usual techniques for avoiding "wrong" computations by turning them into non-halting computations cannot be used any more, hence new proof techniques must be found.

or not a solution (i.e., it can solve Hilbert's Tenth Problem, a famous Turing undecidable problem), (b) can establish the consistency of any formal system ("avoiding" Gödel's Incompleteness Theorem[6]). See more in Calude et al. (2000).

3. Models and technical results

In this part we describe our models (with partially formal details) and solutions (with all technical details).

3.1. Overview of methods and results

In our framework, we need P systems which work in a deterministic manner. A general slogan of parallel computing is that "parallelism can simulate non-determinism" – but this is not always a trivial matter, for instance, because in general it requests to use an exponential space, which should be systematically explored. Here we do not follow this idea, but we directly look for ways to construct P systems which behave deterministically. We have already noted that usually in membrane computing area one does not pay (too much) attention to determinism. Specifically, in most – if not all – proofs dealing with the computing power of generative P systems one uses non-deterministic systems. Deterministic systems appear in several papers on accepting P systems dealing with complexity matters (see Ibarra (2004, in press), the corresponding chapter from Păun (2002) or the book (Pérez-Jiménez et al., 2002)); sometimes, in this case one also allows a certain degree of non-determinism, providing the system is *confluent* (after a phase which we do not care about, the computations should converge to a common path, which always halts).

Here we first address the problem of finding a universality proof for cell-like P systems, working in the analysing mode with multisets of symbol-objects, processed by rewriting-like rules. Actually, we consider systems with membrane creation, able to produce new membranes, from a given list, by rules of the form $a \to [v]_i$, where a is an object, v is a multiset of objects, and the newly created membrane has the label i; the fact that we know the label is important, because in this way we know which rules are available in the region delimited by this new membrane. It is known (see Theorem 7.3.1 from Păun (2002)) that generative P systems using membrane creation and even membrane dissolving features, using only non-cooperative rules characterise the length sets (Parikh sets, if we want to work with vectors and we distinguish the objects in the output) of ET0L languages, hence they are not universal. The proof in Păun (2002) is, of course, based on a non-deterministic P system which simulates any given ET0L system. Universality can be obtained at the price of using priorities among rules or when controlling the permeability of membranes (Corollary 7.3.1 from Păun (2002)), but again non-deterministic systems are essential. Universality can be also obtained using cooperative rules (Theorem 3.3.3 from Păun (2002)), even without using membrane creation rules, but with non-deterministic systems once again.

Our first result in this area combines the two powerful features, cooperating rules and membrane creation (as well as membrane dissolution), in proving the universality of deterministic analysing P systems. Membrane creation can increase the membrane structure of a given system, adding further and further new membranes; by definition, these new membranes are elementary, hence in this way we can increase continuously the depth of the membrane structure. Then, we prove the universality of deterministic analysing P systems of cell-like and tissue-like P systems using symport/antiport rules; the latter case has as consequence the universality of neural-like systems.

All these universality results are then used for proving that in the accelerated mode, each of the respective systems can solve the Halting Problem.

3.2. Turing computability

Classical computability can be presented in a variety of equivalent mathematical formalisms which include Turing machines, Markov algorithms, partial recursive (computable) functions, Chomsky type-zero grammars, register machines, etc. In what follows we will use Minsky register machines (Minsky, 1967). Such a device consists of a given number of *registers* each of which can hold an arbitrarily large non-negative integer number, and a *program* which is

[6] A note of caution: the existence/plausibility of any super-Turing machine affects in no way the validity of Gödel's Incompleteness Theorem.

a sequence of labelled instructions which specify how the numbers stored in registers can change and which instruction should follow after any used instruction. There are three types of instructions:

- l_1: (ADD(r), l_2) (add 1 to register r and go to the instruction with label l_2),
- l_1: (SUB(r), l_2, l_3) (if r is non-empty, then subtract 1 from register r and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- l_h: HALT (the halt instruction).

Thus, formally, a *register machine* is a construct $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and P is the set of instructions (the program).

A register machine can be used for recognising (one also says "accepting") numbers in the following sense: we designate one of the registers, say, register 1, as the input register; we input a number n in this register, and let the machine start computing with all other registers holding 0, from the instruction with label l_0; if the computation halts, that is, it reaches the instruction with label l_h, then n is recognised; otherwise, n is not recognised. The set of all non-negative integers recognised by a given register machine M is denoted by $N(M)$.

A register machine can also compute a function by designating certain registers as input registers and others as output registers. We start with the arguments of the function in the input registers and, providing that the machine halts, we get the value of the function in the output registers; if the computation never halts, then the function is not defined for the given input. The partial functions computed in this way coincide with the *Turing computable* partial functions (see Minsky (1967)). In particular, a set of positive integers A is *Turing computably enumerable* if its partial characteristic function ($\chi_A(n) = 1$, if $n \in A$, and $\chi_A(n) =$ undefined if $n \in A$) is Turing partial computable. We denote by *NCE* the family of Turing computably enumerable sets of numbers.[7]

In this framework, the *Halting Problem* is the problem to decide whether an arbitrary register machine M halts on an arbitrary positive integer n. A fundamental result due to Turing states that the *Halting Problem is Turing undecidable*, that is, there is no fixed Turing machine (hence neither a fixed register machine) M_u capable of deciding whether an arbitrary register machine M halts on an arbitrary positive integer n.

The class of predicates which can be written in the form $(\exists x)\, R(x, y)$, where R is a Turing computable predicate, is denoted by Σ_1. Many important problems can be expressed by predicates in Σ_1; for example, the predicate stating that a Diophantine equation, an equation of the form $P(x_1, \ldots, x_n) = 0$ (where P is a polynomial with integer coefficients) has a solution or not in positive integers is in Σ_1.

3.3. Membrane computing

We introduce here, rather informally, the classes of membrane systems we will consider in this paper. With the intuition given in Section 2, such an informal presentation should be sufficient for understanding the arguments from the subsequent sections.[8]

We first introduce the cell-like P systems with rewriting-like rules, using membrane creation and membrane dissolution features – as we will need below.

A *cell-like P system* (of degree $m \geq 1$) is a construct $\Pi = (O, H, \mu, w_1, \ldots, w_m, R_1, \ldots, R_n)$, where O is a finite alphabet whose elements are called *objects* (actually, we will use as synonymous the terms "symbol" and "object"), $H = \{h_1, \ldots, h_n\}$ is a set of labels for the possible membranes of Π, μ is a membrane structure of degree m (represented as a string of correctly matching parentheses, with a membrane with label i represented by a pair of parentheses of the form $[\,]_i$), w_i is a string[9] over O representing the multiset of objects present in region i of μ, for $1 \leq i \leq m$, and R_j is

[7] The notation comes from the fact that these sets of numbers are the length sets of computably enumerable languages, which are the languages whose strings are recognised by Turing machines, or, equivalently, generated by Chomsky type-zero grammars. An alternative, older notation, was *NRE*, coming from recursively enumerable languages.

[8] Of course, some familiarity with membrane computing would be helpful, especially when dealing with technical details in proofs; we refer the interested reader to the many sources of information in this respect, starting with the monographs (Calude and Păun, 2001; Păun, 2002), and the comprehensive web page already mentioned in Section 2.

[9] The set of all strings over O, the empty string λ included, is denoted by O^*, while the set of non-empty strings is denoted by O^+; by $|x|$ we denote the length of a string $x \in O^*$.

the set of rules which act in the region of membrane j, for $1 \leq j \leq n$. These rules can be of two forms:

- *Multiset processing rules*: $u \to v$ or $u \to v\delta$, where $u \in O^+$, $v \in (O \times TAR)^*$, for $TAR = \{here, out, in\}$, and δ is a special symbol. The use of such a rule means "consuming" the objects indicated by u, and introducing the objects specified by v, in the regions indicated by the targets associated with these objects, in the way already informally presented in Section 2. If δ is present, then the membrane where the rule was applied is dissolved. The contents of the dissolved membrane, objects and inner membranes alike, are added to the contents of the immediately upper membrane.

  A rule $u \to v, u \to v\delta$ is said to be cooperative if $|u| \geq 2$ and non-cooperative if $|u| = 1$.
- *Membrane creation rules*: $a \to [v]_h$, with $a \in O, v \in O^*$, and $h \in H$. By means of such a rule, an object a creates a membrane with label h, with the objects specified by v inside. Knowing the label of the membrane, we also know the rules which can act in its region, namely, the set R_h.

Note that membrane creation rules are non-cooperative, while the multiset processing rules can be cooperative. In general, the target *here* is not mentioned, while *in* and *out* are given as subscripts of the objects they are associated with. All rules are used in a maximally parallel way, chosing the objects to evolve and the rules in a non-deterministic way.

A configuration of the system at a given step is identified by the membrane structure and the objects present in each region at that step. If in each configuration of the system we can have at most one choice of rules to apply, hence either the system halts or the next configuration is uniquely determined, then the system is *deterministic*.

In what follows, we use P systems in the *accepting* mode: we have the system in the form specified above, and we *input* a multiset w to it, placing the objects of w in the skin region; if the computation started after introducing w eventually halts, then we say that w is accepted, otherwise w is rejected. In particular, we can have $w = a^i$, for a specified object a, and then the number i is accepted or not by the system. If we want that the case of accepting an input is also explicitly announced to the environment, then we may also ask that a special object yes will be sent out from the system if and only if the input is accepted (and this happens only once, at the end of the computation).

The set of non-negative integers accepted in the above sense by halting computations by a system Π is denoted by $N(\Pi)$, and the family of all such sets accepted by deterministic systems is denoted by $DN_aOP(coo, mcre, diss)$: *coo, mcre, diss* indicate the use of cooperative rules, membrane creation, and membrane dissolving features[10].

The multiset processing rules as above correspond to reactions which take place in the compartments of a cell. Of a crucial importance is the possibility to transfer objects through membranes - thus integrating the separate "computing agents" from compartments in a global "computer".

Actually, one can compute – even at the level of Turing machines (and beyond in certain circumstances, as we will see below) – by communication only, for instance, by making use of symport and antiport, well-known in biology (see Alberts et al. (2002)). Such rules can be used both in cell-like systems (with the membranes hierarchically arranged) and tissue-like systems (with the membranes placed in the nodes of an arbitrary graph).

A *cell-like P system* (of degree $n \geq 1$) with symport/antiport is a construct of the form $\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m)$, where O is an alphabet of objects, μ is a membrane structure (of degree m, with the membranes injectively labelled with $1, 2, \ldots, m$), w_i is the multiset of objects present in region i, R_i is the set of rules associated with membrane i, for $1 \leq i \leq m$, and $E \subseteq O$ is the set of objects assumed to be present in the environment in arbitrarily many copies. The rules are now associated with membranes, and can be of two forms:

- *Symport rules*: (x, in) or (x, out), where $x \in O^+$. By using such a rule, the objects specified by x are brought in, respectively sent out of the region of the membrane the rules are associated with.
- *Antiport rules*: $(x, \text{out}; y, \text{in})$, for $x, y \in O^+$. Applying such a rule for membrane i means to send out of the membrane the objects specified by x and bring

[10] The notation DN_aOP comes from Deterministic P systems working with Objects represented by symbols (in the literature there also are investigated P systems working with string-objects), Accepting sets of Numbers.

in (from the adjacent external region, which can be the environment in the case of the skin membrane) the objects specified by y.

Starting from the initial configuration – with an input which has to be introduced in the skin region – we pass from a configuration to another one by using the rules in the maximally parallel manner. If the choice of rules to apply is unique in each step, then we speak of a *deterministic system*. The input is accepted if and only if the computation stops.

We denote by $DN_aOP_m(sym_s, anti_{t_1,t_2})$ the family of sets of numbers accepted by deterministic cell-like P systems with at most m membranes (note that we do not have here membrane creation or dissolving), with symport rules (x, in), (x, out) having $|x| \leq s$ and antiport rules $(x, \text{out}; y, \text{in})$ having $(|x|, |y|) \leq (t_1, t_2)$.

Computing by communication can only be performed also with the membranes placed in the nodes of an arbitrary graph – and this corresponds both to inter-cellular communication, much investigated in biology (see, e.g., Loewenstein (1999)), and to communication in networks of neurons.

A *tissue-like P system* (of degree $m \geq 1$) is a construct $\Pi = (O, w_1, \ldots, w_m, E, R)$, where all components are as above, and R is a finite set of rules of the following forms:

- *Symport rules*: (i, x, j), for $x \in O^+$, and $1 \leq i, j \leq m, i \neq j$, with at most one of i, j being 0. By using such a rule, the multiset of objects specified by x is moved from region i to region j, where 0 identifies the environment.
- *Antiport rules*: $(i, x/y, j)$, for $x, y \in O^+$ and $1 \leq i, j \leq m, i \neq j$, possibly with $j = 0$. Using such a rule means that the objects of x are sent from region i to region j, at the same time with sending the objects of y from region j to region i; again, 0 identifies the environment.

The use of rules (maximally parallel) and the definition of computations follow the same pattern as for cell-like systems. This time, the input to be recognised by the system is introduced in a specified membrane – for instance in that with label 1. We denote by $DN_aOtP_m(sym_s, anti_{t_1,t_2})$ the family of sets of numbers recognised by deterministic tissue-like P systems with at most m membranes, using symport and antiport rules with the size of the involved multisets at most s and (t_1, t_2), respectively.

In systems as above, two cells can communicate in a two-way manner, either through antiport rules, or through complementary symport rules. In an attempt to get closer to the neural case, where the communication is, in general, one-way, through the axon of a neuron to the dendrites of another neuron, we can distinguish a restricted class of tissue-like P systems, which we will call here *neural-like* P systems, where the communication between any two cells (we also call them "neurons") is one-way: for any two cells i, j, we always either send objects from i to j (hence only rules of the form (i, x, j) are available), or only from j to i (only rules (j, x, i) are present). The communication with the environment remains unrestricted. The corresponding family of sets of non-negative integers accepted – in the deterministic case - is denoted by $DN_aOnP_m(sym_s, anti_{t_1,t_2})$, with the obvious meaning of the parameters involved.

All the above classes of systems, cell-like, tissue-like, and neural-like, working in the generative mode are known to be universal. In most cases, systems with a rather low number of membranes and rules of a restricted type suffice. However, most if not all the proofs of such results (all those from Păun (2002) at least) are based on non-deterministic systems. Choices are not only allowed but also essentially used in proofs: the correct simulation of a grammar or of a register machine is guessed during a computation of the P system aimed to simulate that grammar or register machine, and "wrong" guesses are turned to non-halting computations.

For our purpose here, such a "computation" is of no use, hence deterministic characterisations of *NCE* are looked for. Fortunately, and somewhat surprising, such results *can be proved* for all classes of systems introduced above – each time with some price to pay in the complexity of the system.

3.4. The price of determinism

The results below are both of interest for membrane computing in general (proving universality for deterministic P systems of various types), but also essential for the results proved in Section 3.5, the main goal of our investigation.

Theorem 1. $NCE = DN_aOP(coo, mcre, diss)$.

Proof. Let us consider a (deterministic) register machine $M = (m, B, l_0, l_h, P)$ as specified in Section 3.2. For each register i we consider a distinct symbol a_i, $1 \leq i \leq m$; let U be the set of all these symbols. We construct the (recognising) cell-like P system

$$\Pi = (O, \{1, 2\}, [\,]_1, l_0c, R_1, R_2),$$

with

$$O = U \cup \{l, l', l'', \bar{l}, \hat{l} | l \in B\} \cup \{c, d\},$$

and the following sets of rules:

$$\begin{aligned} R_1 = \ & \{c \to [c]_2\} \\ & \cup\{\alpha \to \alpha_{in} | \alpha \in B \cup U\} \\ & \cup\{l' \to d, \bar{l} \to d | l \in B - \{l_h\}\}, \\ R_2 = \ & \{l_1 \to l_2a_r | \text{ for } l_1 : (\mathsf{ADD}(r), l_2) \in P\} \\ & \cup\{l_1 \to l'_1l''_1, \\ & l'_1a_r \to \bar{l}_2, \\ & \bar{l}_2 \to l_2\delta, \\ & l''_1 \to \hat{l}_3, \\ & \hat{l}_3 \to \bar{l}_3, \\ & \bar{l}_3 \to l_3\delta | \text{ for } l_1 : (\mathsf{SUB}(r), l_2, l_3) \in P\}. \end{aligned}$$

Let us assume that we introduce n copies of the object a_1 in the system Π (which corresponds to starting the work of the register machine M with the number n in the first register). The system Π simulates the work of the register machine when analysing the input n.

In the initial configuration we have the multiset $l_0ca_1^n$ in the unique membrane, that with label 1. Assume that we are at a time where we have in this membrane an arbitrary multiset of objects from U, as well as the object c and an object $l_1 \in B$.

No rule can be applied, except for $c \to [c]_2$, which creates a membrane with label 2 (and with the object c inside). In the next step, all objects from the skin membrane are introduced in the inner membrane. Here we perform the simulation of rules from P.

The ADD-instructions of the form $l_1 : (\mathsf{ADD}(r), l_2)$ are directly simulated by the rules $l_1 \to l_2a_r$ present in R_2.

If l_1 is the label of a SUB-instruction, then we proceed as follows. In the first step, l_1 is replaced by l'_1 and l''_2. In the next step, l'_1 performs the subtraction, providing that at least one copy of a_r is present (in this way $\bar{l}_2$ is introduced), and l''_3 passes to $\hat{l}_3$; no other object is changed. In the following step, if $\bar{l}_2$ exists, then it dissolves the membrane with label 2, and introduces l_2; simultaneously, $\hat{l}_3$ is replaced by $\bar{l}_3$. If the dissolution takes place, the object $\bar{l}_3$ arrives in the skin region, where it is replaced by the dummy object d. The instruction from P was correctly simulated for the case when the register r was not zero. If this was not the case, then the rule $l'_1a_r \to \bar{l}_2$ cannot be used, and l'_1 waits in membrane 2 until it is dissolved by the rule $\bar{l}_3 \to l_3\delta$ (in the fourth step). Arrived in the skin region, the object l'_1 is replaced by the dummy object d, hence also the case when the register was empty is correctly simulated.

In both cases, c was released in the skin region, hence the rule $c \to [c]_2$ can be applied again.

We continue in this way, simulating the instructions of M. If the register machine eventually halts (it reaches the label l_h), then also the computation in Π halts and conversely. Consequently, $N(M) = N(\Pi)$. As the system Π is deterministic (for each label $l_1 \in B$ there is exactly one instruction $l_1 : (\dots)$ in P), the proof is complete. □

It is worth noting that the P system from the previous proof has always only two membranes (each membrane creation is followed by a membrane dissolution) – hence it is of no interest to consider different clocks in different levels of its membrane structure (see the beginning of Section 3.5). Moreover, the membrane creation can be avoided, by making use in a larger extent of the power of cooperative rules. However, we will essentially use the idea of this proof in the proof of Theorem 5 below, where the acceleration is obtained by increasing the number of membranes.

Theorem 2. $NCE = DN_aOP_1(sym_0, anti_{2,2})$.

Proof. Consider again a register machine $M = (m, B, l_0, l_h, P)$, a distinct symbol a_i for each register $i = 1, 2, \dots, m$, let U the set of all these symbols, and construct the (recognising) symport/antiportcell-like

P system

$$\begin{aligned}\Pi = \; & (O, [\,]_1, l_0, O, R_1),\\ O = \; & U \cup \{l, l', l'', \bar{l}, \hat{l} \mid l \in B\},\\ R_1 = \; & \{(l_1, \text{out}; l_2 a_r, in) \mid \text{for } l_1 : (\mathtt{ADD}(r), l_2) \in P\}\\ & \cup\{(l_1, out; l'_1 l''_1, in),\\ & (l'_1 a_r, out; \bar{l}_2, in),\\ & (l''_1, out; \hat{l}_1, in),\\ & (\bar{l}_2 \hat{l}_1, out; l_2, in),\\ & (l'_1 \hat{l}_1, out; l_3, in) \mid \text{for } l_1 : (\mathtt{SUB}(r), l_2, l_3) \in P\}.\end{aligned}$$

After introducing n copies of the object a_1 in the unique membrane, the system starts simulating the computation of the register machine starting with the number n in the first register.

The ADD instructions correspond directly to antiport rules from R_1, while the simulation of an instruction $l_1 : (\mathtt{SUB}(r), l_2, l_3)$ is simulated as follows. The available object l_1 is sent out, in exchange with l'_1, l''_1. The first object can leave the system together with a copy of a_r (if such a copy exists, a case when the use of the rule $(l'_1 a_r, out; \bar{l}_2, in)$ is mandatory), otherwise it waits unchanged. The latter object exits and $\hat{l}_1$ enters the system. In the next step, if $\bar{l}_2$ is present (hence a unit was subtracted from register r), then $\bar{l}_2$ exits together with $\hat{l}_1$, introducing l_2 (and completing the simulation); if $\bar{l}_2$ is not present, then l'_1 is present, and it leaves the system in the next step together with $\hat{l}_1$, while l_3 is brought in instead. This completes the simulation of the case when register r was empty. □

The previous result directly implies the equality $NCE = DN_aOtP_1(sym_0, anti_{2,2})$. However, in the case of tissue-like systems, the complexity of the antiport rules can be decreased: only rules of the types $(i, a/b, j), (i, a/bc, j)$ are used, where a, b, c are single objects (but we will also use symport rules).

Theorem 3. $NCE = DN_aOtP_3(sym_2, anti_{1,2})$.

Proof. For a register machine $M = (m, B, l_0, l_h, P)$ and an alphabet $U = \{a_i \mid 1 \leq i \leq m\}$, we construct the tissue-like P system Π whose components are pictorially given in Fig. 4; with the experience of the previous proofs and the information from this figure, the reader can easily see which are the elements of Π. We only examine here in some detail the work of the system when simulating a subtraction instruction of M.

Take an instruction $l_1 : (\mathtt{SUB}(r), l_2, l_3)$. Both in the case when a copy of a_r is present in cell 1 and when the register r is empty, the simulation is completed in five

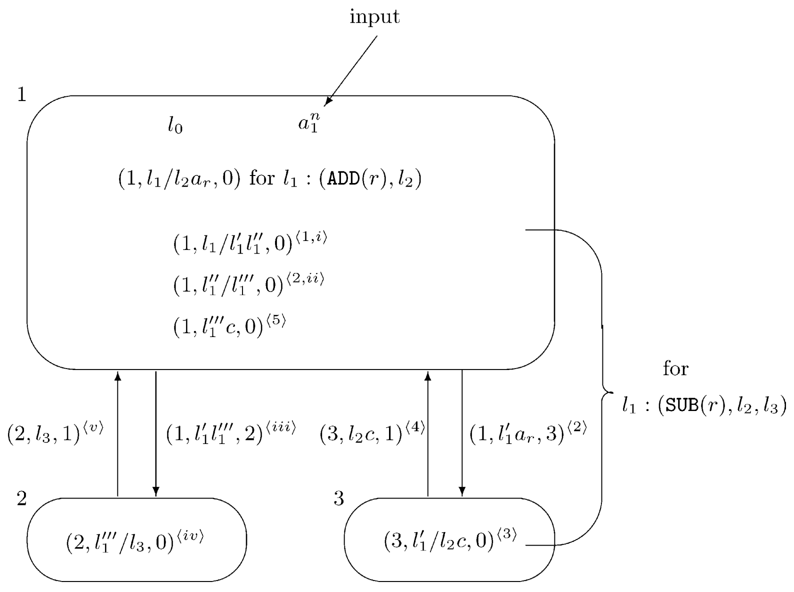
input
1
l_0
a_1^n
$(1, l_1/l_2a_r, 0)$ for $l_1 : (\mathtt{ADD}(r), l_2)$
$(1, l_1/l'_1l''_1, 0)^{\langle 1,i\rangle}$
$(1, l''_1/l'''_1, 0)^{\langle 2,ii\rangle}$
$(1, l'''_1c, 0)^{\langle 5\rangle}$
for
$l_1 : (\mathtt{SUB}(r), l_2, l_3)$
$(2, l_3, 1)^{\langle v\rangle}$
$(1, l'_1l'''_1, 2)^{\langle iii\rangle}$
$(3, l_2c, 1)^{\langle 4\rangle}$
$(1, l'_1a_r, 3)^{\langle 2\rangle}$
2
$(2, l'''_1/l_3, 0)^{\langle iv\rangle}$
3
$(3, l'_1/l_2c, 0)^{\langle 3\rangle}$

Fig. 4. The system from the proof of Theorem 3.

Table 1
Simulating a SUB-instruction in the system from the proof of Theorem 3

Step	Register r non-empty	Register r empty
1	$(1, l_1/l'_1l''_1, 0)$	$(1, l_1/l'_1l''_1, 0)$
2	$(1, l'_1a_r, 3), (1, l''_1/l'''_1, 0)$	$(1, l''_1/l'''_1, 0)$
3	$(3, l'_1/l_2c, 0)$	$(1, l'_1l'''_1, 2)$
4	$(3, l_2c, 1)$	$(2, l'''_1/l_3, 0)$
5	$(1, l'''_1c, 0)$	$(2, l_3, 1)$

steps. The rules used in these steps are marked in Fig. 4 with with Arabic and Roman numbers, respectively, placed in superscripts of the form $\langle \dots \rangle$.

Table 1 indicates the rules used in each step for the two cases – register r non-empty or empty.

It should be noted that if register r is empty, then l'_1 waits in cell 1 until step (iii), when it goes together with l'''_1 to cell 2, where l_3 is brought from the environment, and then sent to cell 1. If the register is not empty, then a copy of a_r is moved in cell 3, together with l'_1; in this way, l'''_1 has to wait until step 5 and it will be moved out of the system with the help of c. In turn, c was brought from cell 3 together with l_2, after having l'_1 here. The reader can complete the missing details of the proof that Π stops if and only if M stops when analysing n (no rule is provided for processing the object l_h in Π). □

Let us note that in the previous proof the universality is obtained by using three cells, working deterministically, with both symport and antiport rules, and communicating (among them and with the environment) in a two-way manner. We have mentioned all these details in order to stress the differences between this P system and a neural-like P system, as defined in Section 3.3, where the communication between "neurons" is one-way.

However, the communication between any two cells from the previous construction is performed by means of symport rules, and, as one can observe from Fig. 4, in each step one communicates in only one direction. In general (see Theorem 6.2.1 from Păun (2002)), the two-way communication realised by symport rules can be easily replaced with one-way communication, providing some intermediate "buffer-cell", as suggested in Fig. 5. A small problem appears in our case, because we also use antiport rules, for communicating with the environment: the communication through buffers takes two steps, hence we have to resynchronise the work of the system. This can be done by replacing each rule of the form $(i, x/y, 0)$ with two rules, $(i, x/\langle y\rangle, 0), (i, \langle y\rangle/y, 0)$, where $\langle y\rangle$ is a new symbol, associated with y. Therefore, universality is obtained also for deterministic neural-like P systems (with seven neurons).

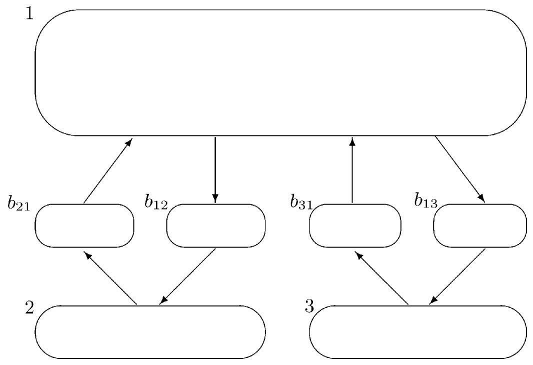

Fig. 5. Making the communication one-way.

Corollary 4. $NCE = DN_aOnP_7(sym_2, anti_{1,2})$.

An interesting question in this area is related to the size of the symport/antiport rules. Rather unexpectedly, it was recently proven (see Bernardini and Gheorghe (2003)) that cell-like P systems with minimal symport and antiport rules (always only one object being moved in any direction) are already universal. The current result in this respect says that five membranes suffice (see Bernardini and Păun (2004)), but the proof is based on a non-deterministic construction. It would be nice to have a similar result also for the deterministic case – for cell-like or tissue-like P systems.

3.5. Accelerated P systems

In general, it is of no interest (from the computational capacity point of view) to consider different time units for different membranes in a cell-like system with a bounded number of membranes (provided that the time units of regions are expressed as rational divisions of a unique, external time unit). Indeed, consider the smallest common multiple of all denominators of fractions expressing the local time units, denote it by r, and take as time unit for all membranes $\tau = 1/r$; for all membranes whose former time units were multiples of τ, introduce synchronising rules of the form $a \to a_1, a_1 \to a_2, \ldots, a_{k-2} \to a_{k-1}$, followed by rules $a_{k-1} \to v$ corresponding to initial rules $a \to v$. In this way, an equivalent system is obtained, with a global clock as in standard P systems.

This argument does not work for systems which can get configurations with arbitrarily many membranes (with arbitrarily many local clocks). Actually, the existence of arbitrarily many local clocks, with the time units decreasing "sufficiently fast" when entering deeper in the membrane structure, is sufficient to reach a computing power beyond Turing computability. (Remember that our starting hypothesis was that "smaller compartments, faster reactions", hence the compartmentalization is crucial. Although this bare assumption is biologically supported/inspired, we infer from it consequences of a mathematical idealized nature which we do not claim to be confirmed by the cell biology: the acceleration can be arbitrary, for instance, as a consequence of having compartments which are arbitrarily small.)

Specifically, in what follows we work with (f, T)-accelerated systems, where we assume that the time unit is strictly smaller from a level of the membrane structure to the next level below, namely, the relation between the time unit t on any level i to the time unit $f(t)$ on level $i+1$ (hence to $f^2(t)$ on level $i+2$, etc.) is given by a Turing computable function f such that $\sum_{i=1}^{\infty} f^i(t) \le T$, for a given constant T.

Theorem 5. *Given any register machine M, there is an (f, T)-accelerated recognising cell-like P system Π, with cooperating rules, using membrane creation and membrane dissolution features, which stops in time at most $9T + 3$ for any input; Π with input n sends to the environment a special object yes if M halts on n, and sends nothing outside ifM does not stop on n.*

Proof. The idea of the proof is to take the system constructed in the proof of TheoremrefT1 and to grow its membrane structure in such a way that the simulation of each further instruction of the starting register machine is done deeper and deeper – hence faster and faster – in the membrane structure. If the computation eventually halts, then the special object yes is sent out, if not, the system will work forever, but this is not relevant outside it, where no more than $9T + 3$ external time units can pass until knowing that the object yes has been produced or not.

Technically, the construction is the following. Take a function f and a constant T as above, and start again from a (deterministic) register machine $M = (m, B, l_0, l_h, P)$. Consider the alphabet $U = \{a_i | 1 \le i \le m\}$, and construct the (recognising) (f, T)-accelerated P system

$$\Pi = (O, \{1, 2\}, [\,]_1, l_0 c, R_1, R_2),$$

with

$$O = U \cup \{l, l', l'', \bar{l}, \hat{l} | l \in B\} \cup \{c, c', d, \mathtt{yes}\},$$

and the following sets of rules:

$$\begin{aligned}
R_1 = \quad & \{c \to [c']_1, \\
& c' \to [c]_2\} \\
& \cup \{\alpha \to \alpha_{in} | \alpha \in B \cup U\} \\
& \cup \{l' \to d, \bar{l} \to d | l \in B - \{l_h\}\} \\
& \cup \{l_h \to \mathtt{yes}_{out}, \mathtt{yes} \to \mathtt{yes}_{out}\}, \\
R_2 = \quad & \{l_1 \to l_2 a_r \delta | \text{ for } l_1 : (ADD(r), l_2) \in P\} \\
& \cup \{l_1 \to l'_1 l''_1, \\
& l'_1 a_r \to \bar{l}_2, \\
& \bar{l}_2 \to l_2 \delta, \\
& l''_1 \to \hat{l}_3, \\
& \hat{l}_3 \to \bar{l}_3, \\
& \bar{l}_3 \to l_3 \delta | \text{ for } l_1 : (SUB(r), l_2, l_3) \in P\}.
\end{aligned}$$

Like in the proof of Theorem 1, the simulation of the instructions of P is performed in membrane 2, which is created inside membrane 1 by means of the rule $c' \to [c]_2$ from R_1. After simulating any instruction from P, membrane 2 is dissolved, and c is released in the enclosing membrane 1. By means of the rule $c \to [c']_1$, c creates first one further membrane 1 (where all objects come in the next step), and then a new membrane 2 is created (where the next instruction of M is simulated). In this way, the membrane structure grows with one level (one membrane 1) for each instruction simulated.

From the proof of Theorem 1, it is clear that the system Π will introduce the label l_h, when starting from a multiset $l_0 c a_1^n$, if and only if M halts when starting with the number n in the first register.

Now, if M halts, hence the label l_h is introduced in Π – this always happens when dissolving membrane 2, hence in the central membrane 1 – then the special object yes is immediately introduced. This object will travel step by step from the deepest existing membrane 1 until exiting the system.

Therefore, we get the object yes in the environment if and only if the register machine halts (when starting with n in the first register).

What is the maximal number of steps the system can perform at any give level of the membrane structure? It is obvious that the largest number of steps in the same membrane (without dissolving and creating new membranes) is done when simulating SUB-instructions, namely, when the corresponding register is empty. Thus, let us examine such a case. We are in a membrane 1, on level i of the membrane structure, with a multiset l_1cw, where $w \in U^*$. From c, we create a new membrane 1 (level $i+1$). In the next step, we both introduce all objects in this new membrane (hence two steps are performed at level i) and we create a membrane 2 (on level $i+2$). In the next step, all objects are introduced in the central membrane 2 (hence this is the second step at level $i+1$). Now, we perform four steps in membrane 2, ending with its dissolution. In this way, we return to level $i+1$, and we repeat the procedure. This means that we perform two steps here, then two steps in the lower level, four steps one level below ($i+3$) and we return to level $i+2$, where we again perform two steps. In total, eight steps performed at level $i+2$. After that, we never return to level $i+2$ for simulating instructions of M – but we will return here for one step when sending the object yes out (if this is the case). Consequently, the maximal number of steps we can perform at any level of the system is 9, with the observation that at the first level we perform two steps in the beginning and one when sending the object yes out. Because these steps are of length 1 (the skin has the same clock as the environment), this means that the maximal time the system Π works is $9T+3$ external units.

In Fig. 6 we illustrate the above analysis of the number of steps performed at each level of the membrane structure; each "constellation" (with "stars" ●, ○, $*$) corresponds to the simulation of a SUB-instruction, in the worst case, when the respective register is empty. The notation indicates the creation of a new membrane 1 (cr1), of a new membrane 2 (cr2), the movement of objects in the respective membranes (mv1, mv2, respectively), and the simulation of the instruction from P, ended with the dissolution of membrane 2 (δ). The statement of the theorem is completely proved. □

Results as above can also be obtained as consequences of Theorems 2 and 3, and Corollary 4, in a direct way, by assuming that the time needed by the ith transition of a computation is $f(i)$, for a function f such that the system is (f, T)-accelerated; in such a case, we get the answer to the Halting Problem in T units of global time, just following the scenario from Fig. 1. Therefore, we can state:

Theorem 6. *Given any register machine M, there is an (f, T)-accelerated recognising P system Π of any of the types (i) cell-like with symport/antiport rules, (ii) tissue-like, (iii) neural-like which stops in at most T units of global time for any input; Π with input n sends to the environment a special object* yes *if M halts on n, and sends nothing outside if M does not stop on n.*

Accelerated P systems of any of the four types appearing in Theorems 5 and 6 can recognise all Turing computable sets of numbers (just take $f(i) = 1$ for all i, hence the "zero-accelerated accelerated systems") and strictly more. How much can we go beyond Turing barrier?

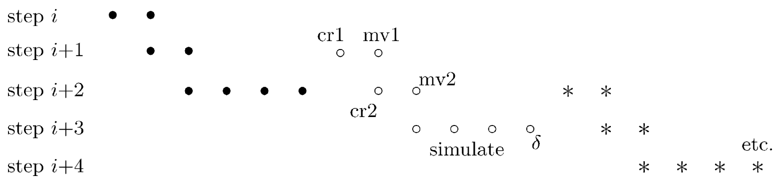

Fig. 6. Counting the steps in Theorem 5.

Theorem 7. *Accelerated P systems of any of the classes mentioned in Theorems 5 and 6 recognise exactly the class* Σ_1.

Proof. One implication follows from the fact that the Halting Problem is Turing complete. For the other implication, if Π is an (f, T)-accelerated P system, then we define the predicate $R_\Pi(n, t)$ to be true if after running Π for time t on n we get `yes`. Clearly, R_Π is Turing computable and n is recognised by Π if and only if there is a time $t \leq T$ such that Π produces `yes`, i.e., $(\exists t)(R_\Pi(n, t))$. □

4. Intelligence, consciousness and Turing non-computability

In this speculative short section we will discuss two possible applications of our results to the understanding of intelligence, in the human brain and nearby Universe.

In classical biophysics, cognitive processes are modelled as neural networks (see Khanna (1990)); this successful approach delivered implementations of learning and memory and fuelled optimism that a sufficient complex artificial neuronal network may, at least theoretically, reproduce the essence of the brain behaviour accounting for intelligence and consciousness. There is ample support (related to semantics, binding, neuronal correlation of consciousness, to name just a few problems) that this line of modelling may ultimately prove insufficient. More importantly, Penrose (Penrose, 1989, 1994) has argued that human understanding (at least of some mathematical facts) must involve a *Turing non-computational element*. The non-computational feature is part of conscious thinking and may be evidenced in the difference and transition between pre-conscious and conscious processing. As a consequence, it was claimed that, by trespassing the Turing barrier, the human mind has a special ability to comprehend unassailable mathematical truths. We agree with this line of arguments.[11]

The view that the ephemeral nature of consciousness evokes a quantum process has evolved into the theory of objective reduction (OR) (see Penrose (1989, 1994), Hameroff and Penrose (1966)) which holds that quantum coherence and wave function self-collapse are essential for consciousness. Are there quantum computations in the brain? Hameroff and Penrose (Hameroff and Penrose, 1966) strongly favour a positive answer; proteins are quantum bits and cytoskeletal microtubules are assemblies of entangled quantum bits proteins.

Other structures within each of the brain's neurons might also participate. Among them, *neural membrane proteins* (see Marshall (1989)). We contend that some of these membranes may, under certain conditions, act in an accelerated way, thus triggering Turing uncomputable computations. Chances that such a spontaneous acceleration occurs may be rather high due to, among other factors, the human brain having about 10^{18} tubulins. The fact that accelerated P systems recognise exactly the class of Σ_1 sets which have a special "finitary" nature (see Calude et al. (2000)) might also reinforce our hypothesis. At this stage we have just advanced this hypothesis; we are planning to devote a detailed analysis of this phenomenon in a different paper.

Finally, is it possible to imagine a plausible/consistent 'biological scenario' to scan the Universe for possible forms of intelligence? It seems that a better candidate to send is a Chaitin Ω number[12], which (see Bennett and Gardner (1979))

> embodies an enormous amount of wisdom in a very small space ... inasmuch as its first few thousands digits, which could be written on a small piece of paper, contain the answers to more mathematical questions than could be written down in the entire universe.

In Chaitin (1975) Chaitin introduced the number Ω representing the probability that a randomly chosen program will halt. In order to make this idea precise, one can modify the definition of the register machine programs in such a way that the programs are *self-delimiting* binary strings.[13]. This allows us to

[11] There are still problems with some arguments involving Gödel Incompleteness Theorem (see also the discussion in Penrose (1990)).

[12] It seems that current projects have used the digits of π.

[13] This can be done, for example as in Chaitin (1987), by introducing input instructions, representing the input positive integer in binary and appending it immediately after the `HALT` instruction. A program not reading the whole data or attempting to read past the last data-bit results in a run-time error, so fails to halt.

choose a random program by flipping an unbiased coin to generate each bit, stopping when we reach a valid (self-delimited) program. Formally, a Chaitin Ω number is defined by

$$\Omega = \sum_{p \text{ halts}} 2^{-|p|}, \tag{1}$$

where $|p|$ is the length in bits of the register machine program p.

The number Ω has some important properties: (a) in contrast with π, which looks (algorithmically) random, but it is not, Ω is random, in particular, Ω is unpredictable and Turing uncomputable, (b) there exists a Turing machine approximating it, (c) knowing the first n bits of the binary (infinite) expansion of Ω allows us to solve the Halting Problem for all register machine programs of no more than n bits in length.[14] For large n, this includes a huge amount of information, in particular, answers to most important mathematical problems, solved (e.g., Fermat Last Theorem) or unsolved (Riemann Hypothesis[15]). For more details see Calude (2002), Calude et al. (2000). Consequently, it is obvious that Ω includes vastly more information than π, or for this matter, any Turing computable number.

Next we will show a method of enumerating the bits of Ω based on a Σ_1 predicate. To this aim we construct, following Ord and Kieu (2003), a program $Q(k, N)$ that takes two positive integers, k and N, as input and halts if and only if $N \cdot 2^{-k} < \Omega$. To write this program we construct first the successive approximations of Ω,

$$\Omega_i = \sum_{p \text{ halts in less than } i \text{ steps}_{|p| \leq i}} 2^{-p}, \tag{2}$$

and check at each stage whether $N \cdot 2^{-k} < \Omega_i$, halting if this is true and continuing otherwise. Since $\{\Omega_i\}$ approaches Ω from below, if $N \cdot 2^{-k} < \Omega$, then there is some i for which $N \cdot 2^{-k} < \Omega_i$, and Q halts as required. On the other hand, if $N \cdot 2^{-k} > \Omega$, then there is no such i and Q will not halt. Hence, Q is in Σ_1 because Ω_i is Turing computable:

$$Q(k, N) = 1 \text{ if and only if } (\exists i)(N \cdot 2^{-k} < \Omega_i). \tag{3}$$

Finally, to determine the first k bits of Ω, we determine the greatest value of N for which $N \cdot 2^{-k} < \Omega$. For this N we have $N \cdot 2^{-k} < \Omega < (N+1) \cdot 2^{-k}$ and thus the binary representation of N expressed as a k digit binary number (with leading zeros if necessary) gives exactly the first k digits of the binary expansion of Ω. So, the whole process reduces to the computation of a Σ_1 set and a classical Turing computation, a task within the power of an accelerated P system (see Theorems 5 and 7).

The main problem is to transmit the bits of Ω. As pointed in Chown (2003), physics is not particularly useful as the intensity of emitted radiation falls with an inverse-square law (the intensity of radiation emitted by a relativistically receding source falls as the fourth power of the Lorentz factor), so the signal is fading extremely fast. We suggest the following 'bio-scenario' for sending bits of Ω: a neuron-like type of accelerated P system, self-replicating, which at each unit of (local) time t computes and emits (as the result of an accelerated computation) the first t digits of Ω and produces an exact replica of itself (which continues the computation in an accelerated mode at time $t + 1$). In this way the space is invaded with more and more self-replicating bio-systems each of which emits some bits of Ω, the sign of our 'intelligence'. Why do we need acceleration? Simply because with a Turing computation the above scenario will produce only finitely different many copies of the system (see Calude (2002)): at some moment the process will stop producing new bits. An accelerated computation will guarantee, by Theorem 5, that the process produces more and more powerful copies, each being capable of emitting more and more digits of Ω. Again, we hope to discuss the above facts in a more technical way in a forthcoming paper.

5. Final remarks

In this paper we have proposed, for the first time in the literature on molecular computing, bio-computing models theoretically capable of trespassing the Turing barrier. Specifically, we have constructed four models expressed in the language of membrane comput-

[14] Information is maximally compressed in Ω: with only n bits one can code the halting/non-halting status of 2^n programs.

[15] A crude estimation shows that to solve the Riemann Hypothesis, one of the Millennium 1 million dollars problem, is equivalent to decide whether a register machine program of less than 10,000 bits in length does or does not halt.

ing, a recent and vivid branch of natural computing, cell-like P systems with multiset rewriting-like rules, cell-like P systems with symport/antiport, tissue-like P systems, and neural-like P systems. Each model exploits some type of “time acceleration” inspired from biology and thus can solve the Halting Problem in a bounded, known (external) time.

The mathematical results on which these acceleration results are based are interesting in themselves for the membrane computing area. They suggest several further research topics. Can these results be improved in the number of membranes, the size of rules, the number of ingredients used? Can we find “deterministic variants” for other universality results from membrane computing? Is there any case when non-determinism is necessary (are deterministic systems of a given type strictly less powerful than the non-restricted systems)? The study of measures of non-determinism is also important. All these problems are both mathematically and biologically appealing. Other problems interest the main goal of this paper. One is to find other ways to increase the power of a P system (not necessarily Turing universal) so that Turing non-computable functions can be computed. A suggestion can come from the so-called “computing by carving” (Păun, 1999): removing arbitrarily many numbers/strings from a computable set/language amounts at passing to the complement of that set/language, an operation known not to preserve Turing enumerability. Has this procedure any biological interpretation/analogy?

More technically, what about accelerating the rules of a system individually, not through its transitions: the first time when used, any rule needs one time unit for being applied, but in the next step it is applied in a shorter time, and so on, that is, each rule needs a time which depends on its “experience”. This leads to an intricate de-synchronisation (slow rules can keep busy objects for time intervals during which faster rules are applied more times – and gets still faster in this way).

Finally, we briefly explored some implications of our results in the quest of understanding intelligence, in the human brain, and our Universe. These ideas, particularly those related to the relevance of neuron-like P systems to the behaviour of the brain, need further investigations.

Uncited references

Ardelean (2002), Frisco and Hoogeboom (2004), Ito (2001), Madhu and Krithivasan (2001), Salomaa (1973), Chaitin (2001).

Acknowledgements

We are grateful to G. Chaitin, M. Chown, and G. Rozenberg for encouragement and critique. Thanks are also due to three anonymous referees, for many remarks which helped to improve the form of presentation. The first author acknowledges partial support from AGAUR 2002PIV.

References

Adamyan, V.A., Calude, C.S., Pavlov, B.S., 2004. Transcending the limits of Turing computability. In: Hida, T. (Ed.), Proceedings of Winter School, Meijo University, Japan, World Scientific, Singapore, to appear.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P., 2002. Molecular Biology of the Cell, fourth ed. Garland Science, New York.

Ardelean, I.I., 2002. The relevance of cell membrane for P systems. General aspects. Fundam. Informat. 49 (1–3), 35–43.

Bennett, C.H., Gardner, M., 1979. The random number omega bids fair to hold the mysteries of the universe. Sci. Am. 241, 20–34.

Bernardini, F., Gheorghe, M., 2003. On the power of minimal symport/antiport. In: Pre-proceedings of Workshop on Membrane Computing, WMC2003, Tarragona, GRLMC Report 28/03. Rovira i Virgili Univ., Tarragona 72–83.

Bernardini, F., Păun, A., Universality of minimal symport/antiport: five membranes suffice, in Martín-Vide et al. (2003), pp. 43–54.

Blake, R.M., 1926. The paradoxes of temporal process. J. Philos. 23, 645–654.

Calude, C.S., 2002. Information and Randomness: An Algorithmic Perspective, second ed., Revised and Extended. Springer-Verlag, Berlin.

Calude, C.S., Casti, J.L., 1998. Parallel thinking. Nature, 392 (9), 549-551.

Calude, C.S., Dinneen, M.J., Svozil, K., 2000. Reflections on quantum computing. Complexity 6 (1), 35–37.

Calude, C.S., Jürgensen, H., Legg, S., 2000. Solving finitely refutable mathematical problems. In: Calude, C.S., Păun Gh. (Eds.), Finite Versus Infinite. Contributions to an Eternal Dilemma. Springer-Verlag, London, pp. 39–52.

Calude, C.S., Păun, Gh., 2001. Computing with Cells and Atoms. Taylor and Francis Publishers, London.

Calude, C.S., Pavlov, B., 2002. Coins, quantum measurements, and Turing's barrier. Quantum Inform. Process. 1 (1–2), 107–127.

Casti, J.L., 1997. Computing the uncomputable. The New Scientist 154 (17), 34.

Casti, J.L., 2003. The One True Platonic Haven, Joseph Henry Press, New York.

Chaitin, G.J., 1975. A theory of program size formally identical to information theory. J. Assoc. Comput. Mach. 22, 329–340.

Chaitin, G.J., 1987. Algorithmic Information Theory, Cambridge University Press, Cambridge, 1987 (third printing 1990).

Chaitin, G.J., 2001. Exploring Randomness. Springer-Verlag, London.

Chown, M., 2003. Personal communication to C. Calude, 18 November.

Chown, M., 2002. Smash and grab (Feature Story), New Scientist, 6 April, 24–28.

Copeland, B.J., 1998. Even Turing machines can compute uncomputable functions. In: Calude, C.S., Casti, J., Dinneen, M.J. (Eds.). Unconventional Models of Computation. Springer-Verlag, Singapore, pp. 150–164.

Copeland, B.J. (Ed.). 2002. Minds Machines, 12, 4 (2002), and 13, 1 (2003).

Copeland, B.J., 2002. Hypercomputation. Minds Machines 12 (4), 461–502.

Davis, M., 2004. The myth of hypercomputation, in Teuscher (2003), pp. 195–211.

Delahaye, J.-P., 2003. La barrière de Turing. Pour la Science 312, 90–95.

Etesi, G., Németi, I., 2002. Non-Turing computations via Malament-Hogarth space-times. Int. J. Theor. Phys. 41, 341–370.

Frisco, P., Hoogeboom, H.J., 2004. Simulating counter automata with P systems with symport/antiport, in Păun et al. (2002), pp. 288-301.

Hameroff, S., Penrose, R., 1966. Consciousness events as orchestrated space-time selections. J. Conscious. Stud. 2, 36–53.

Hoffmeyer, J., 1998. Surfaces inside surfaces. On the origins of agency and life. Cybern. Hum. Knowing 5 (1), 33–42.

http://www.psystems.disco.unimib.it.

http://www.hypercomputation.net/bib.

Ibarra, O.H., 2004. The number of membranes matters, in Martín-Vide et al. (2003), pp. 217–230.

Ibarra, O.H., in press. On the computational complexity of membrane computing systems, Theor. Comput. Sci., in press.

Ito, M., Martín-Vide, C., Păun, Gh., 2001. A characterization of Parikh Sets of ET0L languages in terms of P systems. In: Ito, M., Păun, Gh., Yu, S. (Eds.). Words, Semigroups, and Transducers, World Scientific, Singapore, pp. 239–254.

Kauffman, S., 1993. The Origin of Order. Oxford University Press, Oxford.

Khanna, T., 1990. Foundations of Neural Networks, Addison-Wesley, New York.

Kieu, T.D., 2001. Quantum algorithm for the Hilbert's tenth problem, Los Alamos preprint archive http://arXiv: quant-ph/0110136,v2, 9 November 2001.

Kieu, T.D., 2002. Quantum hypercomputation. Minds Machines 12 (4), 541–561.

Loewenstein, W.R., The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life. Oxford University Press, New York, Oxford.

Madhu, M., Krithivasan, K., 2001. P systems with membrane creation: Universality and efficiency. In: Margenstern, M., Rogozhin, Y. (Eds.), Proceedings of the Third International Conference on Universal Machines and Computations, Chişinau, Moldova, 2001, Lecture Notes in Computer Science, 2055. Springer-Verlag, Berlin, pp. 276–287.

Marcus, S., Bridging P systems and genomics: a preliminary approach, in Păun et al. (2002), pp. 371–376.

Marshall, I.N., 1989. Consciousness and Bose-Einstein condensates. New Ideas Psychol. 7, 73–83.

Martín-Vide, C., Păun, Gh., Rozenberg, G., Salomaa, A. (Eds.). 2003. Membrane Computing. International Workshop, WMC 2003, Tarragona, Spain, Revised Papers, Lecture Notes in Computer Science, 2933. Springer-Verlag, Berlin.

Minsky, M., 1967. Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ.

Ord, T., 2002. Hypercomputation: Computing more than the Turing Machine. Honours Thesis, Computer Science Department, University of Melbourne, Australia, 2002; arxiv.org/ftp/math/papers/0209/0209332.pdf.

Ord, T., Kieu, T.D., 2003. On the existence of a new family of Diophantine equations for Ω. Fundam. Informat. 56 (3), 273–284.

Păun, Gh., 1999. Computing by carving. Soft Comput. 3 (1), 30–36.

Păun, Gh., 2000. Computing with membranes. J. Comput. Syst. Sci. 61 (1), 108–143.

Păun, Gh., 2002. Computing with Membranes: An Introduction. Springer-Verlag, Berlin.

Păun, Gh., Rozenberg, G., 2002. A guide to membrane computing. Theor. Comput. Sci. 287 (1), 73–100.

Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (Eds.). 2002. Membrane Computing. International Workshop, WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, 2597. Springer-Verlag, Berlin.

Penrose, R., 1989. The Emperor's New Mind. Concerning Computers, Minds, and the Laws of Physics. Oxford University Press, Oxford.

Penrose, R., 1990. Précis of The Emperor's New Mind. Concerning Computers, Minds, and the Laws of Physics (together with responses by critics and a reply by the author). Behav. Brain Sci. 13, 643–705.

Penrose, R., 1994. Shadows of the Mind. Oxford University Press, Oxford.

Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F., 2002. Teoría de la Complejidad en Modelos de Computatión Celular con Membranas, Editorial Kronos, Sevilla.

Russell, B.A.W., 1936. The limits of empiricism. Proc. Aristotelian Soc. 36, 131–150.

Salomaa, A., 1973. Formal Languages. Academic Press, New York.

Siegelmann, H.T., 1995. Computation beyond the Turing limit. Science 268, 545–548.

Siegelmann, H.T., 2003. Neural and super-Turing computing. Minds Machines 13 (1), 103–114.

Stewart, I., 1991. Deciding the undecidable. Nature 352, 664–665.

Svozil, K., 1998. The Church-Turing Thesis as a guiding principle for physics. In: Calude, C.S., Casti, J., Dinneen. M.J. (Eds.). Unconventional Models of Computation. Springer-Verlag, Singapore, pp. 371–385.

Teuscher, C., Sipper, M., 2002. Hypercomputation: Hyper or computation? Communications ACM 45 (8), 23–24.

Teuscher, C. (Ed.). 2003. Alan Turing: Life and Legacy of a Great Thinker. Springer-Verlag, Heidelberg.

Weyl, H., 1927. Philosophie der Mathematik und Natureissenschaft, R. Oldenburg, Munich.

Wiedermann, J., van Leeuwen, J., Relativistic computers and non-uniform complexity theory. In: Calude, C.S., Dinneen, M.J., Peper, F. (Eds.). Unconventional Models of Computation (UMC'02). Lecture Notes Comput. Sci. 2509. Springer-Verlag, Berlin, pp. 287–299.