Twenty Six Research Topics About Spiking Neural P Systems

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
and
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

1 Foreword

To continue the tradition of the previous brainstorming weeks on membrane computing, I am collecting here a series of open problems and research topics, not about membrane computing in general, but about one of the directions of research which were pretty much investigated in the last year: spiking neural P systems. In general, one mentions issues which look of a broader nature, but also some precise problems are formulated. As usual with such lists of problems, the selection is subjective, by no means exhaustive.

Of course, choosing only problems related to spiking neural P systems does not mean that there are no longer enough problems waiting to be solved in the general framework of membrane computing – on contrarily (e.g., separate lists can refer to computational complexity issues, to dynamical systems approaches, etc.), but such problems tend to become rather specialized and technical at the present stage of the development of membrane computing. Instead, the membrane computing models with a neural inspiration are at the beginning of a systematic exploration, and, as claimed below, this area of research looks very promising.

2 Forecast

It is obvious that the (human) brain structure and functioning, from neurons, astrocytes, and other components to complex networks and complex (chemical, electrical, informational) processes taking place in it, should be – and only partially is – a major source of inspiration for informatics (I choose this more general term rather that the restrictive, but usual, "computer science", in order to stress that I have in mind both mathematics *per se* and practice, both the theory of computability and the use of computing machineries). If biology is such a rich source of inspiration for informatics as natural computing proves, then the brain should be the "golden mine" of this intellectual enterprise. Risking a forecast, I believe that *if something really great is to appear in informatics in the near future, then this "something" will be suggested by the brain (and this will probably be placed at the level of "strategies" of computing, not at the "tactic" level* – just in balance with the two computing devices already learned from the brain activity and which can be considered the most central notions in informatics, the Turing machine and the finite automaton).

The previous statements do not intend to suggest that spiking neural P systems are the answer to this learning-from-brain challenge, but only to call (once again) the attention to this challenge. Becoming familiar with brain functioning, in whatever reductionistic framework (as spiking neural P systems investigation is), can however be useful. After all, "the road of one thousand miles starts with the first step", Lao Tze said... Let us make from spiking neural P systems "the first step".

3 Some (Neural) Generalities

The neuron is a highly specialized cell, at the same time intricate and simple, robust and fragile, like any other cell, but having the particularity of being involved (in general) in huge networks by means of the synapses established with partner neurons. It is not at all the intention of these lines to give any biological information from this area, but only to point out some of the peculiarities related to neurons and the brain: the functioning of each neuron assumes chemical, electrical, and informational processing at the same time; the axon is not a simple transmitter of impulses, but an information processor; in the communication between neurons the spiking activity plays a central role (which means that the distance in time between consecutive spikes is used to carry information, that is, time is a support of information); the neurons are not cooperating only through synapses, but their relationships are also regulated through the calcium waves controlled by the astrocytes, "eavesdroppers" of axons playing an important role in the neural communication; the brain displays a general emergent behavior which, to my knowledge and to my understanding, cannot be explained only in terms of neuron interrelationships (something is still missing in this picture, maybe of a quantum nature – as Penrose suggests, maybe related to the organization of parts, maybe of a still subtler or even unknown nature). Some of these ideas (especially spiking) are supposed to lead to "neural computing of the third generation", which suggests that already computer scientists are aware of the possibility of major progresses to be made (soon) on the basis of progresses in neuro-biology.

The bibliography of this note contains several titles, both from the general biology of the cell [1], general neurology [41], and from neural computing based on spiking [3], [29], [16], [26], [27], [28], about the axon as an information processor [39], astrocytes and their role in the brain functioning [37], [40]. Of course, these titles are only meant to be initial "dendrites" to the huge bibliography related to (computer science approaches to) brain functioning.

4 Spiking Neural P Systems – Informal Presentation

Spiking neural P systems (SN P systems, for short) were introduced in [23] in the precise (and modest: trying to learn a new "mathematical game" from neurology, not to provide models to it) aim of incorporating in membrane computing ideas specific to spiking neurons; the intuitive goal was to have (1) a tissue-like P system with (2) only one (type of) object(s) in the cells – the *spike*, with (3) specific rules for evolving populations of spikes, and (4) making use of the time as a support of information.

In what follows, I briefly describe several classes of SN P systems investigated so far, as well as some of the main types of results obtained in this area.

In short, an SN P system (of the basic form – later called a *standard* SN P system) consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along the arcs of the graph (they are called *synapses*). The objects evolve by means of *spiking rules*, which are of the form $E/a^c \rightarrow a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \rightarrow \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes.

An extension of theses type of rules was considered (with a mathematical motivation) in [30], [14]: rules of the form $E/a^c \rightarrow a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced (one assumes that $c \geq p$, not to produce more than consuming). Because p can be 0 or greater than 0, we obtain a generalization of both spiking and forgetting rules, while forgetting rules also have a regular expression associated with them.

An SN P system (with standard as well with extended rules) works in the following way. A global clock is assumed and in each time unit each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential locally: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop.

With a spike train we can associate various numbers, which can be considered as *computed* (we also say *generated*) by an SN P system. For instance, in [23] only the distance between the first two spikes of a spike train was considered, then in [33] several extensions were examined: the distance between the first k spikes of a spike train, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, all computations or only halting computations, etc.

An SN P system can also work in the *accepting* mode: a neuron is designated as the *input neuron* and two spikes are introduced in it, at an interval of n steps; the number n is accepted if the computation halts.

Two main types of results were obtained: computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semilinear sets of numbers in the case when a bound was imposed.

Another attractive possibility is to consider the spike trains themselves as the result of a computation, and then we obtain a device generating a (binary) language. We can also consider input neurons and then an SN P system can work as a transducer. Such possibilities were investigated in [34]. Languages – even on arbitrary (i.e., not only binary) alphabets – can be obtained also in other ways: following the path of a designated spike across neurons, as proposed in [12], or using rules of the extended form mentioned above. Specifically, with a step when the system sends out i spikes, we associate a symbol b_i, and thus we get a language over an alphabet with as many symbols as the number of spikes simultaneously produced. This case was investigated in [14], where representations or characterizations of various families of languages were obtained. (An essential difference was found between the case when zero spikes sent out is interpreted as a symbol b_0 and the case when this is interpreted as inserting λ, the empty string, in the result.)

Other extensions were proposed in [21] and [20], where several output neurons were considered, thus producing vectors of numbers, not only numbers. A detailed typology of systems (and of sets of vectors generated) is investigated in the two papers mentioned above, with classes of vectors found in between the semilinear and the recursively enumerable ones.

The proofs of all computational completeness results known up to now in this area are based on simulating register machines. Starting the proofs from small universal register machines, as those produced in [25], one can find small universal SN P systems (working in the generating mode, as sketched above, or in the computing mode, i.e., having both an input and an output neuron and producing a number related to the input number). This idea was explored in [30] and the results are as follows: there are universal computing SN P systems with 84 neurons using standard rules and with only 49 neurons using extended rules. In the generative case, the best results are 79 and 50 neurons, respectively.

In the initial definition of SN P systems several ingredients are used (delay, forgetting rules), some of them of a general form (unrestricted synapse graph, unrestricted regular expressions). As shown in [19], several normal forms can be found, in the sense that some ingredients can be removed or simplified without losing the computational completeness. For instance, the forgetting rules or the delay can be avoided, and the outdegree of the synapse graph can be bounded by 2, while the regular expressions from firing rules can be of very restricted forms. The dual problem, of the indegree bounding, was solved (affirmatively) in [35].

Besides using the rules of a neuron in the sequential mode introduced above, it is possible to also use the rules in a parallel way. A possibility was considered in [24]: when a rule is enabled, it is used as many times as possible, thus exhausting the spikes it can consume in that neuron. As proved in [24], SN P systems with the exhaustive use of rules are again universal, both in the accepting and the generative cases.

In the proof of these results the synchronization plays a crucial role, but both from a mathematical point of view and from a neuro-biological point of view it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory: even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used, the neuron may remain still, maybe receiving spikes from the neighboring neurons; if the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused; if the new spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). This way of using the rules applies also to the output neuron, hence now the distance in time between the spikes sent out by the system is no longer relevant. That is why, for non-synchronized SN P systems we take as a result of a computation the total number of spikes sent out; this, in turn, makes necessary considering only halting computations (the computations never halting are ignored, they provide no output). Non-synchronized SN P systems were introduced and investigated in [7], where it is proved that SN P systems with extended rules are still equivalent with Turing machines (as generators of sets of natural numbers).

5 Some (More) Formal Definitions

To make clearer some of the subsequent formulations, I recall here the definition of central classes of SN P systems, but more details should be found in the papers mentioned in the bibliography. No general notions or notations from language or automata theory, computability, complexity, computer science in general, or membrane computing, are recalled.

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);

2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   b) R_i is a finite set of *rules* of the following general form:

$$E/a^c \to a^p; d,$$

   where E is a regular expression with a the only symbol used, $c \geq 1$, and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$, too.
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron.*

A rule $E/a^c \to a^p; d$ with $p \geq 1$ is called a *firing* (we also say *spiking*) *rule*; a rule $E/a^c \to a^p; d$ with $p = d = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule.* If $L(E) = \{a^c\}$, then the rules are written in the simplified form $a^c \to a^p; d$ and $a^c \to \lambda$. A system having only rules of the forms $E/a^c \to a; d$ and $a^c \to \lambda$ is said to be *restricted* (we also use to say that such a system is a *standard* one).

The rules are applied as follows: if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a^p; d \in R_i$ (with $p \geq 1$) is enabled and it can be applied; applying it means that c spikes are consumed, only $k - c$ remain in the neuron, the neuron is fired, and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately, if $d = 1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In step $t + d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t+d+1$). The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one, hence with $p = 0$, then no spike is emitted (and the neuron cannot be closed, because also $d = 0$).

In the synchronized mode, considered up to now in all SN P systems investigations except [7], a global clock is assumed, marking the time for all neurons, and in each time unit, in each neuron which can use a rule, a rule must be used. Because two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and then one of them is chosen non-deterministically. Note that the neurons work in parallel (synchronously), but each neuron processes sequentially its spikes, using only one rule in each time unit.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron. During the computation, a configuration is described by both the number of spikes present in each neuron and by the state of the neuron, more precisely, by the number of steps to count down until it becomes open (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m\rangle$ is the configuration where neuron $\sigma_i, i = 1, 2, \ldots, m$ contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0\rangle$ (see an example in Figure 2).

Using the rules as suggested above, we can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation, halting or not, we associate a *spike train*, a sequence of digits 0 and 1, with 1 appearing in positions which indicate the steps when the output neuron sends spikes out of the system (we also say that the system itself spikes at that time). With any spike train we can associate various numbers, which are considered as computed (generated) by the system; in the spirit of spiking neural computing, the distance between certain spikes are usually taken as the result of a computation (e.g., the distance between the first two spikes). Because of the non-determinism in using the rules, a given system computes in this way a set of numbers. An SN P system can be also used in the accepting mode: a number n is introduced in the system in the form of the distance between two spikes entering a specified neuron, and this number is accepted if the computation eventually halts.

We denote by $N_{gen}(\Pi)$ the set of numbers generated (in the synchronized way) by a system Π in the form of the number of steps elapsed between the first two spikes of a spike train. Then, by $Spik_2SP_m(rule_k, cons_p, forg_q, del_d)$ we denote the family of such sets of numbers generated by systems with at most m neurons, each of them containing at most k rules, all of them of the standard form, and each rule consuming at most p spikes, forgetting at most q spikes, and having the delay at most d. When using extended SN P systems, we use $Spik_2EP_m(rule_k, cons_p, prod_q, del_d)$ to denote the family of sets $N_{gen}(\Pi)$ generated by systems with at most m neurons, each of them containing at most k rules (of the extended form), each spiking rule consuming at most p spikes, producing at most q spikes, and having the delay at most d. When any of the parameters m, k, p, q, d is not bounded, it is replaced by $*$. When using the rules in the exhausting or the non-synchronized mode, we write $N^{ex}_{gen}(\Pi), N^{nsyn}_{gen}(\Pi)$, respectively, and the superscripts ex and $nsyn$ are also added to $Spik$ in the families notation.

The notations should be changed when dealing with other sets of numbers than the distance between the first two spikes, with accepting systems, when generating or accepting languages, but I do not enter here into details. Instead, I close this section by introducing two important tools in presenting SN P systems, namely, the graphical representation and the transition diagram.

Figures 1, 2 are recalled from [9]. The graphical representation of an SN P system is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses; an arrow also exits from the output neuron, pointing to the environment; in each neuron we specify the rules and the spikes present in the initial configuration.

Figure 1 represents the initial configuration of a system Π. We have three neurons, labeled with 1, 2, 3, with neuron σ_3 being the output one. Each neuron contains two rules, with neurons σ_1 and σ_2 having the same rules (firing rules which can be chosen in a non-deterministic way, the difference between them being in the delay from firing to spiking), and neuron σ_3 having one firing and one forgetting

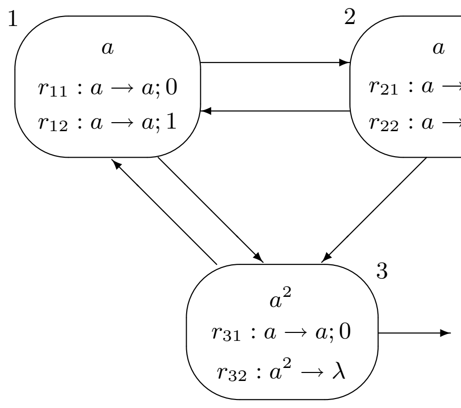

Fig. 1. The initial configuration of the SN P system Π

rule. In the figure, the rules are labeled, and these labels are useful below, in relation with Figure 2.

This figure can be used for analyzing the evolution of the system Π: because the system is finite, the number of configurations reachable from the initial configuration is finite, too, hence, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Figure 2 there are also indicated the rules used in each neuron, with the following conventions: for each r_{jk} we have written only the subscript jk, with **31** being written in bold face, in order to indicate that a spike is sent out of the system at that step; when a neuron $\sigma_j, j = 1, 2, 3$ uses no rule, we have written $j0$, and when it spikes (after being closed for one step), we write js.

The functioning of the system, both as a number generator and as a string generator, can easily be followed on this diagram.

6 Open Problems and Research Topics

The following list of problems should be read with the standard precautions: it is not meant to be exhaustive, there is no ordering of the problems (according to their significance/interest), some problems are very general, others are much more particular, in many cases the formulation is preliminary/informal and addressing the problem should start with a precise/suitable formulation, in many cases related results exist in the literature, and so on. Most problems are stated in a short way, with reference to the discussion from Section 4 and the definitions from Section 5.

A. Let us start with a general and natural idea: linking the study of SN P systems with neural computing. This can be a rich source of ideas, based on trans-

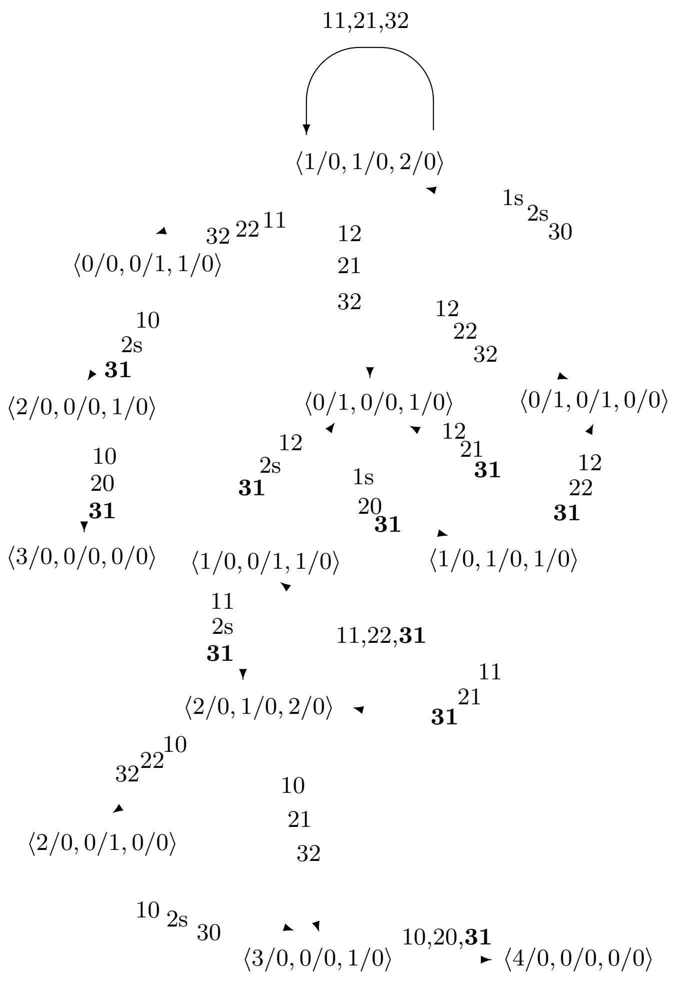

Fig. 2. The transition diagram of system Π from Figure 1

ferring from an area to the other one research topics which make sense also in the destination framework. What means, for instance, training (in general, learning, adaptation, evolving) in terms of SN P systems? More elementary: what means solving a problem by using an SN P system, implicitly, what means to solve a problem in a better way? Maybe the starting point should not be (only) neural computing, which is already an abstract, specialized, reductionistic framework, but (also) from neurology, from learning in the general psycho-pedagogical sense.

B. This problem is related to another general, natural, and important one: bringing more ingredients from neurology. Just a few quick ideas: considering an energy associated with firing/spiking; taking into considerations the antiport processes which are performed in synapses; introducing circadian periodicity in the functioning of neurons and of nets of neurons, with “tiredness”, “resting periods”, etc.

C. In particular, the recent discoveries related to the role of astrocytes in the functioning of the brain need to be examined and formalized. Astrocytes are a class of cells that form a supporting and insulating structure for the neurons, but also participate in the process of communication between neurons. They “listen” the spikes passing along axons and accordingly regulate the release of neurotransmitters from the nerve terminals, thus relating in an intricate way the functioning of different neighboring axons. The regulation is either excitatory or inhibitory, and it is done by means of calcium waves. I refer to [37] and [40] for further details – and further references. How can astrocytes be considered in an SN P system and with what consequences?

D. The neuron-astrocyte coupling is based on signaling pathways of a kind which reminds the controlling pathways which were recently modeled and simulated in terms of P systems in many papers, and this suggests the next general research challenge: applications (in neurology). This is perhaps a too ambitious goal at this stage of the development of the study of SN P systems and it is first necessary to have answers to the previous two problems, but it is important to keep in mind the possibility of applications when devising new classes of SN P systems. It is difficult to forecast which would be the most promising types of applications – looking for conceptual clarifications, for analytical results, for computer experiments and simulations, for all these intertwined? Of course, the cooperation with a biologist/neurologist would be very important in this respect.

E. Making a step from neurobiology to mathematics, the problem appears to consider systems using more than one type of spikes. At the first sight, this is against the spirit of spiking neural computing, and can lead to standard membrane systems. Still, the question makes sense in various setups. For instance, neurology deals both with excitatory and inhibitory impulses, both in neurons and at the level of astrocytes. How inhibitory spikes can be defined and used?

F. Then, there are features of SN P systems which were not considered for general P systems. Using a regular expression for enabling a rule looks like controlling the application of rules by means of promoters, inhibitors, activators, but a notion of delay does not exits in membrane computing. Can it be of any interest also for usual P systems? Then, defining the result of a computation in a P system in terms of the time elapsed between two specified events, in particular, sending a given object outside, was briefly investigated in [5], but this issue deserves further research efforts.

G. Conversely, there are many ingredients of usual P systems which were not considered for SN P systems and might make sense also in this area, at least at a mathematical level. Of a particular interest can be tools to exponentially increase the working space in a polynomial (if possible, even linear) time, for instance, by operations similar to cell division and cell creation in P systems with active membranes. How new neurons can be created (added to a system) in such a way to make possible polynomial solutions to computationally hard (typically, **NP**-complete) problems? The brain is supposed to be a very efficient computing device – how SN P systems can be made efficient from this point of view?

H. This touches a more general issue, that of considering SN P systems with a dynamical structure. The dynamism can be achieved both in terms of neurons and synapses, or only for synapses. From birth to maturity, the brain essentially evolves at the level of synapses, learning means establishing new synapses, cutting them, making them more stable/fast when used frequently, and so on and so forth. How this can be incorporated in SN P systems? A related idea is to associate a duration to each synapse (which is not of interest when the duration is constant), and to vary it in time, according to the intensity of using that synapse, and this looks rather motivated from a learning point of view.

I. Making synapses to have a duration or a length, depending on their use, can be related to a similar idea [8] at the level of spikes: considering a duration of life also for spikes, in the form of a decaying constant associated with them (at the level of the whole system, or locally, for each neuron). If a spike is not used a number of steps larger than the decaying threshold, then it is removed (a sort of forgetting rules are thus implicitly acting, depending on the age of each spike).

J. Moving further to theoretical issues, let us consider an idea related both to "classic" membrane computing and to the efficiency issue: using the rules in a parallel manner. This has been already considered in [24], in the particular form of using the rules in the exhaustive mode: if a neuron contains $kn + r$ spikes and has a rule $E/a^n \to a; d$ such that $a^{kn+r} \in L(E)$ and $k \geq 1, 0 \leq r < n$, then the rule is enabled and it is applied k times; kn spikes are consumed, r remain unused, and k are produced. Besides continuing the research from [24] (where it is only proved that SN P systems with an exhaustive use of rules are Turing complete both in the generative and the accepting modes), several other problems remain to be investigated. Actually, most problems usually considered for SN P systems with a sequential use of rules can be formulated also for the exhaustive mode: generating or accepting languages, translating strings of infinite sequences, looking for small universal systems, etc.

K. Then, the problem arises to consider other forms of parallelism, at the level of each neuron or at the level of the whole system. What about using several rules at the same time, in the same way as the rules of a usual P system are applied in the maximally parallel manner? Variants inspired from grammar systems area can also be considered, thus obtaining a bounded parallelism: at least k, at most k, exactly k rules to be used at a time. This last idea can be transferred also at the level of neurons: in each step, only a prescribed number of neurons, non-deterministically chosen, to be active. Finally, one can borrow to this area the idea of minimal parallelism from [15]: when a neuron can use at least one rule, then at least one must be used, without any restriction about how many. A significant non-determinism is introduced in this way in the functioning of the system.

L. When the number of rules to be used in each neuron is "at least zero" (and this is equivalent with making evolve "at least zero" neurons at a time), we get the rather natural idea of a non-synchronized functioning of an SN P system. In such a case, in each time unit, any neuron is free to use a rule or not.

I have described the functioning of such a system in the end of Section 4. I only recall that, because now "the time does not matter", the spike train can have arbitrarily many occurrences of 0 between any two occurrences of 1, hence the result of a computation can no longer be defined in terms of the steps between two consecutive spikes, but as the total number of spikes sent into the environment by (or contained in) the output neuron. In this way, only halting computations can be considered as successful.

In [7] it is proved that SN P systems with extended rules are Turing equivalent even in the non-synchronized case, but the problem was left open whether this is true also for systems using standard rules. The conjecture is that this does not happens, hence that synchronization plays a crucial role in this case.

Similar to the exhaustive mode of using rules, also the non-synchronization can be investigated in relation with many types of problems usual in the SN P systems area: handling languages, looking for small universal systems, etc.

M. A related issue is to consider the class of systems for which the synchronization does not matter, i.e., they generate/accept the same set of numbers in both modes. Furthermore, time-free, clock-free, time-independent systems can be considered, in the same way as in [4], [6], [38].

N. Several times so far, the idea of efficiency was invoked, with the need to introduce new ingredients in the area of SN P systems in such a way to make possible polynomial solutions to intractable problems. Actually, such a possibility was already considered in [10]: making use use of arbitrarily large pre-computed resources. The framework is the following: an arbitrarily large net of neurons is given, of a regular form (as the synapse graph) and with only a few types of neurons (as contents and rules) repeated indefinitely; the problem to be solved is plug-in by introducing a polynomial number of spikes in certain neurons (of course, polynomially many), then the system is left to work autonomously; in a polynomial time, it activates an exponential number of neurons, and, after a polynomial time, it outputs the solution to the problem. The problem considered in [10] was the `SAT` problem.

This strategy is attractive from a natural computing point of view (we may assume that the brain is arbitrarily large with respect to the small number of neurons currently used, the same with the cells in liver, etc.), but it has no counterpart in the classic complexity theory. A formal framework for defining acceptable solutions to problems by making use of pre-computed resources needs to be formulated and investigated. What kind of pre-computed workspace is acceptable, i.e., how much information may be provided for free there, what kind of net of neurons and what kind of neurons? (We have to prevent "cheating" by already placing the answer to the problem in the given resources and then "solving" the problem just by visiting the right place where the solution waits to be read.) What means introducing a problem in the existing device? (Only spikes, also rules, or maybe also synapses?) Defining complexity classes in this case remains as an interesting research topic.

O. Coming back to the initial definitions, there are several technical issues which are worth clarifying (most probably, for universality and maybe also for efficiency results, they do not matter, but it is also possible to exist other situations where these details matter). For instance, the self-synapses are not allowed in the synapse graph. However, a neuron with a rule $a \rightarrow a$ and a self-synapse can work forever, hence it can be used for rejecting a computation in the case when successful computations should halt. Similarly, (in the initial definition from [23]) the forgetting rules $a^s \rightarrow \lambda$ were supposed to have $a^s \notin L(E)$ for all spiking rules $E/a^c \rightarrow a; d$ from the same neuron, while in extended rules $E/a^c \rightarrow a^p; d$ it was assumed that $c \geq p$. Is there any situation where these restrictions make a difference? Then, in [19] it was shown that some of the ingredients used in the definition of SN P systems with standard rules can be avoided. This is the case with the delay, the forgetting rules, the generality of regular expressions. Can these normal forms be combined, thus avoiding at the same time two of the mentioned features?

P. What then about using a kind of rules of a more general form, namely $E/a^n \rightarrow a^{f(n)}; d$, where f is a partial function from natural numbers to natural numbers (maybe with the property $f(n) \leq n$ for all n for which f is defined), and used as follows: if the neuron contains k spikes such that $a^k \in L(E)$, then c of them are consumed and $f(c)$ are created, for $c = \max\{n \in \mathbf{N} \mid n \leq k$, and $f(n)$ is defined$\}$; if f is defined for no n smaller than or equal to k, then the rule cannot be applied. This kind of rules looks both adequate from a neurobiological point of view (the sigmoid excitation function can be captured) and powerful from a mathematical point of view (arbitrarily many spikes can be consumed at a time, and arbitrarily many produced).

Q. A standard problem when dealing with accepting devices concerns the difference between deterministic and non-deterministic systems. Are they different in power, does determinism imply a decrease of the computing power? Up to now, all computability completeness proofs for the accepting version of SN P systems of various types were obtained for deterministic systems. Are there classes (maybe non-universal) for which the determinism matters?

Actually, the problem can be refined. The determinism is defined usually in terms of non-branching during computations: a computation is deterministic if for every configuration there is (at most) one next configuration. A first subtle point: is this requested for *all* possible configurations or only for all configurations which are *reachable* from the initial one?

Maybe more interesting for SN P systems is the possibility to define a *strong determinism*, in terms of rules: an SN P system is said to be strongly deterministic if $L(E) \cap L(E') = \emptyset$ for all rules $E/a^c \to a; d$ and $E'/a^{c'} \to a; d'$ from any neuron. Obviously, such a system is deterministic also when defining this notion in terms of branching (even for arbitrary configurations, not only for the reachable ones).

Is any class of SN P systems for which these types of determinism are separated?

R. Different from the case of general P systems, where finding infinite hierarchies on the number of membranes was a long awaited result, for SN P systems one can easily find such hierarchies, based on the characterization of semilinear sets of numbers (by means of systems with a bounded number of spikes in their neurons): if for each finite automaton with n states (using only one symbol) one can find an equivalent SN P system with $g(n)$ neurons, and, conversely, for each SN P system with m neurons one can find an equivalent (i.e., generating strings over an one-letter alphabet whose lengths are numbers generated/accepted by the SN P system) with $h(m)$ states, then, because there is an infinite hierarchy of regular one-letter languages in terms of states, we get an infinite hierarchy of sets of numbers with respect to the number of neurons. Still, several problems arise here. First, not always the characterization of semilinear sets of numbers is based on proving the equivalence of bounded SN P systems with the finite automata. Then, this reasoning only proves that the hierarchy is infinite, not also that it is "dense" (*connected* is the term used in classic descriptional complexity: there is n_0 such that for each $n \geq n_0$ there is a set Q_n whose neuron-complexity is exactly n). Finally, what about finding classes intermediate between semilinear and Turing computable for which the hierarchy on the number of neurons is infinite (maybe connected)?

S. The previous question directly suggests two others. The first one is looking for small universal SN P systems (here "universal" is understood in the sense of "programmable" – the existence of a fixed system which can simulate any particular system after introducing a code of the particular system in it – not in the sense of "Turing complete", although there is a direct connection between these two notions). This question is considered in [30] for SN P systems with standard and with extended rules, working either in the computing mode or in the generating mode. For standard rules, 84 and 76 neurons were used, while for extended rules 49 and 50 neurons were used, respectively. Are these results optimal? A negative answer is expected (however, a significant improvement is not very probable). What about universal SN P systems of other types – in particular, with exhaustive or non-synchronized use of rules?

T. Problem **R** also suggests to look for classes of SN P systems which are not equivalent with Turing machines, but also not computing only semilinear sets of numbers, hence equivalent in power with finite automata. This does not look as an easy question, but it is rather interesting, in view of the possibility of finding classes of systems with decidable properties, but (significantly) more powerful than bounded SN P systems. Such a class would be attractive also from the point of view of applications, because of the possibility of finding properties of the modeled processes by analytical, algorithmic means.

U. Again in a direct continuation with the previous issue, there appears the need to find characterizations of classes of languages, other than finite, regular, and recursively enumerable, in terms of SN P systems. The investigations from [9], [12], [14] have left open these questions, and this fits with the general situation in membrane computing (as well as in DNA computing): the Chomsky hierarchy seems not to have a counterpart in nature, families like those of linear, context-free, and context-sensitive languages do not have (easy) characterizations in bio-inspired computing models. The same challenge appears for families of languages generated by L systems (sometimes, with the exception of ET0L languages).

V. L systems can be related with SN P systems also at the level of infinite sequences: both by iterating morphisms (D0L systems) and by taking infinite spike trains we can get classes of infinite sequences. Directly as spike trains we have binary sequences, but, for extended rules (and for SN P systems with a parallel use of rules) we can get as an output of a computation a string or an infinite sequence over an arbitrary alphabet. A preliminary examination of the binary case was done in [34], but many problems were left open, starting with the comparison of SN P systems as tools for handling infinite sequences (of bits) with other tools from language and automata theory (with ω-languages computed by finite automata, Turing machines, etc.) and with known infinite sequences, e.g., those from [42].

A particular problem from [34] is the following. SN P systems cannot compute arbitrary morphisms, but only length preserving morphisms (codes). An extension of these latter functions are the so-called *k-block morphisms*, which are functions $f : \{0,1\}^k \longrightarrow \{0,1\}^k$ (for a given $k \geq 1$) prolonged to $f : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ by $f(x_1x_2\ldots) = f(x_1)f(x_2)\ldots$. In [34] it is only shown that 2-block morphisms can be computed by SN P systems, and the conjecture was formulated that this is true for any k.

In general, more should be found about the use of SN P systems as tools for transducing strings and infinite sequences.

W. Maybe useful in addressing the previous problem – and interesting also from other points of view (e.g., if starting investigations in terms of process algebra), is the issue of compositionality: looking for ways to pass from given systems to more complex systems, for instance, to systems generating/accepting the result of an operation between the sets of numbers or the languages generated/accepted by the initial systems. Morphisms were mentioned also above, but there are many other set-theoretic or language-theoretic operations to consider, as well as serial and parallel composition, embedding as a subsystem, etc. Of course, a central point in such operations is that of synchronization. It is expected that the case of non-synchronized systems is much easier (maybe, instead, less interesting theoretically).

X. I have mentioned at the beginning of these notes that the axon is not a simple transmitter of spikes, but a complex information processor. This suggests considering computing models based on the axon functioning (Ranvier nodes amplification of impulses, and other processes) and a preliminary investigation was carried out in [13]. Many questions remain to be clarified in this area (see also the questions formulated in [13]), but a more general and probably more interesting problem appears, namely, of combining neurons and axons (as information processing units) in a global model; maybe also astrocytes can be added, thus obtaining a more complex model, closer to reality.

Y. I will conclude with two general issues, where nothing was done up to now. First, SN P systems have a direct (pictural) similarity with Petri nets, where tokens (like spikes) are moved through the net according to specific rules. Bridging the two areas looks then rather natural – with "bridging" understood as a move of notions, tools, results in both directions, from Petri nets to SN P systems and the other way round.

Z. Then, directly important for possible applications is the study of SN P systems as dynamical systems, hence not focusing on their output, but on their evolution, on the properties of the sequences of configurations reachable from each other. The whole panoply of questions from the (discrete) dynamical systems theory can be brought here, much similar to what happened in general membrane computing.

As it was the case also other times, I have to stop because of reaching the end of the alphabet... – with the hope that the reader will shorten this list by providing answers to some problems.

7 Final Remarks

Many other open problems and research topics can be found in the papers devoted to SN P systems – the interested reader can check the titles below in this respect (the bibliography contains all papers about SN P systems which I was aware of at the beginning of November 2006). On the other hand, because the research in this area is quite vivid, it is possible that some of these problems were solved at the same time or shortly after writing these notes, without being possible to mention the respective results here. That is why, the reader is advised to follow the developments in this area, for instance, through the information periodically updated at the Milano web page [43].

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.
2. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended variants of spiking neural P systems generating strings and vectors of non-negative integers. In *Pre-proc. WMC7*, Leiden, July 2006, 88–101, and [18], 123–134.

3. A. Carnell, D. Richardson: Parallel computation in spiking neural nets. Available at `http://people.bath.ac.uk/masdr/`.
4. M. Cavaliere, V. Deufemia: On time-free P systems. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 69–90.
5. M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-related outputs of computations in P systems. In *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 107–122.
6. M. Cavaliere, D. Sburlan: Time-independent P systems. In *Membrane Computing. International Workshop WMC5, Milano, Italy, 2004*, LNCS 3365, Springer, 2005, 239–258.
7. M. Cavaliere, M. Ionescu, Gh. Păun: Asynchronous spiking neural P systems. Submitted, 2007.
8. M. Cavaliere, M. Ionescu: SN P systems with decaying spikes. Personal communication, 2006.
9. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [17], Vol. I, 169–194.
10. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. In [17], Vol. I, 195–206, and *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
11. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: Universality and languages. *Natural Computing*, to appear.
12. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. In [17], Vol. I, 207–224, and *Proc. Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.
13. H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. In [17], Vol. I, 225–240, and *Pre-proc. BIC-TA*, Wuhan, 2006, 60–70.
14. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [17], Vol. I, 241–265.
15. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism. *Theoretical Computer Sci.*, to appear.
16. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.
17. M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.
18. H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing, International Workshop, WMC7, Leiden, The Netherlands, 2006, Revised, Selected, and Invited Papers*. LNCS 4361, Springer, Berlin, 2006.
19. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [17], Vol. II, 105–136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
20. O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. In *Pre-proc. WMC7*, Leiden July 2006, 387–396, and [18], 424–442.
21. O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. In *Proc. UC2006*, York, LNCS 4135, Springer, 2006, 113–129.
22. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. In *Proc. DNA12* (C. Mao, Y.

Yokomori, B.-T. Zhang, eds.), Seoul, June 2006, 32–42.

23. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
24. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, to appear.
25. I. Korec: Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.
26. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
27. W. Maass: Paradigms for computing with spiking neurons. In *Models of Neural Networks. Early Vision and Attention* (J.L. van Hemmen, J.D. Cowen, E. Domany, eds.), Springer, Berlin, 2002, 373–402.
28. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
29. C. O'Dwyer, D. Richardson: Spiking neural nets with symbolic internal state. Available at `http://people.bath.ac.uk/masdr/`.
30. A. Păun, Gh. Păun: Small universal spiking neural P systems. In [17], Vol. II, 213–234, and *BioSystems*, in press.
31. Gh. Păun: *Membrane Computing – An Introduction.* Springer, Berlin, 2002.
32. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. In *Proc. of Developments in Language Theory Conference, DLT 2006*, Santa Barbara, CA, June 2006, LNCS 4036, Springer, Berlin, 2006, 20–35.
33. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
34. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
35. Gh. Păun, M.J. Perez-Jimenez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
36. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Spiking neural P systems. An early survey. *Intern. J. Fund. Computer Sci.*, 2007.
37. G. Perea, A. Araque: Communication between astrocytes and neurons: a complex language. *J. Physiol. – Paris*, 96 (2002), 199–207.
38. D. Sburlan: *Promoting and Inhibiting Contexts in Membrane Computing.* PhD Thesis, Univ. Sevilla, Spania, 2006.
39. I. Segev, E. Schneidman, Axons as computing devices: basic insights gained from models. *J. Physiol. (Paris)*, 93 (1999), 263–270.
40. X. Shen, P. De Wilde: Long-term neuronal behavior caused by two synaptic modification mechanisms. *Neurocomputing*, to appear.
41. G.M. Shepherd: *Neurobiology.* Oxford University Press, NY Oxford, 1994.
42. N.J.A. Sloane, S. Plouffe: *The Encyclopedia of Integer Sequences.* Academic Press, New York, 1995.
43. The P Systems Web Page: `http://psystems.disco.unimib.it`.