Spiking Neural P Systems. A Quick Survey and Some Research Topics

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
Email: gpaun@us.es

***Abstract*—After presenting the basic definition of spiking neural P systems (SN P systems), illustrated with two examples, we recall some results concerning the computing power and the size of universal SN P systems. We end this note with a couple of research topics.**

***Keywords*-membrane computing; P system; spiking neural P system; computational complexity;**

I. Introduction

Membrane computing is a branch of natural computing which abstracts computing models from the structure and the functioning of the biological cell. Initially ([19]), isolated cells were considered, then also cell communities, such as tissues, organs, colonies (of bacteria) were investigated. The obtained models (called P systems) are distributed computing devices, working in parallel, processing multisets in the compartments defined by membranes. These systems were proved to be computationally complete, equivalent in power with Turing machines, and, if possibilities were provided to generate an exponential working space during the computation (e.g., by means of cell division), then computationally hard problems, typically, **NP**-complete problems, were proved to be solved in a polynomial time (by a time-space trade-off). Also, a series of applications were reported, in biology and biomedicine, computer graphics, cryptography, robot control, distributed evolutionary computing. Details can be found, e.g., in [22] and at the domain website, [24].

Since the first years of membrane computing development attempts were made to also model the neuron cell as well as the networks of neurons. The neural-like P systems introduced in [14] consider one-membrane neurons, in the nodes of a graph whose edges correspond to synapses, with multisets of objects in the neurons and with a state associated with each neuron. The used rules both change the states of neurons and the objects inside them.

A model much closer to the actual neurons, modeling the way the neurons communicate through spikes (see, e.g., [12] for biological details), was introduced in [10], under the name of spiking neural (SN) P systems. The bibliography of this research direction is rather large (at the level of the year 2009 a list of titles is provided in [20]). This class of P systems will be quickly presented in this paper.

In short an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. The fundamental feature of the model is that we deal with a unique object, the spike (electrical impulses of identical shape – voltage, duration – sent from a neuron to another one along the synapses). Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the spike train itself, interpreted as a a binary sequence, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

When *extended* SN P systems are considered, that is, able to produce several spikes at the same time, strings over arbitrary alphabets can be obtained: if i spikes are sent out at the same time, then we say that the symbol b_i was generated.

A large number of computing devices can be obtained, subject of the usual research program in computer science: power, efficiency, normal forms, relations with other computing models, applications, simulation on usual computers, implementations.

After giving the formal definition of the extended SN P systems, we illustrate it with two simple examples, then we recall a few results about their computing power and a series of variants, and we end with some research topics which look interesting from a mathematical and computational point of view.

II. Extended SN P Systems

We assume the reader to be familiar with some basic automata and language theory notions, so that we do not introduce them here (Chomsky hierarchy, regular expressions), and we only mention a few notations. FIN, REG, RE are the families of finite, regular and recursively enumerable languages, while $NFIN, NREG, NRE$ are the families of length sets of these languages (NRE is thus the family of Turing computable sets of numbers).

978-1-4799-8448-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SYNASC.2014.11

The "standard" way of proving the equivalence of SN P systems with Turing machines is to simulate register machines, [16], that is why we also recall their definition: such a device is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \mathtt{HALT}$ (the halt instruction).

A register machine generates a set of numbers by starting with all registers empty (storing number zero); when the machine halts (it reaches instruction with label l_h), the number contained in register 1 is said to be generated by M. Similarly, a number placed initially in register 1 is said to be accepted by M if the machine eventually halts.

An extended SN P system of degree $m \geq 1$ is a construct of the form

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

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 in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a^p; d$, where E is a regular expression over a, and $c \geq p \geq 1$, $d \geq 0$;
      (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
4) $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

The rules of type (1) are *spiking rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a^p; d$ can be applied. The application of this rule means removing c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces p spikes after d time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spikes are emitted immediately, if $d = 1$, then the spikes are emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send spikes along it, then those particular spikes are lost). In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$).

The rules of type (2) are *forgetting* rules and they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \to a^p; d$ of type (1) has $E = a^c$, then we will write it in the simplified form $a^c \to a^p; d$. If in all rules we have $d = 0$, then we say that the system is without delay and the parameter d is omitted when writing the rules. Rules $E/a^c \to a; d$ are non-extended (this was the variant introduced in [10], while extended rules were introduced in [3]).

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used, nondeterministically choosing the rule to apply. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

During a computation, the configuration of the system is described by the vector $\langle r_1/t_1, \ldots, r_m/t_m \rangle$, stating that neuron σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, for $i = 1, 2, \ldots, m$; thus, the initial configuration of the system is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$ (initially, all neurons are open).

Using the rules as described above, one 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*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

An SN P system can be used in various ways. In the generative mode, we start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron; the subsequent behavior of the system (spiking again or not, halting or not) is ignored. We denote by $N_2(\Pi)$ the set of numbers computed in this way by an SN P system Π. We can also use Π in the accepting mode: a number n is introduced in the system as the number of steps between two spikes which enter neuron σ_{in} from the environment, and the number n is accepted if and only if the computation halts. Similarly, an SN P system can be used in the computing mode (an output is computed, starting from a given input). In the generative case, the neuron (with label) in is ignored, in the accepting mode the neuron out is ignored.

We denote by $N_2SNP_m(rules_k, cons_c)$ the families of all sets $N_2(\Pi)$ generated by SN P systems with at most $m \geq 1$ neurons, $k \geq 1$ rules in each neuron, consuming at most $c \geq 1$ spikes in spiking rules, and without using forgetting rules and the delay feature. (We ignore the forgetting rules and the delay, because it was proven that these features can be removed without decreasing the computing power, [9].)

III. Two Examples

The first example has a more general significance: this is a part of a larger SN P system which simulates a register machine. Here we have the module which simulates a SUB instruction, $l_i : (\mathtt{SUB}(r), l_j, l_k)$. Figure 1 presents this module in the standard way of representing graphically an SN P system.

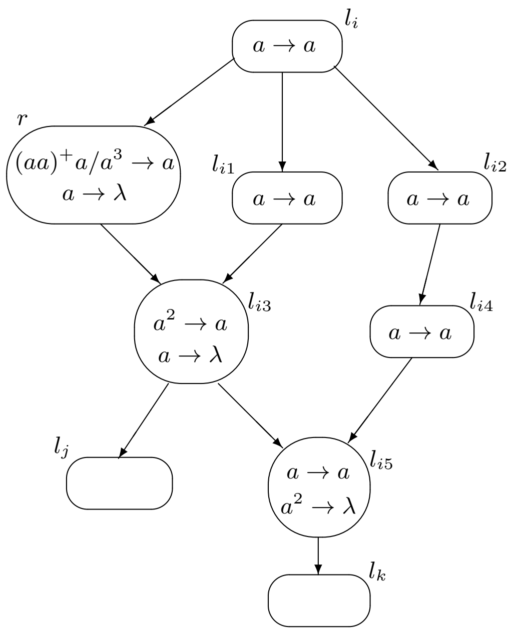

Figure 1. A SUB module, simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$.

A neuron σ_r is associated with each register; if the register contains the number n, then the corresponding neuron contains $2n$ spikes. (Thus, increasing or decreasing register r by 1 corresponds to adding to or removing 2 spikes from σ_r.) Neurons are also associated with labels. There are also auxiliary neurons. Initially, all neurons are empty. The computation starts after introducing a spike in neuron σ_{l_0}. In general, a module as above (similar modules are associated with ADD instructions) becomes active when a spike is introduced in σ_{l_i}. Immediately, neuron σ_{l_i} fires, sending one spike to each neuron (with labels) r, l_{i1}, l_{i2}.

If register r is empty, then neuron σ_r can use only the rule $a \to \lambda$. Simultaneously, neurons l_{i1}, l_{i2} fire. Neuron l_{i3} has only one spike, and forgets it, while neuron l_{i4} send one spike to neuron l_{i5}, which, in turn, will send a spike to σ_{l_k}, thus activating the module associated with the instruction $l_k : (\mathtt{OP}(r), l_u, l_v)$.

If register r is not empty, then σ_r contains now at least 3 spikes and it will use the rule $(aa)^+a/a^3 \to a$, which both decreases by 2 the number of spikes in the neuron associated with the register r and sends a spike to neuron l_{i3}. Now, in the end σ_{l_j} will get a spike and σ_{l_k} not, which means that the module associated with the instruction $l_j : (\mathtt{OP}(r), l_u, l_v)$ is activated.

In both cases, the simulation of the SUB instruction is correct.

In the next example (Figure 2), the SN P system computes a Boolean function of three variables, namely,

$$f : \{0,1\}^3 \longrightarrow \{0,1\},\ f(b_1, b_2, b_3) = 1 \text{ iff } b_1+b_2+b_3 \neq 2.$$

We leave to the reader the task of checking the way the system works – as well as to generalize the idea of this construction to computing *any* Boolean function, of any number of variables.

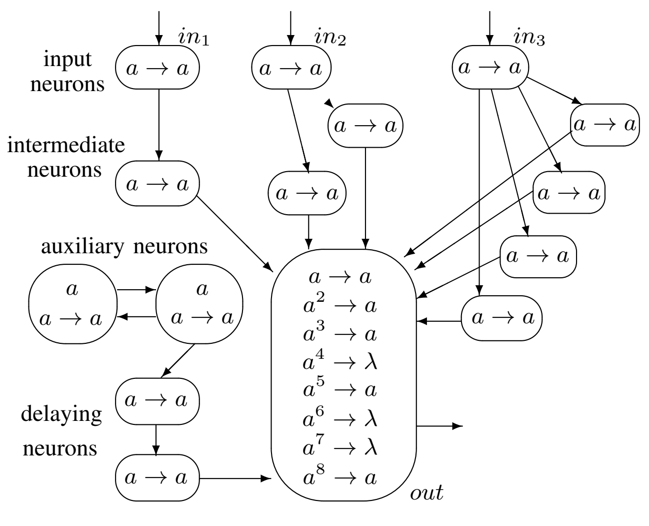

Figure 2. Computing a Boolean function

IV. Basic Results

The following results give a hint on the computing power of SN P systems (here, we deal with the generating mode, but similar results hold true also for the accepting mode).

Theorem 1: (i) $NFIN = N_2SNP_1(rule_*, cons_1) = N_2SNP_2(rule_*, cons_*)$.

(ii) $N_2SNP_*(rule_k, cons_p) = NRE$ for all $k \geq 2$, $p \geq 3$.

(iii) SN P systems with a bound on the number of spikes in their neurons characterize $NREG$.

Note that in the previous characterization of NRE we do not get a bound on the number of neurons. Such a bound can be found if we start the proof from universal register machines, as those in [8]. The first results of this type were reported in [18]:

Theorem 2: There is a universal computing SN P system with standard rules having 84 neurons and a universal computing SN P system with extended rules (without delay) which has 49 neurons.

These results were repeatedly improved – we refer to the literature for details (as well as for results concerning the power of SN P systems as language generators – even as infinite sequences processors).

V. A Series of Variants

Starting from the basic model, many variants of SN P systems were considered, with motivation coming from biology (e.g., considering further ingredients, such as astrocytes), computer science and mathematics. We quickly list here some of them, with minimal details; references can be found in [22] and in the bibliography provided at [24]:

1) parallel/exhaustive use of rules: if enabled, a rule is used as many times as possible in a neuron
2) asynchronous: neurons fire or not (in the extended case, asynchronous SN P systems are universal, but the problem is open for non-extended systems)
3) sequential: one rule is used in the whole system (a possibility to choose the rules to apply is to look for the neuron with the smallest/biggest number of spikes)
4) homogeneous: considering systems with the same set of rules in each neuron
5) locally synchronized: modules (subsets of neurons) are considered, and in each step one module is used, with its neurons working synchronously
6) time-free: a time-function is considered, associating a duration with each rule execution; the system is time-free if it provides the same result irrespective of the associated time-function
7) weighted: "multipliers" are added to synapses and the number of transmitted spikes is increased by these multipliers
8) systems with rules on synapses, not in neurons
9) systems with anti-spikes: besides a, also $\bar{a}$ is considered, subject to usual spiking and forgetting rules, but also considering an implicit rule $a\bar{a} \rightarrow \lambda$ (when a spike meets an anti-spike, they annihilate, and this operation takes no time and, moreover, has priority over spiking and forgetting rules); interesting enough, this feature add efficiency to SN P systems: **NP**-complete problems can be solved in polynomial time by SN P systems with anti-spikes
10) SN P systems with neuron division or neuron budding: this is the standard way for speeding-up P systems, and it works also for SN P systems
11) systems with astrocytes: these are cells which sense the spike traffic along axons and, according to that, control the feeding of neurons (so they add further "programming" possibilities to SN P systems)
12) dP SN P systems: this is a combination of SN P systems with the dP systems from [21], thus handling in a distributed way the complex problems, which are split into subproblems and processed separately, on modules which cooperate during the computation
13) computing along the axon: arranging a sort of neurons along a line (corresponding to the Ranvier nodes in neurology)
14) combinations of these

Universality is obtained in most cases (not, for instance, for computing along the axon).

VI. Applications and Software

We only mention here the papers [6], [15], [25], [26], where some applications are described, as well as the papers [1], [13], [27], where simulators (sometimes, called implementations) are reported. General information about the P-lingua programming language (also a platform for executing P systems) can be found at the P-lingua website, `http://www.p-lingua.org`.

VII. Research Topics

There are many specific open problems in this area (several of them can be found in the volumes emerged from the yearly meetings Brainstorming Week on Membrane Computing – see information at [24]), but here we only mention some more general issues, dealing with research directions which were not explored so far.

For instance, there is no attempt to imagine a class of SN P systems able to compute beyond Turing. Hypercomputing is a much explored area in theoretical computer science, and there also are some attempts in this direction in membrane computing. We mention only [2] and [23]. Can the ideas explored in these papers be extended also to SN P systems? The question is of interest in view of the common assertion that the brain is able of super-Turing computing. How this can be achieved in terms of SN P systems?

A second research topic, formulated also in other contexts (see, e.g., [7]), is the deeper investigation of the using of pre-computed resources as a strategy for solving computationally hard problems. An example of having at work this strategy can be found in [11]. Other applications are to be found, and, more important, to establish a formal framework for this strategy (to define formally allowed pre-computed resources, their complexity, and then to define complexity classes in this framework).

Two more precise and more technical questions are (i) to consider also regular expressions associated with astrocytes (up to now, the astrocytes have associated thresholds, and they act, in an excitatory or inhibitory manner, depending on these thresholds), and (ii) to use SN P systems as computing devices for numerical functions (n arguments are introduced in n neurons, as spike trains or as numbers of spikes placed in these neurons) and m values are obtained in corresponding output neurons). Are such function computing devices of any usefulness? Which is the their efficiency? (This should be compared with the use of numerical P systems in robot control, see, e.g., [17].)

In there recent paper [5], one proposes to consider P systems with *white holes*, symmetrical to black holes, i.e., sending out whatever they contain. This can be easily captured in terms of SN P systems by assuming that each white hole neuron contains all rules of the form $a^n \to a^n$, for all $n \geq 1$. How an SN P system with one (or several) white hole behaves? What about SN P systems composed of only white hole neurons?

VIII. Final Remarks

We conclude this note with the belief that the study of SN P systems is both interesting/challenging from a mathematical point of view, and also of interest for applications – especially in the latter direction further research efforts are necessary.

References

[1] F. Cabarle, H. Adorna, and M. A. Martinez-del-Amor, Simulating spiking neural P systems without delay using GPUs. *Proc. 9th Brainstorming Week on Membrane Computing*, Sevilla, 2011, 23–42.

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

[3] H. Chen and T.-O. Ishdorj, Gh. Păun, and M. J. Pérez-Jiménez, Spiking neural P systems with extended rules. *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, 241–265.

[4] H. Chen, T.-O. Ishdorj, Gh. Păun, and M. J. Pérez-Jiménez, Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.

[5] E. Csuhaj-Varjú, M. Gheorghe, and G. Vaszil, P systems – A formal approach to social networks (Abstract). *Proc. 15th Conf. on Membrane Computing*, Prague, 2014, 7–10.

[6] D. Díaz-Pernil, F. Peña-Cantillana, and M. A. Gutiérrez-Naranjo, Skeletonizing images by using spiking neural P systems. *Proc. 8th Brainstorming Week on Membrane Computing*, Sevilla, 2010, vol. I, 91–110.

[7] M. Gheorghe, Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, Frontiers of membrane computing: Open problems and research topics. *Intern. J. Found. Computer Sci.*, 24, 5 (2013), 547–623.

[8] I. Korec, Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.

[9] O. H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Paton, P. Sosik, and S. Woodworth, Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.

[10] M. Ionescu, Gh. Păun, and T. Yokomori, Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.

[11] T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang, Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Proc. 7th Brainstorming Week on Membrane Computing*, Sevilla, 2009, vol. II, 1–27.

[12] W. Maass, and C. Bishop, eds., *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.

[13] L. F. Macias-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M. J. Pérez-Jiménez, and A. Riscos-'uñez, A P-lingua based simulator for spiking neural P systems. *Proc. 12th Conf. on Membrane Ccomputing*, Fontainebleau, 2011, LNCS 7184, 257–281.

[14] C. Martin-Vide, J. Pazos, Gh. Păun, and A. Rodriguez-Paton, Tissue P systems. *Theoretical Computer Sci.*, 296, 2 (2003), 295–326.

[15] J. M. Mingo, Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. *Proc. 7th Brainstorming Week on Membrane Computing*, Sevilla, 2009, vol. II, 59–72.

[16] M. Minsky, *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[17] A. B. Pavel and C. Buiu, Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, 11, 3 (2012), 387–393.

[18] A. Păun and Gh. Păun, Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.

[19] Gh. Păun, Computing with membranes. *J. Computer and System Sciences*, 61, 1 (2000), 108–143.

[20] Gh. Păun, A bibliography of spiking neural P systems. *Proc. 7th Brainstorming Week on Membrane Computing*, Sevilla, 2009, 207–212.

[21] Gh. Păun and M. J. Pérez-Jiménez, Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.

[22] Gh. Păun, G. Rozenberg, and A. Salomaa, eds., *Handbook of Membrane Computing*. Oxford University Press, 2010.

[23] P. Sosik and O. Valik, On evolutionary lineages of membrane systems. *Proc. 6th Conf. on Membrane Computing*, Vienna, 2005, LNCS 3850, 67–78.

[24] The P Systems Website, `http://ppage.psystems.eu`.

[25] J. Wang and H. Peng, Adaptive fuzzy spiking neural P systems for fuzzy inference and learning. *Proc. 8th Brainstorming Week on Membrane Computing*, Sevilla, 2010, vol. I, 235–248.

[26] T. Wang, G. Zhang, and M. J. Pérez-Jiménez, Application of weighted fuzzy reasoning spiking neural P systems to fault diagnosis in traction power supply systems of high-speed railways. *Proc. 10th Brainstorming Week on Membrane Computing*, Sevilla, 2012, 329–350.

[27] Z. Xu, M. Cavaliere, P. An, S. Vrudhula, and Y. Cao, The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits. *Fundamenta Informaticea*, to appear.