

Spiking Neural P Systems. A Tutorial

Gheorghe PĂUN

Institute of Mathematics of the Romanian Academy

PO Box 1-764, 014700 București, Romania

E-mail: george.paun@imar.ro, gpaun@us.es

Abstract. We briefly present (basic ideas, some examples, classes of spiking neural P systems, some results concerning their power, research topics) a recently initiated branch of membrane computing with motivation from neural computing. Further details can be found at the web page of membrane computing, from <http://psystems.disco.unimib.it>.

1 The General Framework

The most intuitive way to introduce spiking neural P systems (in short, SN P systems) is by watching the movie available at http://www.igi.tugraz.at/tnatschl/spike_trains_eng.html, in the web page of Wolfgang Maass, Graz, Austria: neurons are sending to each others *spikes*, electrical impulses of identical shape (duration, voltage, etc.), with the information “encoded” in the frequency of these impulses, hence in the time passes between consecutive spikes. For neurologists, this is nothing new, related drawings already appears in papers by Ramón y Cajal, a pioneer of neuroscience at the beginning of the last century, but in the recent years “computing by spiking” is a vivid research area, with the hope to lead to a neural computing “of the third generation” – see [12], [21], etc.

For membrane computing it is somehow natural to incorporate the idea of spiking neurons (already neural-like P systems exist, based on different ingredients – see [23], efforts to compute with a small number of objects were recently made in several papers – see, e.g., [2], using the time as a support of information, for instance, taking the time between two events as the result of a computation, was also considered – see [3]), but still important differences exist between the general way of working with multisets of objects in the compartments of a cell-like membrane structure – as in membrane computing – and the way the neurons communicate by spikes. A way to answer this challenge was proposed in [18]: neurons as single membranes, placed in the nodes of a graph corresponding to synapses, only one type of objects present

in neurons, the spikes, with specific rules for handling them, and with the distance in time between consecutive spikes playing an important role (e.g., the result of a computation being defined either as the whole spike train of a distinguished output neuron, or as the distance between consecutive spikes). Details will be given immediately.

What is obtained is a computing device whose behavior resembles the process from the neuron nets, meant to generate strings or infinite sequences (like in formal language theory), to recognize or translate strings or infinite sequences (like in automata theory), to generate or accept natural numbers, or to compute number functions (like in membrane computing). Results of all these types will be mentioned below. Nothing is said here, because nothing was done so far, about using such devices in “standard” neural computing applications, such as pattern recognition. Several open problems and research topics will be mentioned below (a long list of such topics, prepared for the Fifth Brainstorming Week on Membrane Computing, Sevilla, January 29-February 2, 2006, can be found in [24]), but probably this is the most important one: connecting SN P systems with neural computing, more generally, looking for applications of SN P systems.

It is worth mentioning here that “general” membrane computing is now an area of intense research related to applications, mainly in biology/medicine, but also in economics, distributed evolutionary computing, computer graphics, etc. (see [11], [27], [28]), but this happens after a couple of years of research of a classic language-automata-complexity type; maybe this will be the case also for the spiking neural P systems, which need further theoretical investigation before passing to applications.

2 An Informal Overview – With An Example

Very shortly, an SN P system consists of a set of *neurons* (cells, consisting of only one membrane) placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along *synapses* (arcs of the graph). Thus, the architecture is that of a tissue-like P system, with only one kind of objects present in the cells. 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. We say that the rules “cover” the neuron, all spikes are taken into consideration when using a rule.

The system works in a synchronized manner, i.e., in each time unit, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: 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. The binary sequence obtained in this way is called the *spike train* of the system – it might be infinite if the computation does not stop.

Figure 1 recalls an **example** from [18], and this also introduces the standard way to represent an SN P system (note that the output neuron, σ_7 in this case, is indicated by an arrow pointing to the environment), and a simplification in writing the spiking rules: if we have a rule $E/a^c \rightarrow a; d$ with $L(E) = \{a^c\}$, then we write simply $a^c \rightarrow a; d$. If all rules are of this form, then the system is called *bounded* (or *finite*), because it can handle only finite numbers of spikes in the neurons.

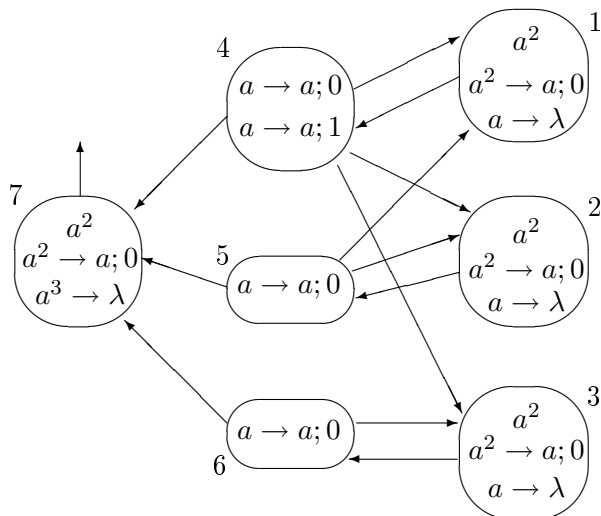


Figure 1: An SN P system generating all even natural numbers

In the beginning, only neurons $\sigma_1, \sigma_2, \sigma_3$, and σ_7 contain spikes, hence they fire in the first step – and spike immediately. In particular, the output neuron spikes, hence a spike is also sent to the environment. Note that in the first step we cannot use the forgetting rule $a \rightarrow \lambda$ in $\sigma_1, \sigma_2, \sigma_3$, because we have more than one spike present in each neuron.

The spikes of neurons $\sigma_1, \sigma_2, \sigma_3$ will pass to neurons $\sigma_4, \sigma_5, \sigma_6$. In step 2, $\sigma_1, \sigma_2, \sigma_3$ contain no spike inside, hence will not fire, but $\sigma_4, \sigma_5, \sigma_6$ fire. Neu-

rons σ_5, σ_6 have only one rule, but neuron σ_4 behaves non-deterministically, choosing between the rules $a \rightarrow a; 0$ and $a \rightarrow a; 1$. Assume that for $m \geq 0$ steps we use here the first rule. This means that three spikes are sent to neuron σ_7 , while each of neurons $\sigma_1, \sigma_2, \sigma_3$ receives two spikes. In step 3, neurons $\sigma_4, \sigma_5, \sigma_6$ cannot fire, but all $\sigma_1, \sigma_2, \sigma_3$ fire again. After receiving the three spikes, neuron σ_7 uses its forgetting rule and gets empty again. These steps can be repeated arbitrarily many times.

In order to have neuron σ_7 firing again, we have to use sometimes the rule $a \rightarrow a; 1$ of neuron σ_4 . Assume that this happens in step t (it is easy to see that $t = 2m + 2$). This means that at step t only neurons σ_5, σ_6 emit their spikes. Each of neurons $\sigma_1, \sigma_2, \sigma_3$ receives only one spike – and forgets it in the next step, $t + 1$. Neuron σ_7 receives two spikes, and fires again, thus sending the second spike to the environment. This happens in moment $t + 1 = 2m + 2 + 1$, hence between the first and the second spike sent outside have elapsed $2m + 2$ steps, for some $m \geq 0$. The spike of neuron σ_4 (the one “prepared-but-not-yet-emitted” by using the rule $a \rightarrow a; 1$ in step t) will reach neurons $\sigma_1, \sigma_2, \sigma_3$, and σ_7 in step $t + 1$, hence it can be used only in step $t + 2$; in step $t + 2$ neurons $\sigma_1, \sigma_2, \sigma_3$ forget their spikes and the computation halts. The spike from neuron σ_7 remains unused, there is no rule for it. Note the effect of the forgetting rules $a \rightarrow \lambda$ from neurons $\sigma_1, \sigma_2, \sigma_3$: without such rules, the spikes of neurons σ_5, σ_6 from step t will wait unused in neurons $\sigma_1, \sigma_2, \sigma_3$ and, when the spike of neuron σ_4 will arrive, we will have two spikes, hence the rules $a^2 \rightarrow a; 0$ from neurons $\sigma_1, \sigma_2, \sigma_3$ would be enabled again and the system will continue to work.

Let us return to the general presentation. In the spirit of spiking neurons, in the basic variant of SN P systems introduced in [18], the result of a computation is defined as the distance between consecutive spikes sent into the environment by the (output neuron of the) system. In [18] only the distance between the first two spikes of a spike train was considered, then in [25] 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.

Therefore, as seen above, the system Π from Figure 1 computes the set $N_2(\Pi) = \{2n \mid n \geq 1\}$ – where the subscript 2 reminds that we consider the distance between the first two spikes sent to the environment.

Systems working in the accepting mode were also considered: 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 (hence for finite SN P systems).

Another attractive possibility is to consider the spike trains themselves as the result of a computation, and then we obtain a (binary) language generating device. We can also consider input neurons and then an SN P system can work as a transducer. Such possibilities were investigated in [26]. Languages – even on arbitrary alphabets – can be obtained also in other ways: following the path of a designated spike across neurons, or using extended rules. 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 [9].

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 [20], one can find small universal SN P systems. This idea was explored in [22] – the results are recalled in Theorem 5.4.

In the initial definition of SN P systems several ingredients are used (delay, forgetting rules), some of them of a general form (general synapse graph, general regular expressions). As shown in [15], rather restrictive 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 removed, both the indegree 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.

There were investigated several other types of SN P systems: with several output neurons ([16], [17]), with a non-synchronous use of rules ([4]), with an exhaustive use of rules (whenever enabled, a rule is used as much as possible for the number of spikes present in the neuron, [19]), with packages of spikes sent along specified synapse links ([1]), etc. We refer the reader to the bibliography of this note, with many papers being available at [27].

3 A Formal Definition

We introduce the SN P systems in a general form, namely, in the extended (i.e., with the rules able to produce more than one spike) computing (i.e., able to take an input and provide an output) version.

A computing extended *spiking neural P system*, of degree $m \geq 1$, is a

construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, \text{in}, \text{out}), \text{ where:}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \dots, \sigma_m$ 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) $E/a^c \rightarrow a^p; d$, where E is a regular expression over a and $c \geq p \geq 1, d \geq 0$;
 - (2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ of type (1) from R_i , we have $a^s \notin L(E)$;
3. $\text{syn} \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in \text{syn}$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $\text{in}, \text{out} \in \{1, 2, \dots, m\}$ indicate the *input* and the *output* neurons, respectively.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, those of type (2) are called *forgetting* rules. An SN P system whose firing rules have $p = 1$ (they produce only one spike) is said to be of the *standard* type (non-extended).

The firing rules are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a^p; d \in R_i$ can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in σ_i ; this corresponds to the right derivative operation $L(E)/a^c$), 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 $t, t + 1, t + 2, \dots, t + d - 1$ 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 a spike along it, then that particular spike is 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 $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i , the spikes reach immediately all neurons σ_j such that $(i, j) \in \text{syn}$ and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; spikes sent to a closed neurons are “lost”.

The forgetting rules are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i .

If a rule $E/a^c \rightarrow a^p; d$ of type (1) has $E = a^c$, then we write it in the simplified form $a^c \rightarrow a^p; d$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. Since two firing rules, $E_1/a^{c_1} \rightarrow a^{p_1}; d_1$ and $E_2/a^{c_2} \rightarrow a^{p_2}; d_2$, 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 in that case, only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are used in the sequential manner in each neuron, at most one in each step, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the *total* number of spikes contained in the neuron.

The initial configuration of the system is described by the numbers n_1, n_2, \dots, n_m , of spikes present in each neuron, with all neurons being open. 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, $\langle r_1/t_1, \dots, r_m/t_m \rangle$ is the configuration where neuron σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, $i = 1, 2, \dots, m$; with this notation, the initial configuration is $C_0 = \langle n_1/0, \dots, n_m/0 \rangle$.

A computation in a system as above starts in the initial configuration. In order to compute a function $f : \mathbf{N}^k \rightarrow \mathbf{N}$, we introduce k natural numbers n_1, \dots, n_k in the system by “reading” from the environment a binary sequence $z = 10^{n_1-1}10^{n_2-1}1 \dots 10^{n_k-1}1$. This means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z and no spike otherwise. Note that we input exactly $k + 1$ spikes, i.e., after the last spike we assume that no further spike is coming to the input neuron. The result of the computation is also encoded in the distance between two spikes: we impose the restriction that the system outputs exactly two spikes and halts (sometimes after the second spike), hence it produces a train spike of the form $0^{b_1}10^{r-1}1b^{b_2}$, for some $b_1, b_2 \geq 0$ and with $r = f(n_1, \dots, n_k)$ (the system outputs no spike a non-specified number of steps from the beginning of the computation until the first spike).

The previous definition covers many types of systems/behaviors. If the neuron σ_{in} is not specified, then we have a generative system: we start from the initial configuration and we collect all results of computations, which can be the distance between the first two spikes (as in [18]), the distance between

all consecutive spikes, between alternate spikes, etc. (as in [25]), or it can be spike train itself, either taking only finite computations, hence generating finite strings (as in [5], [9], etc.), or also non-halting computations (as in [26]). Similarly, we can ignore the output neuron and use an SN P system in the accepting mode: a number introduced in the system as the distance between two spikes entering the input neuron is accepted if and only if the computation halts. In the same way we can accept input binary strings or strings over arbitrary alphabets. In the second case, a symbol b_i is taken from the environment by introducing i spikes in the input neuron.

4 Two Examples

Not all types of SN P systems will be discussed below, and only two of them are illustrated in this section.

The first example, given in Figure 2, is actually of a more general interest, as it is a part of a larger SN P system which simulates a register machine. The figure presents the module which simulates a SUB instruction; moreover, it does it without using forgetting rules (the construction is part of the proof that forgetting rules can be avoided – see [15]).

The idea of simulating a register machine $M = (n, H, l_0, l_h, R)$ (n registers, set of labels, initial label, halt label, set of instructions) by an SN P system Π is to associate a neuron σ_r with each register r and a neuron σ_l with each label l from H (there also are other neurons – see the figure), and to represent the fact that register r contains the number k by having $2k$ spikes in neuron σ_r . Initially, all neurons are empty, except neuron σ_{l_0} , which contains one spike. During the computation, the simulation of an instruction $l_i : (\text{OPP}(r), l_j, l_k)$ starts by introducing one spike in the corresponding neuron σ_{l_i} , and this triggers the module associated with this instruction.

For instance, in the case of a subtraction instruction $l_i : (\text{SUB}(r), l_j, l_k)$, the module is initiated when a spike enters the neuron σ_{l_i} . This spike causes neuron σ_{l_i} to immediately send a spike to the neurons $\sigma_{l_{i1}}, \sigma_{l_{i2}}$, and σ_r . If register r is not empty, then the rule $a(aaa)^+ / a^3 \rightarrow a; 0$ will be applied and the spike emitted will cause neurons $\sigma_{l_{i3}}, \sigma_{l_{i5}}$, and finally neuron σ_{l_j} to spike. (In this process, neuron $\sigma_{l_{i4}}$ has two spikes added during one step and it cannot spike.) If register r is empty, hence neuron σ_r contains only the spike received from σ_{l_i} , then the rule $a \rightarrow a; 1$ is applied and the subsequent spikes will cause neurons $\sigma_{l_{i4}}, \sigma_{l_{i6}}$, and finally neuron σ_{l_k} to spike. (In this process, neuron $\sigma_{l_{i3}}$ has two spikes added during one step and does not spike.) After the computation of the entire module is complete, each neuron is left with either zero spikes or an even number of spikes, allowing the module to be run

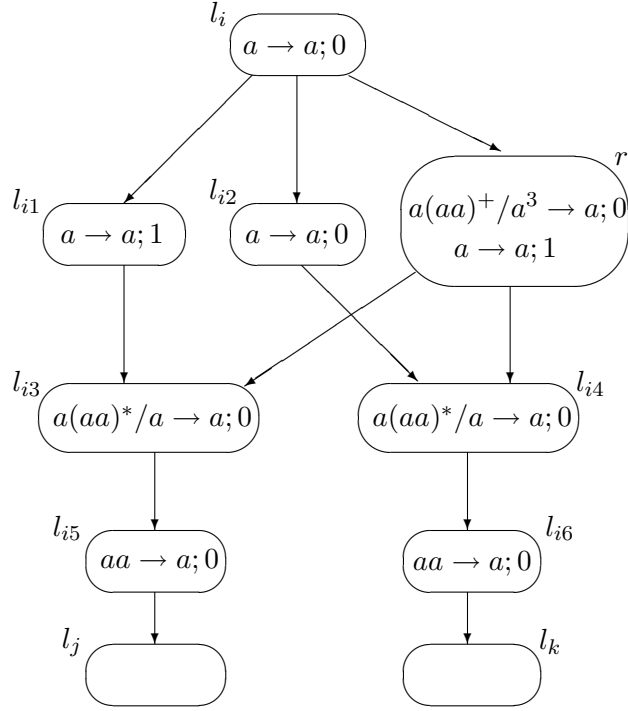


Figure 2: Module SUB (simulating $l_i : (\text{SUB}(r), l_j, l_k)$)

again in a correct way.

The second example deals with an SN P system used as a transducer, and it illustrates the following result from [26]: *Any function $f : \{0, 1\}^k \rightarrow \{0, 1\}$ can be computed by an SN P system with k input neurons (also using further $2^k + 4$ neurons, one being the output one).*

The idea of the proof of this result is suggested in Figure 3, where a system is presented which computes the function $f : \{0, 1\}^3 \rightarrow \{0, 1\}$ defined by

$$f(b_1, b_2, b_3) = 1 \text{ iff } b_1 + b_2 + b_3 \neq 2.$$

The three input neurons, $\sigma_{in_1}, \sigma_{in_2}, \sigma_{in_3}$, are continuously fed with bits b_1, b_2, b_3 , and the output neuron will provide, with a delay of 3 steps, the value of $f(b_1, b_2, b_3)$.

5 Some Results

There are several parameters describing the complexity of an SN P system: number of neurons, number of rules, number of spikes consumed or forgotten by a rule, etc. Here we consider only some of them and we denote by

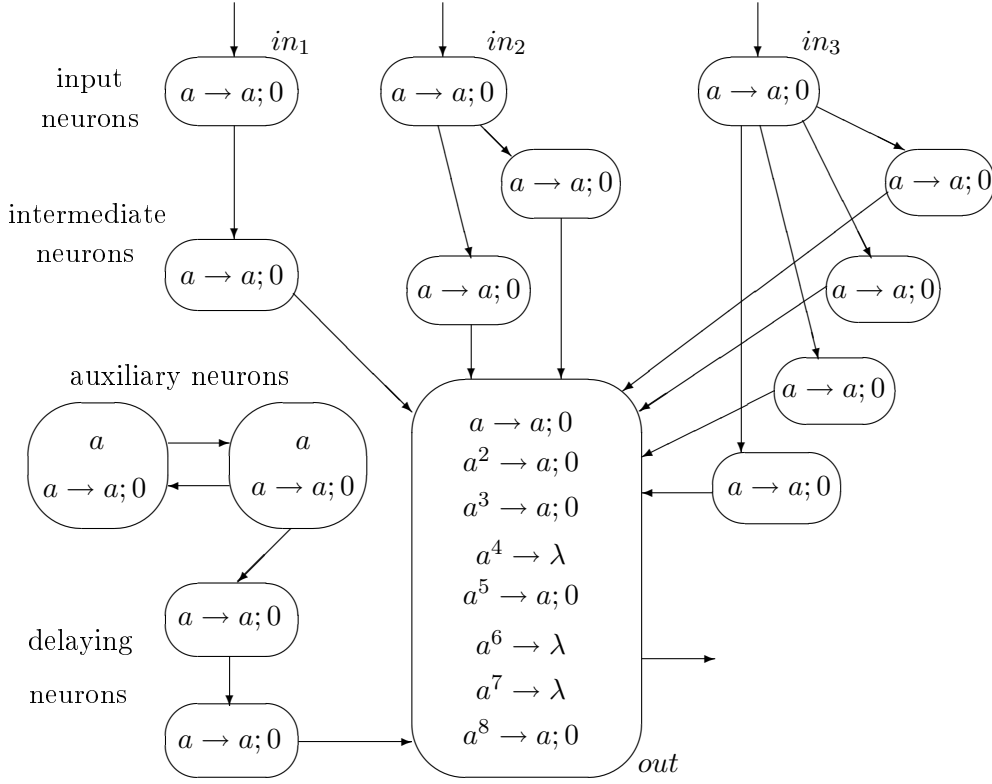


Figure 3: Computing a Boolean function of three variables

$N_2SNP_m(rule_k, cons_p, forg_q)$ the family of all sets $N_2(\Pi)$ computed as specified in Section 3 by SN P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^r \rightarrow a; t$ having $r \leq p$, and all forgetting rules $a^s \rightarrow \lambda$ having $s \leq q$. When any of the parameters m, k, p, q is not bounded, it is replaced with $*$. When we work only with SN P systems whose neurons contain at most s spikes at any step of a computation (*finite* systems), then we add the parameter $bound_s$ after $forg_q$. (Corresponding families are defined for other definitions of the result of a computation, as well as for the accepting case, but the results are quite similar, hence we do not give details here.)

By $NFIN, NREG, NRE$ we denote the families of finite, semilinear, and Turing computable sets of (positive) natural numbers (number 0 is ignored); they correspond to the length sets of finite, regular, and recursively enumerable languages, whose families are denoted by FIN, REG, RE . We also invoke below the family of recursive languages, REC (those languages with a decidable membership).

The following results were proved in [18] and extended in [25] to other ways of defining the result of a computation.

Theorem 5.1. (i) $NFIN = N_2SNP_1(rule_*, cons_1, forg_0) = N_2SNP_2(rule_*, cons_*, forg_*)$.

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

(iii) $NSLIN = N_2SNP_*(rule_k, cons_p, forg_q, bound_s)$, for all $k \geq 3, q \geq 3, p \geq 3$, and $s \geq 3$.

Point (ii) was proved in [18] also for the accepting case, and then the systems used can be required to be deterministic (at most one rule can be applied in each neuron in each step of the computation). In turn, universality results were proved in [19] and [4] also for the exhaustive and for the non-synchronized modes of using the rules, respectively, but only for extended rules. The universality of standard systems remains *open* for these cases.

Let us now pass to mentioning some results about languages generated by SN P systems, starting with the restricted case of binary strings, [5]. We denote by $L(\Pi)$ the set of strings over the alphabet $B = \{0, 1\}$ describing the spike trains associated with halting computations in Π ; then, we denote by $LSNP_m(rule_k, cons_p, forg_q)$ the family of languages $L(\Pi)$, generated by SN P systems Π with the complexity bounded by the parameters m, k, p, q as specified above. When using only systems with at most s spikes in their neurons (finite), we write $LSNP_m(rule_k, cons_p, forg_q, bound_s)$ for the corresponding family. As usual, a parameter m, k, p, q, s is replaced with $*$ if it is not bounded.

Theorem 5.2. (i) *There are finite languages (for instance, $\{0^k, 10^j\}$, for any $k \geq 1, j \geq 0$) which cannot be generated by any SN P system, but for any $L \in FIN, L \subseteq B^+$, we have $L\{1\} \in LSNP_1(rule_*, cons_*, forg_0, bound_*)$, and if $L = \{x_1, x_2, \dots, x_n\}$, then we also have $\{0^{i+3}x_i \mid 1 \leq i \leq n\} \in LSNP_*(rule_*, cons_1, forg_0, bound_*)$.*

(ii) *The family of languages generated by finite SN P systems is strictly included in the family of regular languages over the binary alphabet, but for any regular language $L \subseteq V^*$ there is a finite SN P system Π and a morphism $h : V^* \rightarrow B^*$ such that $L = h^{-1}(L(\Pi))$.*

(iii) *$LSNP_*(rule_*, cons_*, forg_*) \subset REC$, but for every alphabet $V = \{a_1, a_2, \dots, a_k\}$ there are two symbols b, c not in V , a morphism $h_1 : (V \cup \{b, c\})^* \rightarrow B^*$, and a projection $h_2 : (V \cup \{b, c\})^* \rightarrow V^*$ such that for each language $L \subseteq V^*, L \in RE$, there is an SN P system Π such that $L = h_2(h_1^{-1}(L(\Pi)))$.*

These results show that the language generating power of SN P systems is rather eccentric; on the one hand, finite languages (like $\{0, 1\}$) cannot

be generated, on the other hand, we can represent any RE language as the direct morphic image of an inverse morphic image of a language generated in this way. This eccentricity is due mainly to the restricted way of generating strings, with one symbol added in each computation step. This restriction does not appear in the case of extended spiking rules. In this case, a language can be generated by associating the symbol b_i with a step when the output neuron sends out i spikes, with an important decision to take in the case $i = 0$: we can either consider b_0 as a separate symbol, or we can assume that emitting 0 spikes means inserting λ in the generated string. Thus, we both obtain strings over arbitrary alphabets, not only over the binary one, and, in the case where we ignore the steps when no spike is emitted, a considerable freedom is obtained in the way the computation proceeds. This latter variant (with λ associated with steps when no spike exits the system) is considered below.

We denote by $LSN^e P_m(rule_k, cons_p, prod_q)$ the family of languages $L(\Pi)$, generated by SN P systems Π using extended rules, with the parameters m, k, p, q as above.

The next counterparts of the results from Theorem 5.2 were proved in [9].

Theorem 5.3. (i) $FIN = LSN^e P_1(rule_*, cons_*, prod_*)$ and this result is sharp, as $LSN^e P_2(rule_2, cons_2, prod_2)$ contains infinite languages.

(ii) $LSN^e P_2(rule_*, cons_*, prod_*) \subseteq REG \subset LSN^e P_3(rule_*, cons_*, prod_*)$; the second inclusion is proper, because $LSN^e P_3(rule_3, cons_4, prod_2) - REG \neq \emptyset$; actually, $LSN^e P_3(rule_3, cons_6, prod_4)$ contains non-semilinear languages.

(iii) $RE = LSN^e P_*(rule_*, cons_*, prod_*)$.

It is an *open problem* to find characterizations or representations in this setup for families of languages in the Chomsky hierarchy different from FIN, REG, RE . We close this section by mentioning the results from [22]:

Theorem 5.4. *There is a universal computing SN P system with (i) standard rules and 84 neurons and with (ii) extended rules and 49 neurons, and there is a universal SN P system used as a generator of sets of numbers with (iii) standard rules and 76 neurons and with (iv) extended rules and 50 neurons.*

These values can probably be improved (but the feeling is that this improvement cannot be too large).

Tool-kits for handling strings or infinite sequences, on the binary or on the arbitrary alphabet, are provided in [26] and [10]. For instance, in this latter paper one gives constructions of SN P systems for computing the union and concatenation of two languages generated by SN P systems, the intersection with a regular language, while the former paper shows how length

preserving morphisms (codings) can be computed; the problem remains *open* for arbitrary morphisms, Kleene $*$, inverse morphisms.

An interesting result is reported in [6]: SAT can be decided in constant time by using an arbitrarily large pre-computed SN P system, of a very regular shape (as synapse graph) and with empty neurons, after plugging the instance of size (n, m) (n variables and m clauses) of the problem into the system, by introducing a polynomial number of spikes in (polynomially many) specified neurons. This way of solving a problem, by making use of a pre-computed resource given for free, on the one hand, resembles the supposed fact that only part of the brain neurons are active (involved in “computations”) at each time, on the other hand, is not very common in computability and request further research efforts (what kind of pre-computed resource is allowed, so that no “cheating” is possible? how the given resources should be activated? define and study complexity classes for this framework).

6 Plenty of Research Topics

Many problems were already mentioned above, many others can be found in the papers listed below, and further problems are given in [24]. We recall only some general ideas: bring more ingredients from neural computing, especially related to learning/training/efficiency; incorporate other facts from neurobiology, such as the role played by astrocytes, the way the axon not only transmit impulses, but also amplifies them; consider not only “positive” spikes, but also inhibitory impulses; define a notion of *memory* in this framework, which can be read without being destroyed; provide ways for generating an exponential working space (by splitting neurons? by enlarging the number of synapses?), in such a way to trade space for time and provide polynomial solutions to computationally hard problems; define systems with a dynamical synaptic structure; compare the SN P systems as generator/acceptor/transducers of infinite sequences with other devices handling such sequences; investigate further systems with exhaustive and other parallel ways of using the rules, as well as systems working in a non-synchronized way; find classes of (accepting) SN P systems for which there is a difference between deterministic and non-deterministic systems; find classes which characterize levels of computability different from those corresponding to finite automata (semilinear sets of numbers or regular languages) or to Turing machines (recursively enumerable sets of numbers or languages).

We close with a more technical idea: use more general types of rules, for instance, of the form $E/a^n \rightarrow a^{f(n)}; d$, 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 $c = \max\{n \in \mathbf{N} \mid n \leq k, \text{ and } f(n) \text{ is defined}\}$; 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 mathematically powerful.

References

- [1] A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended variants of spiking neural P systems generating strings and vectors of non-negative integers. In [14], 123–134.
- [2] A. Alhazov, R. Freund, A. Riscos-Nunez: One and two polarizations, membrane creation and objects complexity in P systems. *Proc. SYNASC 05*, Timișoara, IEEE Press, 2005, 385–394
- [3] M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-related outputs of computations in P systems. *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 107–122.
- [4] M. Cavaliere, O.H. Ibarra, M. Ionescu, Gh. Păun: Unsynchronized spiking neural P systems. In preparation, 2006.
- [5] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [13], Vol. I, 169–194.
- [6] H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. In [13], Vol. I, 195–206, and *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
- [7] H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. In [13], Vol. I, 207–224, and *Proc. DCFS2006*, Las Cruces, NM, June 2006.
- [8] H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. In [13], Vol. I, 225–240.
- [9] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [13], Vol. I, 241–265.
- [10] H. Chen, T.-O. Ishdorj, Gh. Păun, 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.
- [11] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*, Springer, Berlin, 2006.
- [12] W. Gerstner, W. Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.

- [13] M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.
- [14] H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing, International Workshop, WMC7, Leiden, The Netherlands, 2006, Selected and Invited Papers*, LNCS 4361, Springer, Berlin, 2007.
- [15] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [13], Vol. II, 105–136, and *Theoretical Computer Sci.*, to appear.
- [16] O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. In [14], 424–442.
- [17] O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. In *Proceedings of Fifth Unconventional Computation Conference, UC2006*, York, UK, September 2006.
- [18] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
- [19] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. Submitted, 2006.
- [20] I. Korec: Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.
- [21] W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, 1999.
- [22] A. Păun, Gh. Păun: Small universal spiking neural P systems. In [13], Vol. II, 213–234, and *BioSystems*, in press.
- [23] Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
- [24] Gh. Păun: Twenty six research topics about spiking neural P systems. Available at [27], 2006.
- [25] 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.
- [26] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted 2005.
- [27] The P Systems Web Page: <http://psystems.disco.unimib.it>.
- [28] The Sheffield P Systems Applications Web Page: http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P_Systems_applications.htm