Spiking Neural P Systems: A Short Introduction and New Normal Forms

Linqiang Pan[1,2], Gheorghe Păun[2,3], Mario J. Pérez-Jiménez[2]

[1] Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, lqpan@us.es
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro, gpaun@us.es

Summary. Spiking neural P systems are a class of P systems inspired from the way the neurons communicate with each other by means of electrical impulses (called "spikes"). In the few years since this model was introduced, many results related to the computing power and efficiency of these computing devices were reported. The present paper quickly surveys the basic ideas of this research area and the basic results, then, as typical proofs about the universality of spiking neural P systems, we present some new normal forms for them. Specifically, we consider a natural restriction in the architecture of a spiking neural P system, to have neurons of a small number of types (i.e., using a small number of sets of rules). We prove that three types of neurons are sufficient in order to generate each recursively enumerable set of numbers as the distance between the first two spikes emitted by the system; the problem remains open for accepting SN P systems. The paper ends with the complete bibliography of this domain, at the level of April 2009.

1 Introduction

Spiking neural P systems (SN P systems, for short) were introduced in [32] in the aim of defining computing models based on ideas specific to spiking neurons, currently much investigated in neural computing (see, e.g., [21], [42], [43]). The resulting models are a variant of tissue-like and neural-like P systems from membrane computing – we refer to [56] for basic information in membrane computing, [71] for a comprehensive presentation, and to the web site [81] for the up-to-date information.

In short, an 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 *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 $E/a^c \to a; d$, 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 $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten, 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. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop.

The result of a computation is encoded in the distance between consecutive spikes sent into the environment by the (output neuron of the) system. In [32] only the distance between the first two spikes of a spike train was considered, then in [66] 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.

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.

Both in the generating and the accepting case, SN P systems were proved to be computationally complete (equivalent with Turing machines; we also say that SN P systems are "universal": the equivalence with Turing machines is constructive, hence starting the proof of equivalence from universal Turing machines, or from equivalent universal devices, directly lead to universal SN P systems).

Recently, SN P systems were also used in order to devise (theoretical) ways to solve computationally hard problems in a feasible (polynomial) time. This is usually achieved in membrane computing by means of tools which allow producing an exponential working space in a linear time; the standard way to do it is membrane division. In the SN P area, a different strategy was first explored: with inspiration from the fact that the brain consists of a huge number of neurons out of which only a small part are used, in [9] one address computationally hard problems by assuming that an arbitrarily large SN P system is given "for free", pre-computed, with a structure as regular as possible, and without spikes inside; solving a problem starts by

introducing spikes in certain neurons (in a polynomially bounded number of neurons a polynomially bounded number of spikes are introduced); then, by moving spikes along synapses, the system self-activates, and a specific output provides the answer to the problem. This was illustrated in [9] for `SAT`. Then, both ways to produce this "pre-computed" resource during the computation were considered ([74]), and rules for dividing neurons ([53]), with inspiration from the observation that there are so called neural stem cells, which divide repeatedly, producing new neurons ([18]). The research in this last direction is just started, and further progresses are expected.

The present survey is a brief one, many further directions of research were explored (asynchronous SN P systems, inhibitory spikes and/or synapses, using the rules in a parallel way, Hebbian learning for SN P systems, modeling certain neurophysiological processes, and so on); the bibliography which closes the paper can be a good source of information for the reader.

In order to have a sort of a case study of research in this area, we end the paper with some results (obtained during the Seventh Brainstorming Week on Membrane Computing, Sevilla, February 2-6, 2009), concerning a new normal form for SN P systems. Details will be given in Section 6.

2 Prerequisites

We assume the reader to have some familiarity with (basic elements of) language and automata theory, e.g., from [73], as well as with basics of membrane computing, e.g., from [56], [71], and [81], and we introduce here only a few notations, as well as the notion of register machines, used in the proofs from Section 6 (based on paper [51]).

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. For a regular expression E we denote by $L(E)$ the regular language identified by E.

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 (the languages with a decidable membership).

A *register machine* (see, e.g., [47]) 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:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (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),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents.

We can also use a register machine in the accepting mode: a number is stored in the first register (all other registers are empty); if the computation starting in this configuration eventually halts, then the number is accepted. Again, all sets of numbers in NRE can be obtained, even using deterministic register machines, i.e., with the ADD instructions of the form $l_i : (\texttt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$).

Convention: when evaluating or comparing the power of two number generating/accepting devices, number zero is ignored.

3 Spiking Neural P Systems

We introduce here the SN P systems in the standard form (with non-extended rules):

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

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

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m, \text{ where:}$$

   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) $E/a^c \to a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
   (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 *firing* (we also say *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 $E/a^c \to a; d$ 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 a spike 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 spike is emitted immediately, if $d = 1$, then the spike is 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 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 $t + d + 1$).

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

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, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, 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, but neurons function in parallel with each other.

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 a computation, the system is described both by the numbers of spikes present in each neuron and by the state of each neuron, in the open-closed sense. Specifically, if a neuron is closed, we have to specify the number of steps until it will become again open, i.e., the configuration is written in the form $\langle p_1/q_1, \ldots, p_m/q_m \rangle$; the neuron σ_i contains $p_i \geq 0$ spikes and will be open after $q_i \geq 0$ steps ($q_i = 0$ means that the neuron is already open).

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*, the binary sequence with occurrences of 1 indicating time instances when the output neuron sends a spike out of the system (we also say that the system itself spikes at that time).

In [32], with any spike train containing at least two spikes, the first two being emitted at steps t_1, t_2, one associates a result, in the form of the number $t_2 - t_1$; we say that this number is computed by Π. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript indicates that we only consider the distance between the first two spikes of any computation; note that 0 cannot be computed, that is why we disregard this number when estimating the computing power of any device).

This idea was extended in [66] to several other sets of numbers which can be associated with a spike train: taking into account the intervals between the first k spikes, $k \geq 2$ (direct generalization of the previous idea), or between all intervals; only halting computations can be considered or arbitrary computations; an important difference is between the case when all intervals are considered and the case when the intervals are taken into account alternately (take the first interval, ignore the next one, take the third, and so on); the halting condition can be combined with the alternating style of defining the output.

The result of a computation can be defined also as usual in membrane computing, as the number of spikes present in the output neuron in the end of a computation – we have then to work with halting computations. It is also possible to consider SN P systems working in the accepting mode: we start the computation from an initial configuration, and we introduce in the input neuron two spikes, in steps t_1 and t_2; the number $t_2 - t_1$ is accepted by the system if the computation eventually halts.

Then, the spike train itself can be considered as the result of a computation. The halting computations will thus provide finite strings over the binary alphabet, the non-halting computations will produce infinite sequences of bits. If also an input neuron is provided, then a transducer is obtained, translating input binary strings into binary strings.

4 Example

We illustrate the previous definition with only one example; further constructions of SN P systems will be examined in Section 6. The system is given in a graphical form in Figure 1, following the standard way to pictorially represent a configuration of an SN P system, in particular, the initial configuration. Specifically, each neuron is represented by a "membrane" (a circle or an oval), marked with a label and having inside both the current number of spikes (written explicitly, in the form a^n for n spikes present in a neuron) and the evolution rules; the synapses linking the neurons are represented by arrows;

$L \in FIN$, $L \subseteq B^+$, *we have* $L\{1\} \in LSNP_1(rule_*, cons_*, forg_0, bound_*)$, *and if* $L = \{x_1, x_2, \ldots, x_n\}$, *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* $h : V^* \longrightarrow B^*$ *such that* $L = h^{-1}(L(\Pi))$.

(iii) $LSNP_*(rule_*, cons_*, forg_*) \subset REC$, *but for every alphabet* $V = \{a_1, a_2, \ldots, a_k\}$ *there are two symbols* b, c *not in* V, *a morphism* $h_1 : (V \cup \{b, c\})^* \longrightarrow B^*$, *and a projection* $h_2 : (V \cup \{b, c\})^* \longrightarrow V^*$ *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 (with a delay) in each computation step. This restriction does not appear in the case of extended spiking rules, of the form $E/a^c \to a^p; d$: this time, $p \geq 1$ spikes can be produced when consuming c spikes of the neuron (we assume that $c \geq p$). 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 $LSN^eP_m(rule_k, cons_p, prod_q)$ the family of languages $L(\Pi)$, generated by SN P systems Π using extended rules, with at most m neurons, each neuron having at most k rules, each rule consuming at most p spikes and producing at most q spikes. Again, the parameters m, k, p, q are replaced by $*$ if they are not bounded.

The next counterparts of the results from Theorem 2 were proved in [10].

Theorem 3. (i) $FIN = LSN^eP_1(rule_*, cons_*, prod_*)$ *and this result is sharp in the sense that* $LSN^eP_2(rule_2, cons_2, prod_2)$ *contains infinite languages.*

(ii) $LSN^eP_2(rule_*, cons_*, prod_*) \subseteq REG \subset LSN^eP_3(rule_*, cons_*, prod_*)$; *the second inclusion is proper, because* $LSN^eP_3(rule_3, cons_4, prod_2)$ *contains non-regular languages; actually, the family* $LSN^eP_3(rule_3, cons_6, prod_4)$ *contains non-semilinear languages.*

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

6 New Normal Forms

A neuron σ_i (in the initial configuration of an SN P system) is characterized by n_i, the number of spikes present in it, and by R_i, its associated set of rules. An SN P system is said to be in the kR-normal form, for some $k \geq 1$, if there are at most k different sets $R_1, \ldots, R_k$ of rules used in the m neurons of the system. An SN P system is said to be in the knR-normal form, for some $k \geq 1$,

if there are at most k different pairs $(n_1, R_1), \ldots, (n_k, R_k)$ describing the m neurons of the system.

We denote by $N_\alpha SNP_*(k\beta)$ the families of all sets $N_\alpha(\Pi)$ computed by SN P systems in the $k\beta$-normal form, for $\alpha \in \{2, gen, acc\}$, $\beta \in \{R, nR\}$, and $k \geq 1$, without forgetting rules, but with the spiking rules using the delay feature (we do not restrict the number of neurons, that is why SNP has the subscript $*$).

6.1 A 3*R*-Normal Form Result

We are going now to prove one the new normal form result mentioned in the Introduction: SN P systems with only three different sets of rules are universal when generating numbers encoded in the first two spikes of the spike train.

Theorem 4. $NRE = N_2SNP_*(3R)$.

Proof. We show that $NRE \subseteq N_2SNP_*(3R)$; the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). Let us consider a register machine $M = (m, H, l_0, l_h, I)$ with the properties specified in Section 2. We construct an SN P system Π which simulates M in the way somewhat standard already when proving that a class of SN P systems is universal. Specifically, we construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN which provides the result (in the form of a suitable spike train). Each register r of M will have a neuron r in Π, and if the register contains the number n, then the associated neuron will contain $2n$ spikes.

The modules will be given in a graphical form, indicating their initial configuration, the synapses, and for each neuron, the associated set of rules; all neurons are initially empty, with the exception of the neuron associated with the initial label l_0 of M, which contains one spike, and with exception of a few other neurons, as shown in the following figures.

Fig. 3. Module ADD simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

We consider the following three sets of rules:

$$R_1 = \{a(aa)^+/a \to a; 0, \quad a \to a; 0\},$$

$$R_2 = \{a(aa)^*/a^3 \to a; 0, \quad a \to a; 1\},$$

$$R_3 = \{a(aaa)^*/a \to a; 0, \quad a \to a; 1\}.$$

The ADD module used to simulate an addition instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is indicated in Figure 3. No rule in R_1 can be applied in the presence of an even number of spikes. If a spike enters the neuron (with the label) l_i, then this neuron starts using its rules; initially, this is the case with neuron l_i. Neuron l_i spikes and one spike is sent to both neuron l_{i1} and neuron s_{i2}, which also spike in the next step. In this way, two spikes are sent to neuron r, and this represents the increment of register r by one. Neuron l_{i2} also sends a spike to neurons l_{i3}, l_{i4}, and l_{i5}. Neurons l_{i3} and l_{i5} spike immediately, while neuron l_{i4} can non-deterministically choose either rule to use as both of them are enabled by the existence of a single spike – this ensures the non-deterministic passage to one of the instructions l_j or l_k.

Assume that $\sigma_{l_{i4}}$ uses the rule $a(aa)^+/a \to a; 0$. This means that in the next step $\sigma_{l_{i8}}$ receives two spikes, hence no rule there can be used. Simultaneously, neurons l_{i6} and l_{i7} receive one spike each, and both of them spike. In this way, $\sigma_{l_{i9}}$ receives one spike and $\sigma_{l_{i7}}$ continues having one spike. Neuron l_{i9} contains now a number of spikes of the form $3n + 3$, for some $n \geq 0$ (initially we had two spikes here, hence $n = 0$) and no rule is enabled. In the next step, this neuron receives one further spike, and the first rule is fired (the number of spikes is now $3(n + 1) + 1$). All neurons l_j and l_{i11}, l_{i12} receive one spike. The last two neurons send back to $\sigma_{l_{i9}}$ one spike each, hence the number of spikes in this neuron will be again congruent with 2 modulo 3, as at the beginning. Thus, the neuron associated with the label l_j has been activated.

If neuron l_{i4} uses the rule $a \to a; 1$, then $\sigma_{l_{i7}}$ receives two spikes at the same (after one time unit) time and this branch remains idle, while neurons $l_{i8}, l_{i10}, l_{i13}, l_{i14}$ behave like neurons $l_{i7}, l_{i9}, l_{i11}, l_{i12}$, and eventually σ_{l_k} is activated and the number of spikes from $\sigma_{l_{i10}}$ returns to the form $3s + 2$, for some $s \geq 0$.

The simulation of the ADD instruction is correctly completed.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of the last two authors was supported by Project TIN2006-13452 of the Ministerio de Educación y Ciencia of Spain and by Project of Excellence with *Investigador de Reconocida Valia*, from Junta de Andalucia, grant P08 – TIC 04200.

Note: The bibliography which follows includes several papers on SN P systems published first in the proceedings volumes of the Brainstorming Week on Membrane Computing, with indications of the form "BWMC2007" with the obvious meaning, then specifying the pages. The four brainstorming volumes cited in this way are:

1. M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.: *Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30-February 3, 2006, vol. I and II, Fenix Editora, Sevilla, 2006.
2. M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez, eds.: *Fifth Brainstorming Week on Membrane Computing*, Sevilla, January 29-February 2, 2007, Fenix Editora, Sevilla, 2007.
3. D. Diaz-Pernil, C. Graciani, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez, eds.: *Sixth Brainstorming Week on Membrane Computing*, Sevilla, February 4-8, 2008, Fenix Editora, Sevilla, 2008.
4. R Gutérrez-Escudero, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, eds.: *Seventh Brainstorming Week on Membrane Computing*, Sevilla, February 2-6, 2009, Fenix Editora, Sevilla, 2009.

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. *WMC7, 2006*, 88–101, and *Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers*, LNCS 4361, Springer, 2006, 123–134.
2. A. Binder, R. Freund, M. Oswald: Extended spiking neural P systems with astrocytes - variants for modelling the brain. *Proc. 13th Intern. Symp. AL and Robotics*, AROB2008, Beppu, Japan, 520–524.
3. A. Binder, R. Freund, M. Oswald, L. Vock: Extended spiking neural P systems with excitatory and inhibitory astrocytes. Submitted, 2007.
4. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems; decidability and undecidability. *Proc. DNA13*, LNCS 4848, Springer, 2007.
5. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Sci.*, 2009.
6. M. Cavaliere, I. Mura: Experiments on the reliability of stochastic spiking neural P systems. *Natural Computing*, 7, 4 (2008), 453–470.
7. R. Ceterchi, A.I. Tomescu: Spiking neural P systems – a natural model for sorting networks. *BWMC2008*, 93–106.
8. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *BWMC2006*, vol. I, 169–194, and *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.
9. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *BWMC2006*, vol. I, 195–206, and *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
10. 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*, 7, 2 (2008), 147–166.
11. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. *BWMC2006*, vol. I, 207–224, and *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.
12. H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. *BWMC2006*, vol. I, 225–240, and *Pre-proceedings BIC-TA*, Wuhan, 2006, 60-70, and *Progress in Natural Science*, 17, 4 (2007), 418–423.
13. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. *BWMC2006*, vol. I, 241–266.
14. 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.
15. R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking. *ACMC/FCT 2007 Workshop*, Budapest, *Intern. J. Found. Computer Sci.*, 19 (2008), 1223–1234.
16. R. Freund, M. Oswald: Spiking neural P systems with inhibitory axons. *AROB Conf.*, Japan, 2007.
17. R. Freund, M. Oswald: Regular ω-languages defined by extended spiking neural P systems. *Fundamenta Informaticae*, 83, 1-2 (2008), 65–73.
18. R. Galli, A. Gritti, L. Bonfanti, A.L. Vescovi: Neural stem cells: an overview. *Circulation Research*, 92 (2003), 598–608.

19. M. Garcia-Arnau, D. Pérez, A. Rodriguez-Patón, P. Sosik: On the power of elementary operations in spiking neural P systems. Submitted, 2008.
20. M. Garcia-Arnau, A. Rodriguez-Patón, D. Pérez, P. Sosik: Spiking neural P systems: Stronger normal forms. *BWMC2007*, 157–178.
21. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
22. M.A. Gutiérrez-Naranjo, A. Leporati: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. *BWMC2008*, 193–210.
23. M.A. Gutiérrez-Naranjo, A. Leporati: Performing arithmetic operations with spiking neural P systems. *BWMC2009*.
24. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez. A first model for Hebbian learning with spiking neural P systems. *BWMC2008*, 211–234.
25. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *BWMC2006*, vol. II, 105–136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
26. O.H. Ibarra, A. Păun, A. Rodriguez-Patón: Sequentiality induced by spike numbers in SN P systems. *Proc. 14th Intern. Meeting on DNA Computing*, Prague, June 2008, 36–46.
27. O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. and *Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers*, LNCS 4361, Springer, 2006, 424–442.
28. O.H. Ibarra, S. Woodworth: Spiking neural P systems: some characterizations. *Proc. FCT 2007*, Budapest, LNCS 4639, 23–37.
29. O.H. Ibarra, S. Woodworth: Characterizing regular languages by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1247–1256.
30. O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. *Proc. UC2006*, LNCS 4135, Springer, 2006, 113–129.
31. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. *Proc. DNA12* (C. Mao, Y. Yokomori, B.-T. Zhang, eds.), Seul, June 2006, 32–42, and *DNA Computing. 12th Intern. Meeting on DNA Computing, DNA12, Seoul, Korea, June 2006, Revised Selected Papers* (C. Mao, T. Yokomori, eds.), LNCS 4287, Springer, 2007, 1–16.
32. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
33. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with an exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
34. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 383–394, and *Computing and Informatics*, 27 (2008), 515–528.
35. M. Ionescu, C.I. Tîrnăucă, C. Tîrnăucă: Dreams and spiking neural P systems. *Romanian J. Inform. Sci. and Technology*, 12 (2009), in press.
36. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing*, to appear.
37. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang: Deterministic solutions to `QSAT` and `Q3SAT` by spiking neural P systems with pre-computed resources. *BWMC2009*.

38. I. Korec: Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.
39. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to SAT and Subset-Sum by spiking neural P systems. Submitted, 2007.
40. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *BWMC2007*, 227–246.
41. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problems with spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 405–424.
42. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
43. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
44. V.P. Metta, K. Krithivasan: Spiking neural P systems and Petri nets. Submitted, 2008.
45. J.M. Mingo: Una approximacion al control neural del sueno de ondas lentas mediante spiking neural P systems. Submitted, 2008.
46. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. *BWMC2009*.
47. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
48. T. Neary: A small universal spiking neural P system. *Intern. Workshop. Computing with Biomolecules* (E. Csuhaj-Varju et al., eds.), Viena, 2008, 65–74.
49. T. Neary: On the computational complexity of spiking neural P systems. *Unconventional Computation. 7th Intern. Conf. Vienna, 2008* (C.S. Calude at al., eds.), LNCS 5204, 2008, 189–205.
50. A. Obtulowicz: Spiking neural P systems and modularization of complex networks from cortical neural network to social networks. *BWMC2009*.
51. L. Pan, Gh. Păun: New normal forms for spiking neural P systems. *BWMC2009*.
52. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *BWMC2009*.
53. L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with neuron division and budding. *BWMC2009*.
54. L. Pan, J. Wang, H.J. Hoogeboom: Excitatory and inhibitory neural P systems. Submitted, 2007.
55. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.
56. Gh. Păun: *Membrane Computing – An Introduction*. Springer, Berlin, 2002.
57. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. *Proc. 10th DLT Conf.* (invited talk), Santa Barbara, USA, 2006, LNCS 4036, Springer, Berlin, 2006, 20–35.
58. Gh. Păun: Twenty six research topics about spiking neural P systems. *BWMC2007*, 263–280.
59. Gh. Păun: A quick overview of membrane computing with some details about spiking neural P systems. *Frontiers of Computer Science in China*, 1,1 (2007), 37–49.
60. Gh. Păun: Spiking neural P systems. A tutorial. *Bulletin of the EATCS*, 91 (Febr. 2007), 145–159.
61. Gh. Păun: Spiking neural P systems. Power and efficiency. *Bio-Inspired Modeling of Cognitive Tasks, Proc. IWINAC 2007* (J. Mira, J.R. Alvarez, eds.), Mar Menor, 2007, LNCS 4527, 153–169.

62. Gh. Păun: Spiking neural P systems used as acceptors and transducers. *CIAA 2007, 12th Conf.*, Prague, July 2007, LNCS 4783 (J. Holub, J. Zdarek, eds.), Springer, Berlin, 2007, 1–4.
63. Gh. Păun: Spiking neural P systems with astrocyte-like control. *JUCS*, 13, 11 (2007), 1707–1721.
64. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. Recent results, research topics. Submitted, 2007.
65. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. An overview. *Advancing Artificial Intelligence through Biological Process Applications* (A.B. Porto, A. Pazos, W. Buno, eds.), Medical Information Science Reference, Hershey, New York, 2008, 60–73.
66. 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.
67. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Computing morphisms by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1371–1382.
68. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Manuscript, 2005.
69. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
70. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Spiking neural P systems. An early survey. *Intern. J. Found. Computer Sci.*, 18 (2007), 435–456.
71. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Preess, 2009 (in press).
72. D. Ramirez-Martinez, M.A. Gutiérrez-Naranjo: A software tool for dealing with spiking neural P systems. *BWMC2007*, 299–314.
73. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
74. J. Wan, T.-O. Ishdorj: Revisiting the efficiency of spiking neural P systems. *BWMC2009*.
75. X. Zhang, T.-O. Ishdorj, X. Zeng, L. Pan: Solving PSPACE-complete problems by spiking neural P systems with pre-computed resources. Submitted, 2008.
76. X. Zhang, Y. Jiang, L. Pan: Small universal spiking neural P systems with exhaustive use of rules. *Proc. Third Intern. Conf. on Bio-Inspired Computing. Theory and Appl.*, Adelaide, 2008, 117–127.
77. X. Zhang, J. Wang, L. Pan: A note on the generative power of axon P systems. *Intern. J. CCC*, 4, 1 (2009), 92–98.
78. X. Zhang, X. Zeng, L. Pan: On string languages generated by SN P systems with exhaustive use of rules. *Natural Computing*, 7 (2008), 535–549.
79. X. Zhang, X. Zeng, L. Pan: Smaller universal spiking neural P systems. *Fundamenta Informaticae*, 87 (2008), 117–136.
80. X. Zhang, X. Zeng, L. Pan: On string languages generated by asynchronous spiking neural P systems. *Theoretical Computer Science*, DOI: 10.1016/j.tcs.2008.12.055.
81. The P Systems Website: `http://ppage.psystems.eu`.