International Journal of Foundations of Computer Science
© World Scientific Publishing Company

SPIKING NEURAL P SYSTEMS: AN EARLY SURVEY

GHEORGHE PĂUN
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: george.paun@imar.ro, gpaun@us.es

MARIO J. PÉREZ-JIMÉNEZ
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: marper@us.es

ARTO SALOMAA
Turku Center for Computer Science – TUCS
Lemminkäisenkatu 14, 20520 Turku, Finland
asalomaa@utu.fi

Spiking neural P systems were introduced in the end of the year 2005, in the aim of incorporating in membrane computing the idea of working with unique objects ("spikes"), encoding the information in the time elapsed between consecutive spikes sent from a cell/neuron to another cell/neuron. More than one dozen of papers where written in the meantime, clarifying many of the basic properties of these devices, especially related to their computing power.

The present paper quickly surveys the basic ideas and the basic results, presenting a complete to-date bibliography, and also giving a completing result related to the normal forms possible for spiking neural P systems: we prove that the indegree of such systems (the maximal number of incoming synapses of neurons) can be bounded by 2 without losing the computational completeness.

A series of research topics and open problems are formulated.

Keywords: spiking neuron, membrane computing, P system, register machine, semilinear set, normal form, Chomsky hierarchy

2000 Mathematics Subject Classification: 68Q10, 68Q42, 68Q45

1. Introduction; A Quick Overview of the Literature

Spiking neural P systems (SN P systems, for short) were introduced in [13] in the aim of defining computing models based on ideas specific to spiking neurons, currently much investigated in neural computing (see, e.g., [7], [15], [16]). The resulting models are a variant of tissue-like and neural-like P systems from membrane computing (see [19] and the up-to-date information at the web site [24]), with very specific ingredients and way of functioning.

Very shortly, an SN P system consists of a set of *neurons* (cells, consisting of only one membrane) placed in the nodes of a graph and sending signals (*spikes*, denoted in what follows by the symbol a) along *synapses* (edges 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 $a^k \in L(E)$ 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. We will give details in Section 2. There also are *forgetting rules*, of the form $a^s \rightarrow \lambda$, 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. (This is another major difference with respect to usual P systems, where a sub-multiset of the multiset of objects is "rewritten" by each applied 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.

In the spirit of spiking neurons, the result of a computation is encoded in the distance between consecutive spikes sent into the environment by the (output neuron of the) system. (This idea, of taking the distance between two events as the result of a computation, was already considered for symport/antiport and for catalytic P systems in [1].) In [13] only the distance between the first two spikes of a spike train was considered, then in [21] 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.

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

Another attractive possibility is to consider the spike trains themselves as the result of a computation, and then we obtain a (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 [22]. Languages – even on arbitrary alphabets – can be obtained also in other ways: following the path of a designated spike across neurons, as proposed in [4] (this essentially resembles the trace languages investigated for usual P systems, see [19] and [24]), or generalizing the form of rules. Specifically, one uses rules of the form $E/a^c \rightarrow a^p; d$, with the meaning that, provided that the neuron is covered by E, c spikes are consumed and p spikes are produced, and sent to all connected neurons after d steps (such rules are called *extended*). Then, 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 as many symbols as the number of spikes simultaneously produced. This case was investigated in [6].

Other extensions were proposed in [11] and [10], where several output neurons were considered, thus producing vectors of numbers, not only numbers. A detailed typology of systems (and generated sets of vectors) is investigated in the two papers mentioned above, with classes of vectors found in between the semilinear and the recursively enumerable ones.

The proofs of all computational completeness results known up to now in this area are based on simulating register machines. Starting the proofs from small universal register machines, as those produced in [14], one can find small universal SN P systems (working in the generating mode, as sketched above, or in the computing mode, i.e., having both an input and an output neuron and producing a number related to the input number). This idea was explored in [18] and the results are as follows: there are universal computing SN P systems with 84 neurons using standard rules and with only 49 neurons using extended rules. In the generative case, the best results are 79 and 50 neurons, respectively. Of course, these results are probably not optimal, hence it is a research topic to improve them.

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 [9], 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, while the outdegree of the synapse graph can be bounded by 2, and the regular expressions from firing rules can be of very restricted forms.

The dual problem, of the indegree bounding, was formulated as an open problem in [9]. We solve here this problem, proving, like in the case of the outdegree, that again a normal form holds true: systems with indegree two are computationally complete.

In the next section we will introduce the spiking neural P systems, then (Section 3) we will give some examples, also introducing in this way other ways of using them (generating strings, considering traces). Section 4 presents some results, without proofs, illustrating the computing power of these devices. Section 5 gives the indegree normal form mentioned above. Further remarks and, mainly, further research topics are mentioned in Section 6.

References

[1] 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.

[2] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [8], Vol. I, 169–194.

[3] H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. In [8], Vol. I, 195–206.

[4] H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. In [8], Vol. I, 207–224, and *Proc. DCFS2006*, Las Cruces, NM, June 2006.

[5] H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. In [8], Vol. I, 225–240.

[6] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [8], Vol. I, 241–265.

[7] W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.

[8] M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.

[9] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [8], Vol. II, 105–136.

[10] O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. In *Pre-proceedins of Seventh Workshop on Membrane Computing, WMC7*, Leiden, The Netherlands, July 2006.

[11] 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.

[12] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. In *Proceedings of 12th DNA Based Computing Conference, DNA12*, Seul, June 2006.

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

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

[15] W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.

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

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

[18] A. Păun, Gh. Păun: Small universal spiking neural P systems. In [8], Vol. II, 213–234.

[19] Gh. Păun: *Membrane Computing – An Introduction.* Springer, Berlin, 2002.

[20] Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. In *Proceedings of Developments in Language Theory Conference, DLT 2006*, Santa Barbara, CA, June 2006.

[21] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, to appear (also available at [24]).

[22] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted 2005.

[23] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.

[24] The P Systems Web Page: `http://psystems.disco.unimib.it`.