Computationally Complete Spiking Neural P Systems Without Delay: Two Types of Neurons Are Enough

Rudolf Freund[1] and Marian Kogler[1,2]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: {rudi,marian}@emcc.at

[2] Institute of Computer Science, Martin Luther University Halle-Wittenberg
Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
Email: kogler@informatik.uni-halle.de

Abstract. In this article, we consider spiking neural P systems without delay with specific restrictions on the types of neurons. Two neurons are considered to be of the same type if the rules, the number of spikes in the initial configuration, and the number of outgoing synapses are identical. We show that we are able to achieve computational completeness in both the generating and the accepting case with only two types of neurons, where the number of neurons with unbounded rules is constant (and even minimal).

1 Introduction

Spiking neural P systems, introduced by Ionescu et al. in [3], are a special class of P systems (see [7], [8], [12]) inspired by the working of the (human) brain. In spiking neural P systems, the cells represent neurons, which are connected by synapses. Spiking neural P systems have only one symbol, the *spike*, multiple copies of which can be sent ("fired") from neurons via synapses to other neurons or simply "forgotten" (i.e., removed). The exchange of spikes is regulated by firing and forgetting rules in the neurons.

The problem which "ingredients" are needed to achieve computational completeness or universality with spiking neural P systems has been a challenging question since the introduction of this kind of systems. Several answers have been given, for instance showing that many additional conditions such as delays can be left off [2] or that a limited number of rules per neuron suffices [1], [5], or giving universal P systems with a small number of neurons [6], [11].

Recently, Zeng et al. [10] have considered homogeneous spiking neural P systems, i.e., P systems where the rules in every neuron are identical. However, to achieve universality in this model, some extra conditions such as delays are required.

In our paper, we consider spiking neural P systems with only two types of neurons, without using delays. We consider two neurons to be of the same type if the rules, the number of spikes in the initial configuration, and the number of outgoing edges (synapses) are identical. We show that we are able to achieve computational completeness both in the accepting and in the generating case. Moreover, we are able to give a constant (even minimal) bound on the required number of neurons with unbounded rules.

The rest of the paper is organized as follows: in the next section, we consider some basic formalisms from formal language theory and give the definition of spiking neural P systems. In Section 3 we present our results regarding computational completeness. Finally, we conclude the paper with a short summary of the results obtained in this paper and some interesting open questions for future research.

2 Definitions

The reader is assumed to be familiar with basic notions of formal language theory (see, for instance, [9]).

In the following, by NRE we denoe the family of recursively enumerable sets of natural numbers. For a regular expression E, the corresponding regular language is denoted by $L(E)$.

A nondeterministic register machine is a construct $M = (n, B, p_0, p_h, I)$ where

1. n, $n \geq 1$, is the number of registers,
2. B is the set of instruction labels,
3. p_0 is the start label,
4. p_h is the halting label (only used for the HALT instruction), and
5. I is a set of (labeled) instructions, where every $i \in I$ is of one of the following forms:
   - $p_i : (\texttt{ADD}(r), p_j, p_k)$ increments the value in register r and continues with one of the instructions labeled by p_j and p_k, chosen in a nondeterministic way,
   - $p_i : (\texttt{SUB}(r), p_j, p_k)$ tries to decrement the value in register r; if the register was non-empty before the instruction, the value in the register is decremented and the computation continues with the instruction labeled by p_j, if not, the value in the register is not changed and the computation continues with the instruction p_k;
   - $p_h : \texttt{HALT}$ halts the machine.

As the only nondeterminism occurs in the ADD-instructions, we can easily construct deterministic register machines by imposing the condition $p_j = p_k$. In this case, we write $p_i : (\texttt{ADD}(r), p_j)$. We will be using nondeterministic register machines as generators and deterministic register machines as acceptors.

A deterministic register machine accepts a natural number by starting with the number as input in the first register, with all other registers being empty. Starting from the instruction labeled with p_0, the instructions are applied and the contents of the registers is changed; if and when the machine reaches p_h

and therefore halts, the number is accepted. Register machines can accept all recursively enumerable sets of natural numbers with three registers (see, for instance, [4]).

In the generating case, the (now nondeterministic) register machine starts with empty registers at the initial instruction p_0. When the machine halts, the contents of the first register forms the result. Every recursively enumerable set of natural numbers can be generated with only three registers, where the first register is never decremented [4].

2.1 Spiking Neural P Systems

A spiking neural P system (without delays) is a construct

$$\Pi = (O, \rho_1, ..., \rho_n, syn, in, out)$$

where

1. $O = \{a\}$ is the (unary) set of objects (the object a is called spike),
2. $\rho_1, ..., \rho_n$ are the neurons, where $\rho_i = (d_i, R_i)$ for $1 \leq i \leq n$, with d_i being the number of spikes initially present in the neuron i and R_i being the set of rules, where the rules have one of the following forms:
   - $E/a^i \to a^j$, where E is a regular expression over O and $i, j \geq 1$ (firing rules) or
   - $a^i \to \lambda$, where $i \geq 1$ (forgetting rules).

   There must not be any rule $a^i \to \lambda$ such that $a^i \in L(E)$ for some E of a firing rule.
3. $syn \subseteq \{1, ..., n\} \times \{1, ..., n\}$ are the synapses, where $(i, j) \in syn$ indicates a synapse from i to j,
4. in is the input neuron (with the only function to spike once in generating spiking neural P systems in order to start a computation), and
5. out is the output neuron (no function in accepting spiking neural P systems).

A computation of a spiking neural P system starts from the initial configuration $(d_1, ..., d_n)$ (d_i represents the number of spikes in neuron i, $1 \leq i \leq n$) and then proceeds by making computation steps until the system halts, i.e., when in no neuron a rule can be applied any more. With the system being in the configuration $(c_1, ..., c_n)$, where c_i represents the number of spikes in neuron i, $1 \leq i \leq n$, a computation step is carried out as follows: in every neuron, one rule – if possible – is chosen in a nondeterministic way and applied. A firing rule $E/a^i \to a^j \in R_k$ can be applied in neuron k if, for $c_k = a^m$, $c_k \in L(E)$ and $m \geq i$, removing i spikes from k and adding j spikes in every neuron l where $(k, l) \in syn$. A forgetting rule $a^i \to \lambda$ can be applied in neuron k if and only if $c_k = a^i$; such a rule removes all spikes from the neuron.

Per step and neuron, only one rule may be applied. This is in contrast to other variants of P systems, which usually utilize the maximally parallel mode (where as many rules as possible are applied in parallel). As spiking and forgetting rules are mutually exclusive, the only way a nondeterministic computation step may happen is by the existance of two rules where the languages generated by their regular expressions have a non-empty intersection and the number of fired spikes is not equal.

A spiking neural P system inputs and outputs numbers via a spike train. A spike train starts with a spike given in step t_1 and ends with a spike given in step t_2. The number is specified by $t_2 - t_1 - 1$, i.e., the number of steps that elapse between the two spikes. It accepts an input by a series of configurations, starting from the initial configuration and ending in a halting configuration.

Rules of the form $E/a^i \to a^j$ where $L(E)$ is finite (infinite) are called *bounded* (*unbounded*) rules.

Two neurons ρ_i and ρ_j are of the same type if and only if $R_i = R_j$, $d_i = d_j$ and $|\{(i,k) \in sym \mid k \in \{1,...,n\}\}| = |\{(j,k) \in sym \mid k \in \{1,...,n\}\}|$.

The families of sets of natural numbers generated/accepted by spiking neural P systems with at most k types of neurons and at most m neurons containing unbounded rules are denoted by

$$NSN_{gen}P_* \left(types_k, unbounded_m\right) \quad \text{and} \quad NSN_{acc}P_* \left(types_k, unbounded_m\right),$$

respectively.

3 Results

Theorem 1. *Accepting spiking neural P systems without delays with only two types of neurons and only three neurons containing unbounded rules are computationally complete, i.e.,*

$$NSN_{acc}P_*(types_2, unbounded_3) = NRE.$$

Proof. We prove computational completeness by simulating deterministic register machines with three registers, where the first one is the input register. We use two types of neurons, where neurons of type 1 are responsible for most of the computations, and neurons of type 2 are equivalent to registers (if the register r contains the value i, the neuron r contains $2i$ spikes). Therefore, only three neurons of type 2 (containing unbounded rules) are required, whereas we have an unbounded number of neurons of type 1 (only containing bounded rules). Both types have exactly two outgoing synapses and λ (i.e., no spike) in the initial configuration. The two types of neurons and their ingredients are shown in Figure 1.

In the following, we will denote cells of type 1 by squares and cells of type 2 by dashed squares without giving the rules or the initial configuration again.

As all neurons have an out-degree of two and the generation of additional spikes is only possible with extra steps, it is sometimes necessary to utilize dummy neurons. A dummy neuron structure takes in one spike and forgets it. In the following, we will not explicitly give dummy neurons; rather, if a neuron has an out-degree of less than two, we implicitly assume that the out-degree can be extended with dummy neurons.

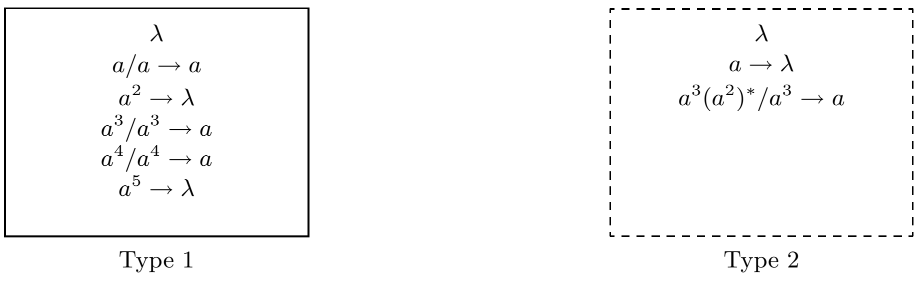

Fig. 1. The two types of neurons used in the proof of Theorem 1

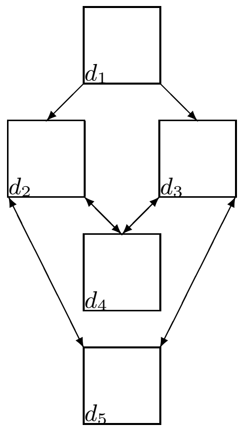

Fig. 2. A dummy neuron structure

When a spike enters the neuron d_1, it triggers the rule $a/a \to a$, which causes the neuron to spike into the neurons d_2 and d_3, which both fire into the neurons d_4 and d_5, thus giving two spikes in both of these neurons, which are forgotten by the rule $a^2 \to \lambda$. As the neurons d_4 and d_5 never spike, the synapses from the neurons d_4 and d_5 back to the neurons d_2 and d_3 have no effect.

The simulation of an ADD instruction $p_i : (\texttt{ADD}(r), p_j)$ is accomplished by the neuron structure as shown in Figure 3:

When a spike enters the neuron p_{i_1}, the simulation of the instruction is started. The neuron fires and sends a spike into p_{i_2} and p_{i_3}, which fire two spikes into r (thereby increasing the value in register r by 1) and one spike into the neuron structure for p_j (to start the simulation of the instruction p_j). Therefore, the simulation of an ADD-instruction requires three neurons and two steps.

For SUB-instructions, we do not consider the instructions individually; rather, we construct a structure responsible for all SUB(r)-instructions. In the following,

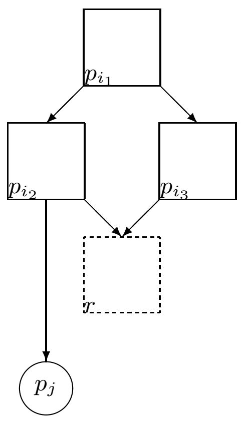

Fig. 3. A neuron structure simulating an `ADD`-instruction

we will denote the `SUB`-instructions for some register r by $p_i : (SUB(r), p_{i_s}, p_{i_f})$ for $1 \leq i \leq n$, where the p_i are the instructions, the p_{i_s} are the follow-up instructions in case of success (i.e., when the register is non-empty) and the p_{i_f} the follow-up instructions in case of failure (i.e., when the register is empty). Figure 4 illustrates the neuron structure needed for the simulation of `SUB`(r)-instructions; p_i denotes the instruction currently being executed and n the number of `SUB`(r)-instructions.

In this diagram, only one possible instruction is described in detail. The spike from register r has to be forwarded to all possible p'_{j_1} (for this, $\frac{\lceil\sqrt{n}\rceil(\lceil\sqrt{n}\rceil-1)}{2}$ neurons are needed) and the spike for the instruction needs to be delayed until this spike arrives (using $2(\lceil\sqrt{n}\rceil - 1)$ cells).

When the simulation of the instruction starts, one spike is sent into the neuron representing register r. If the register is empty (i.e. the neuron contains no spike), the spike is forgotten by the rule $a \to \lambda$, if not, the rule $a^3(a^2)^*/a \to a$ removes two spikes (thus decrementing the value in the register) and fires one spike. However, as the register does not know which instruction caused the subtraction, this spike is forwarded to all possible instructions. In parallel, one spike is fired by the instruction neuron, to be duplicated and forwarded to the neuron of the instruction. To make sure that the spikes from the register and the spikes from the instruction reach the register at the same time, the two spikes from the instruction neuron (after duplication) are delayed by additional neurons of type 1.

Finally, the neurons p'_{i_f} and p'_{i_s} act as "gatekeepers" of the respective instructions.

The neuron p'_{i_s} corresponding to the current instruction receives two spikes. If the subtraction succeeded (i.e., if the register was non-empty), every neuron p'_{i_s} receives two spikes. If only two spikes are present (i.e., if the neuron belongs to a different instruction or the operation was not successful), they are forgotten by the rule $a^2 \to \lambda$; otherwise, four spikes are in the neuron, which causes it

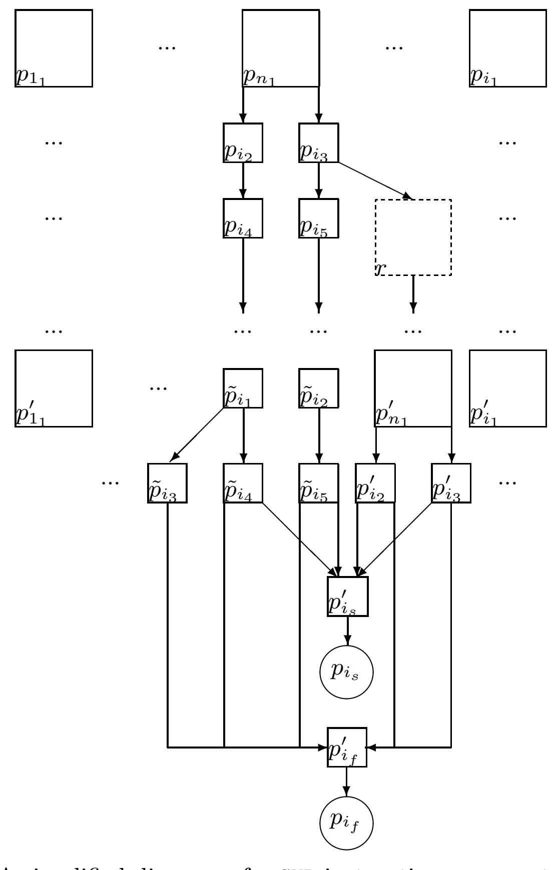

Fig. 4. A simplified diagram of a SUB-instruction neuron structure

to fire by the rule $a^4/a^4 \rightarrow a$ and therefore to start the simulation of the next instruction p_{i_s}.

Moreover, the neuron p'_{i_f} receives three spikes from the current instruction, which causes it to fire if the subtraction did not succeed and therefore to start the simulation of the next instruction p_{i_f}; if the subtraction succeeded, two additional spikes are sent from the register, triggering the forgetting rule $a^5 \rightarrow \lambda$. If only these two spikes representing a successful subtraction are sent in, the forgetting rule $a^2 \rightarrow \lambda$ can be applied and removes them.

The simulation of a SUB-instruction takes $5 + (\lceil\sqrt{n}\rceil - 1)$ steps and $15 + (\lceil\sqrt{n}\rceil - 1)$ neurons (not counting dummy structures). Additionally, $\frac{\lceil\sqrt{n}\rceil(\lceil\sqrt{n}\rceil-1)}{2}$ neurons for every register are needed, where n is the number of SUB-instructions that affect this register.

The HALT-instruction is represented by a dummy structure, which consumes the spike, thus ending the computation of the P system.

Finally, we consider the input to be given by two spikes: one in the first step and one in the $n + 2$th step, where n is the input number. The neuron structure as depicted in Figure 5 puts $2n$ spikes into the input register r_i and one spike into the first neuron of the neuron structure simulating the initial instruction p_0.

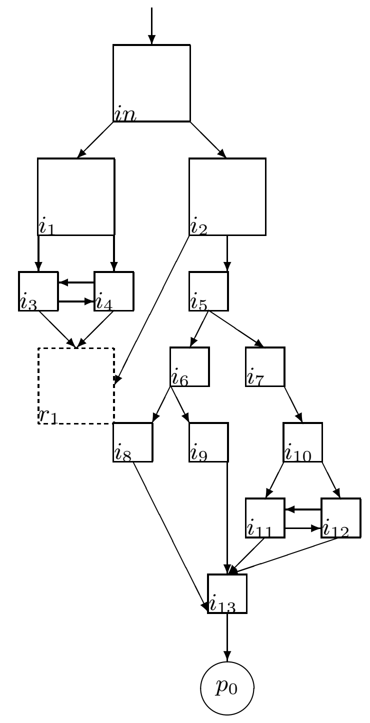

Fig. 5. The neuron structure initializing the computation

When the first spike enters the system, thereby starting the spike train, it initializes the two loops and puts one spike into the register (which is forgotten because of the rule $a \to \lambda$). Per step, the first loop (neurons i_3 and i_4) puts two spikes into the register, thereby increasing the count by 1. The second loop (neurons i_{11} and i_{12}) puts two spikes into an intermediate neuron i_{13} (where they are forgotten immediately).

After the second spike has entered, the loops stop (as the loop cells now contain two spikes, and the forgetting rules $a^2 \to \lambda$ can be applied), another single spike is fired into the register (as the input is the number of steps between the two spikes, but the loop runs one more time, it is necessary to decrement the register once) and four spikes enter i_{14}, thus causing it to fire a spike into the structure simulating the first instruction p_0.

We observe that on any number as input, the spiking neural P system halts if and only if the corresponding register machine halts. This observation concludes the proof. $\square$

Theorem 2. *Generating spiking neural P systems without delays with only two types of neurons and only three neurons containing unbounded rules are computationally complete, i.e.,*

$$NSN_{gen}P_*(types_2, unbounded_3) = NRE.$$

Proof. Our construction is similar to the proof given for Theorem 1. However, we need to simulate nondeterministic register machines here, so we need to modify the simulation of ADD-instructions. To be able to simulate nondeterminism, we introduce two additional rules to the neuron type 1 (see Figure 6), which allows for the nondeterministic simulation of an instruction $p_i : (\mathtt{ADD}(r), p_j, p_k)$ (see Figure 7).

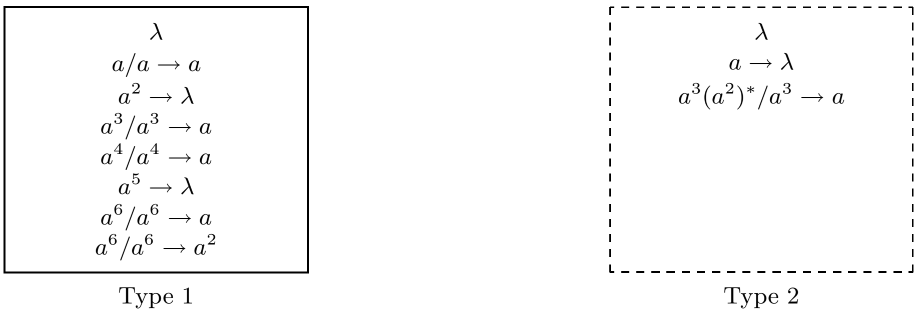

Fig. 6. The two types of neurons used in the proof of Theorem 2

After the instruction has been executed, the neuron $p_{i_{18}}$ contains six spikes, which nondeterministically triggers one of the rules $a^6/a^6 \to a$ and $a^6/a^6 \to a^2$. If the first rule is executed, the instruction p_j immediately follows; otherwise, the instruction p_k is executed. (This construction is vaguely similar to the one used for SUB-instructions; however, in this case, both input spikes come from the same source.) This simulation requires 22 neurons per ADD-instruction and is executed in seven steps.

The simulation of SUB-instructions is exactly as in the proof given in the proof of the previous theorem. However, we do not need to consider SUB-instructions for the output register 1, as we can assume that this first register is never decremented.

To start a computation, the input neuron has to receive a spike from the environment, which then is immediately sent to the first neuron of the neuron structure simulating the initial instruction p_0.

When the machine halts with the HALT-instruction, we still have to output the result as a spike train. As the first register is never subtracted from during the simulation of the register machine, we can construct the output structure as shown in Figure 8:

The basic mechanism of the output structure is as follows: the system spikes if and only if there was no spike from the current subtraction from the register

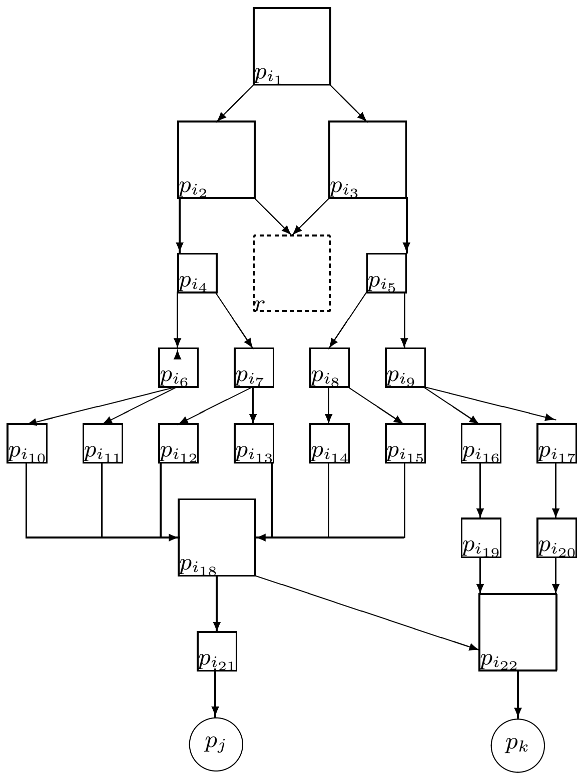

Fig. 7. A neuron structure simulating an `ADD`-instruction

or from the last subtraction from the register. This is the case at the beginning (where there is no preceding subtraction) and at the end (when the last subtraction cannot be executed, as the register is empty). The number of steps between these two spikes is always the number in the register minus one (or, if the register was empty, there would be only one spike). Therefore, we need to increment the register (by adding two spikes) before starting the output.

We observe that the output of the spiking neural P system (given by the spike train) is equivalent to the output of the corresponding register machine (given by the value in the first register upon halting). Finally, we remark that only three neurons (representing the registers) are of type 2 (and only neurons of this type contain unbounded rules). This observation concludes the proof. □

4 Conclusions

By using the characterization of types (where two neurons are of the same type if the rules, the initial configuration, and the number of outgoing synapses are identical), we were able to show that for obtaining computational completeness only two types of neurons are needed, where only the second type of neurons

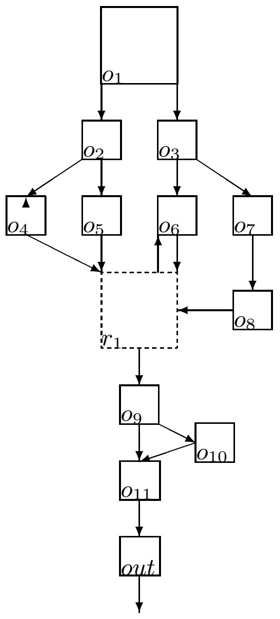

Fig. 8. The output structure

contains unbounded rules. We also showed that three neurons of the second type are enough, which already is the minimal number needed anyway as registers can only be represented by neurons with unbounded rules; however, the number of neurons of the first type could not be bounded.

Interesting questions for further research remain, such as:

- Which ingredients are needed to generalize the results for recursively enumerable sets of vectors of natural numbers?
- Which changes in the constructions of the proofs elaborated in this paper are necessary if the input (in the accepting case) or the output (in the generating case) are initially (finally) given as contents of an unbounded input (output) neuron?
- To which number of neurons with unbounded rules can the constructions in the proofs elaborated in this paper be reduced when simulating universal register machines (using spike trains with three spikes to introduce the code of the machine to be simulated as well as its input)?
- For all these variants of spiking neural P systems without delay, is one type of (unbounded) neurons sufficient?

References

1. M. García-Arnau, D. Peréz, A. Rodríguez-Patón, and P. Sosík. Spiking neural P systems: Stronger normal forms. In M. A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, and A. Riscos-Núñez, editors, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 33–62, 2007.
2. O. H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, and S. Woodworth. Normal forms for spiking neural P systems. *Theor. Comput. Sci.*, 372(2-3):196–217, 2007.
3. M. Ionescu, Gh. Păun, and T. Yokomori. Spiking neural P systems. *Fundam. Inf.*, 71(2,3):279–308, 2006.
4. M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
5. L. Pan and Gh. Păun. Spiking neural P systems: An improved normal form. *Theoretical Computer Science*, 411(6):906 – 918, 2010.
6. A. Păun and Gh. Păun. Small universal spiking neural P systems. *Biosystems*, 90(1):48 – 60, 2007.
7. Gh. Păun. Computing with membranes. *J. of Computer and System Sci.*, 61(1):108–143, 2000.
8. Gh. Păun. *Membrane Computing: An Introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
9. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages, vol. 1: Word, Language, Grammar.* Springer-Verlag, Secaucus, NJ, USA, 1997.
10. X. Zeng, X. Zhang, and L. Pan. Homogeneous spiking neural P systems. *Fundam. Inf.*, 97(1-2):275–294, 2009.
11. X. Zhang, X. Zeng, and L. Pan. Smaller universal spiking neural P systems. *Fundam. Inf.*, 87(1):117–136, 2008.
12. The P Systems Web Page: `http://ppage.psystems.eu`.