Performing Arithmetic Operations with Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo[1], Alberto Leporati[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
alberto.leporati@unimib.it

Summary. We consider spiking neural P systems as devices which can be used to perform some basic arithmetic operations, namely addition, subtraction, comparison and multiplication by a fixed factor. The input to these systems are natural numbers expressed in binary form, encoded as appropriate sequences of spikes. A single system accepts as inputs numbers of any size. The present work may be considered as a first step towards the design of a CPU based on the working of spiking neural P systems.

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [5] as a new class of distributed and parallel computing devices. They were inspired by *membrane systems* (also known as *P systems*) [9, 10, 12], in particular by tissue–like P systems [8], and are based on the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons.

In SN P systems the processing elements are called *neurons*, and are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consists of a number of copies of a single object type, namely the *spike*. Neurons may also contain *firing* and/or *forgetting* rules. The *firing rules* allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cell. The application of the rules depends on the contents of the neuron; in the general case, applicability is determined by checking the contents of the neuron against a regular set associated with the rule. As inspired from biology, when a neuron sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period

of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike arrives at the target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, an SN P system of degree $m \geq 1$, as defined in [6], is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out), \text{ where:}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, 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) $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers. If $E = a^c$, then it is usually written in the following simplified form: $a^c \rightarrow a; d$; similarly, if a rule $E/a^c \rightarrow a; d$ has $d = 0$, then we can simply write it as $E/a^c \rightarrow a$. Hence, if a rule $E/a^c \rightarrow a; d$ has $E = a^c$ and $d = 0$, then we can write $a^c \rightarrow a$;
      (2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (where $L(E)$ denotes the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π.

The rules of type (1) are called *firing* (also *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains $k \geq c$ spikes, and $a^k \in L(E)$, then the rule $E/a^c \rightarrow a; d \in R_i$ can be applied. The execution of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$, then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. (Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.) 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*, 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), and cannot fire new rules. In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$) and select rules to be fired.

Rules of type (2) are called *forgetting* rules, and 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. Note that, by definition, if a firing rule is applicable then no forgetting rule is applicable, and vice versa.

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 $E_2 : a^{c_1} \to a; d_2$, 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. In such a case, only one of them is nondeterministically chosen. 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, 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 number of steps to wait until it becomes open (this number is zero if the neuron is already open). A *computation* in a system as above starts in the initial configuration. A positive integer number is given in input to a specified *input neuron*. This number may be encoded in many different ways, for example as the interval of time steps elapsed between the insertion of two spikes into the neuron (note that this is a unary encoding). Other possible encodings are discussed below. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. The computation proceeds in a sequential way into each neuron, and in parallel among different neurons. Generally, a computation may not halt. However, in any case the output of the system is usually considered to be the time elapsed between the arrival of two spikes in a designated *output cell*. Defined in this way, SN P systems compute functions of the kind $f : \mathbb{N} \to \mathbb{N}$; they can also indirectly compute functions of the kind $f : \mathbb{N}^k \to \mathbb{N}$ by using a bijection from $\mathbb{N}^k$ to $\mathbb{N}$.

As discussed in [6], there are other possibilities to encode natural numbers read from and/or emitted to the environment by SN P systems; for example, we can consider the number of spikes contained in the input and in the output neuron, respectively, or the number of spikes read/produced in a given interval of time. Also, an alternative way to compute a function $f : \mathbb{N}^k \to \mathbb{N}$ is to introduce k natural numbers $n_1, n_2, \ldots, n_k$ in the system by "reading" from the environment a binary sequence $z = 0^b 10^{n_1} 10^{n_2} 1 \ldots 10^{n_k} 10^g$, for some $b, g \geq 0$; this means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z. Note that we input exactly $k+1$ spikes, and that this is again a unary encoding. Sometimes we may need to impose that the system outputs exactly two spikes and halts (sometimes after the second spike) hence producing a spike train of the form $0^{b'} 10^r 10^{g'}$, for some $b', g' \geq 0$ and with $r = f(n_1, n_2, \ldots, n_k)$. In what follows we will also consider systems which have k input neurons. For these systems, the input values $n_1, n_2, \ldots, n_k$ will arrive *simultaneously* to the system, each one entering through the corresponding input neuron. Moreover, the input numbers will be sometimes encoded in *binary* form, using the same number of bits in order to synchronize the different parts of the systems: the sequence of bits that encodes a natural number will be represented as a spike train such that, at each time step, the presence of a spike denotes 1 in the corresponding position, whereas the absence of a spike denotes 0. For further details, we refer the reader to the next sections.

If we do not specify an input neuron (hence no input is taken from the environment) then we use SN P systems in the *generative* mode; we start from the initial configuration, and the distance between the first two spikes of the output neuron (or the number of spikes, etc.) is the result of the computation. Note that generative SN P systems are inherently nondeterministic, otherwise they would always reproduce the same sequence of computation steps, and hence the same output. Dually, we can neglect the output neuron and use SN P systems in the *accepting* mode; for $k \geq 1$, the natural number $n_1, n_2, \ldots, n_k$ are read in input and, if the computation halts, then the numbers are accepted.

In [5] it was shown that generative SN P systems are universal, that is, can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P systems with a bounded number of spikes in the neurons. SN P systems are universal also in their computing version (that is, when they compute functions $f : \mathbb{N} \to \mathbb{N}$), as it can be easily shown by simulating register machines [6]. These results can also be obtained with even more restricted forms of spiking P systems; for example, [4] shows that at least one of these features can be avoided while keeping universality: time delay (refractory period) greater than 0, forgetting rules, outdegree of the synapse graph greater than 2, and regular expressions of complex form. These results have been further extended in [3], where it is shown that universality is kept even if we remove some combinations of two of the above features. Finally, in [11] the behavior of spiking neural P systems on infinite strings and the generation of infinite sequences of 0 and 1 was investigated, whereas in [1] spiking neural P systems were studied as language generators (over the binary alphabet $\{0, 1\}$).

Spiking neural P systems have also been used to solve decision problems, both in a *semi–uniform* and in a *uniform* way [7]. When solving a problem $\mathcal{Q}$ in the *semi–uniform* setting, for each specified instance $\mathcal{I}$ of $\mathcal{Q}$ we build in a polynomial time (with respect to the size of $\mathcal{I}$) an SN P system $\Pi_{\mathcal{Q},\mathcal{I}}$, whose structure and initial configuration depend upon $\mathcal{I}$, that halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of $\mathcal{Q}$. On the other hand, a *uniform* solution of $\mathcal{Q}$ consists of a family $\{\Pi_{\mathcal{Q}}(n)\}_{n\in\mathbb{N}}$ of SN P systems such that, when having an instance $\mathcal{I} \in \mathcal{Q}$ of size n, we introduce a polynomial (in n) number of spikes in a designated (set of) input neuron(s) of $\Pi_{\mathcal{Q}}(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. Indeed, in the semi–uniform setting we do not even need any input neuron, as the instance of the problem is embedded into the structure (number of spikes, graph of neurons and synapses, rules) from the very beginning.

In this paper, we consider SN P systems in a completely different way. We will view SN P systems as components of a restricted Arithmetic Logic Unit in which one or more natural numbers are provided in binary form, some arithmetic operation is performed and the result is sent out (to the environment) also in bi-

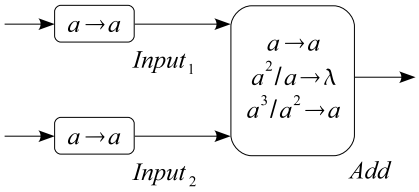

Fig. 1. An SN P system that performs the addition among two natural numbers expressed in binary form

nary form. The arithmetic operations we will consider are addition, subtraction and multiplication among natural numbers. Each number will be provided to the system as a sequence of spikes: at each time step, zero or one spike will be supplied to the input neuron, depending upon whether the corresponding bit of the number is 0 or 1. Also the output neuron will emit the computed number to the environment in binary form, encoded as a spike train.

The paper is organized as follows. In Section 2 we present an SN P system which can be used to add two natural numbers expressed in binary form, of any length (that is, composed of any number of bits). In Section 3 we present an analogous SN P system, that computes the difference (subtraction) among two natural numbers. Section 4 contains the description of a very simple system that can be used to compare two natural numbers. Section 5 first extends the system presented in Section 2 to perform the addition of any given amount of natural numbers, and then describes a spiking neural P system that performs the multiplication of any natural number, given as input, by a fixed factor embedded into the system. Finally, section 6 concludes the paper and suggests some possible directions for future research.

2 Addition

In this section we describe a simple SN P system that performs the addition of two natural numbers. We call such a system the *SN P system for 2-addition*. It is composed of three neurons (see Figure 1): two *input* neurons and an *addition* neuron, which is also the output neuron. Both input neurons have a synapse to the addition neuron. Each input neuron receives one of the numbers to be added as a sequence of spikes, that encodes the number in binary form. As explained above, no spike in the sequence at a given time instant means 0 in the corresponding position of the binary expansion, whereas one spike means 1. Note that the numbers provided as input to the system may be arbitrarily long. The input neurons have only one rule, $a \to a$, which is used to forward the spikes to the addition neuron as soon as they arrive. The addition neuron has three rules: $a \to a$, $a^2/a \to \lambda$ and $a^3/a^2 \to a$, which are used to compute the result.

Formally, the SN P system for 2-addition is defined as a structure:

$$\Pi_{Add} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{Add}, syn, in_1, in_2, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Add} = (0, R_{Add})$, with $R_{Add} = \{a \to a, a^2/a \to \lambda, a^3/a^2 \to a\}$;
- $syn = \{(Input_1, Add), (Input_2, Add)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Add$.

The following theorem holds.

Theorem 1. *The SN P system for 2-addition outputs the addition in binary form of two non-negative integers, provided to the neurons σ_{Input_1} and σ_{Input_2} in binary form.*

Proof. Let t denote the current time step. In the initial configuration ($t = 0$), the system does not contain any spike. At $t = 1$, a binary digit has been provided to each of the input neurons, in the form of absence or presence of a spike. Such a digit is associated with the power 2^0 in the binary representation of the input numbers. At $t = 2$ these spikes are placed in neuron σ_{Add}. We can now divide the future behavior of σ_{Add} in three cases, depending upon the number of spikes it contains, that may be 0, 1 or 2.

- If there are no spikes, no rules are activated and in the next step 0 spikes are sent to the environment. This encodes the operation $0 + 0 = 0$.
- If there is 1 spike, then the rule $a \to a$ is triggered. The spike is consumed and one spike is sent out. This encodes $0 + 1 = 1 + 0 = 1$.
- If there are 2 spikes, then the rule $a^2/a \to \lambda$ is triggered. This means that no spike is sent out, which can be interpreted as a 0 in the binary form of the output. Note that only one spike is consumed in the application of the rule. This means that in the next computation step the spikes in the addition neuron will be the spikes provided from the input neuron plus the one which has not been consumed. This encodes $1 + 1 = 10$.

In the general case, we observe that the spikes in the addition neuron either come from the input neurons or remain in the neuron from the previous step. Since only one spike can remain, the number of spikes contained in σ_{Add} can be 0, 1, 2 or 3 at each computation step. The cases for 0, 1, or 2 spikes are treated as described above. If there are 3 spikes, then the rule $a^3/a^2 \to a$ is applied. One spike is sent out, two of them are consumed and one remains for the next step. This encodes the operation $1 + 1 + 1 = 11$. □

As an example, let us consider the addition $28 + 21 = 49$, that in binary form can be written as $11100_2 + 10101_2 = 110001_2$. Table 1 reports the number of spikes contained in each neuron of Π_{Add}, as well as the number of spikes sent to

Time step	$Input_1$	$Input_2$	Add	Output
$t = 0$	0	0	0	0
$t = 1$	**0**	**1**	0	0
$t = 2$	**0**	**0**	1	0
$t = 3$	**1**	**1**	0	**1**
$t = 4$	**1**	**0**	2	**0**
$t = 5$	**1**	**1**	2	**0**
$t = 6$	0	0	3	**0**
$t = 7$	0	0	1	**1**
$t = 8$	0	0	0	**1**

Table 1. Number of spikes in each neuron of Π_{Add}, and number of spikes sent to the environment, at each time step during the computation of the addition $11100_2 + 10101_2 = 110001_2$

the environment, at each time step during the computation. The input and the output sequences are written in bold. Note that the first instant of time for which the output is valid is $t = 3$, due to the time needed for the first input bits to reach the output neuron and to be processed.

3 Subtraction

The *subtraction SN P system*, illustrated in Figure 2, consists of ten neurons. The first input number, the *minuend*, is provided to neuron σ_{Input_1} in binary form, encoded as a spike train as described above. Similarly, the second input number (the *subtrahend*) is supplied in binary form to neuron σ_{Input_2}. Neuron σ_{Input_1} is linked to three auxiliary neurons, called σ_{aux_1}, σ_{aux_2} and σ_{aux_3}, whereas σ_{Input_2} is connected with another auxiliary neuron called σ_{aux_4}. The set of neurons σ_{aux_1}, σ_{aux_2} and σ_{aux_3} act as a multiplier of the minuend: they multiply by 3 the number of spikes provided by neuron σ_{Input_1}. The system contains also a neuron called σ_{gen}, which is connected with σ_{aux_flow} and σ_{aux_5}. These latter neurons are also mutually connected by two synapses. The target of the subsystem built by the neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5} is to provide a constant flow of spikes to σ_{Sub}. All the neurons mentioned up to now have only one rule: $a \to a$. The neurons σ_{aux_i}, for $1 \leq i \leq 5$, are connected with neuron σ_{Sub}; this is both the output neuron and the neuron in which the result of the subtraction is computed, by means of six rules: $a \to \lambda$, $a^2/a \to a$, $a^3/a^2 \to \lambda$, $a^4 \to a$, $a^5 \to \lambda$ and $a^6/a^5 \to a$. At the beginning of the computation all neurons are empty except σ_{gen}, which contains one spike.

Formally, the subtraction SN P system is defined as a structure:

$$\Pi_{Sub} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{aux_1}, \sigma_{aux_2}, \sigma_{aux_3}, \sigma_{aux_4}, \sigma_{aux_5}, \sigma_{gen}, \sigma_{aux\_flow}, \sigma_{Sub}, syn, in_1, in_2, out)$$

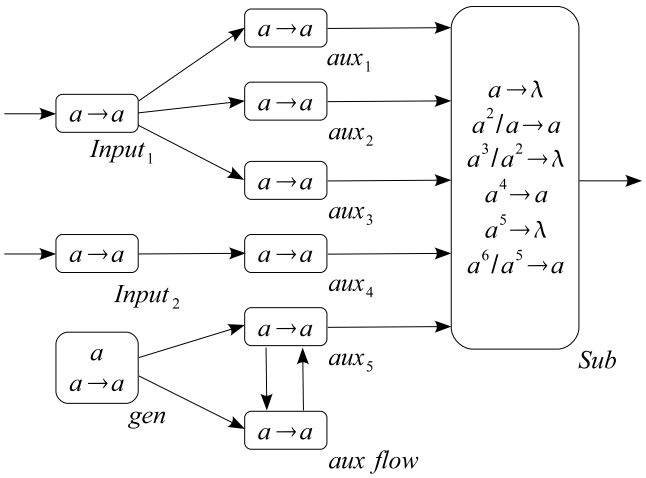

Fig. 2. An SN P system that performs the subtraction among two natural numbers expressed in binary form

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{aux_i} = (0, R_{aux_i})$, with $R_{aux_i} = \{a \to a\}$, for all $1 \leq i \leq 5$;
- $\sigma_{aux\_flow} = (0, R_{aux\_flow})$, with $R_{aux\_flow} = \{a \to a\}$;
- $\sigma_{gen} = (1, R_{gen})$, with $R_{gen} = \{a \to a\}$;
- $\sigma_{Sub} = (0, R_{Sub})$, with $R_{Sub} = \{a \to \lambda, a^2/a \to a, a^3/a^2 \to \lambda, a^4 \to a, a^5 \to \lambda, a^6/a^5 \to a\}$;
- $syn = \{(Input_1, aux_1), (Input_1, aux_2), (Input_1, aux_3), (Input_2, aux_4), (gen, aux_5), (gen, aux\_flow), (aux_5, aux\_flow), (aux\_flow, aux_5), (aux_1, Sub), (aux_2, Sub), (aux_3, Sub), (aux_4, Sub), (aux_5, Sub)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Sub$.

The following theorem holds.

Theorem 2. *The subtraction SN P system outputs the subtraction, in binary form, of two non-negative integer numbers, provided in binary form to neurons σ_{Input_1} (the minuend) and σ_{Input_2} (the subtrahend).*

Proof. At the beginning of the computation (time $t = 0$) all the neurons of the system are empty but neuron σ_{gen}, that contains one spike. Let us focus first on the subsystem composed of the neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5}. At time $t = 1$, one spike is placed in each of the neurons σ_{aux_flow} and σ_{aux_5}, whereas neuron σ_{gen} contains no spikes. Since this neuron has no incoming synapses, it will not receive

spikes and thus it will be empty along all the computation. At time $t = 2$, σ_{aux_5} has sent one spike to σ_{Sub} and another one to σ_{aux_flow}. Meanwhile, σ_{aux_flow} has sent a spike to σ_{aux_5}. This means that at $t = 2$ the spikes in both neurons σ_{aux_flow} and σ_{aux_5} are the same as those at time $t = 1$, and one spike has been sent to the subtraction neuron. Therefore, starting from $t = 2$, in each time unit neuron σ_{Sub} receives one spike from σ_{aux_5}.

Now let us focus on the input neurons. At the beginning of the computation they are empty. At time $t = 3$, neuron σ_{Sub} can receive 0 or 3 spikes from σ_{Input_1} (through σ_{aux_1}, σ_{aux_2} and σ_{aux_3}), and 0 or 1 spike from σ_{Input_2} (through σ_{aux_4}). This means that, at time $t = 3$, σ_{Sub} may contain 1, 2, 4 or 5 spikes. We can thus consider the following four cases.

- If σ_{Sub} contains 1 spike, then it comes from σ_{aux_5}. The rule $a \to \lambda$ is triggered, so that the spike is consumed and no spike is sent out. This encodes the operation $0 - 0 = 0$.
- If σ_{Sub} contains 2 spikes, then one of them comes from σ_{aux_5} and the other one from σ_{aux_4}. The rule $a^2/a \to a$ is triggered; as a consequence, one spike is sent out and one spike is consumed, so one spike remains in σ_{Sub} for the next step. This encodes $x0 - y1 = z1$ where x, y and z are numbers in binary form such that $x - (y + 1) = z$.
- If σ_{Sub} contains 4 spikes, then one of them comes from σ_{aux_5} and the other three from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. The rule $a^4 \to a$ is triggered, so that all the spikes are consumed and one spike is sent out. This encodes $1 - 0 = 1$.
- If σ_{Sub} contains 5 spikes, then each of them comes from σ_{aux_1} to σ_{aux_5}. The rule $a^5 \to \lambda$ is triggered; as a consequence, all the spikes are consumed and no spike is sent out. This encodes the operation $1 - 1 = 0$.

In the general case, we consider that the spikes in neuron σ_{Sub} come from the input neurons, or remains in the neuron from the previous step. Since only one spike can remain at each computation step, the number of spikes can be 1, 2, 3, 4, 5 or 6.

- If σ_{Sub} contains 1 spike, then it comes from σ_{aux_5}. No spike has remained from the previous step, nor comes from the input neurons. The rule $a \to \lambda$ is applied. As a result, the spike is consumed and no spike is sent out. This encodes $0 - 0 = 0$.
- If σ_{Sub} contains 2 spikes, then one of them comes from σ_{aux_5} and the other one either comes from σ_{aux_4} or remains from the previous step (no both cases may occur at the same time). The case in which the second spike comes from σ_{aux_4} has already been considered above. If it remains from the previous step, then it comes from the operation in the second digit in $x00 - y01 = z11$ where $x - (y + 1) = z$. The rule $a^2/a \to a$ is triggered; as a consequence, one spike is sent out and one spike is consumed, thus leaving one spike in the neuron for the next step.
- If σ_{Sub} contains 3 spikes, then one of them comes from σ_{aux_5}, the other one from σ_{aux_4} and the last one remains from the previous step. This situation

comes from the operation on the second digit of $x00 - y11 = z01$, where $x - (y + 1) = z1$. The rule $a^3/a^2 \to \lambda$ is applied. This means that two spikes are consumed, no spike is sent out and one spike remains in the neuron for the next step.

- If σ_{Sub} contains 4 spikes, then one of them comes from σ_{aux_5} and the other three come from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. This case has been already considered above: the rule $a^4 \to a$ is triggered, which consumes all the spikes and sends out one spike. This encodes $1 - 0 = 1$.
- If σ_{Sub} contains 5 spikes, then one of them comes from σ_{aux_5} and three of them come from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. The fifth spike can come from σ_{aux_4} or can remain from the previous step (the two events cannot occur both at the same time). The case in which it comes from σ_{aux_4} has already been considered above. If it remains from the previous step, then the situation comes from $x10 - y01 = z01$, where $x - y = z1$. The rule $a^5 \to \lambda$ is applied, which consumes all the spikes. No spike is sent out and no spike remains for the next step.
- If σ_{Sub} contains 6 spikes, then one of them remains from the previous step and the others come from σ_{aux_1}, σ_{aux_5}, σ_{aux_2} and σ_{aux_3}. The situation comes from $x_1 0 - y11 = z11$ where $x - (y + 1) = z$. The rule $a^6/a^5 \to a$ is triggered; as a consequence, five spikes are consumed, one spike is sent out and one spike remains for the next step. □

Time step	$Input_1$	$Input_2$	aux_1	aux_2	aux_3	aux_4	aux_5	Sub	Output
$t = 0$	0	0	0	0	0	0	0	0	0
$t = 1$	**0**	**1**	0	0	0	0	1	0	0
$t = 2$	**0**	**1**	0	0	0	1	1	1	0
$t = 3$	**1**	**0**	0	0	0	1	1	2	0
$t = 4$	**1**	**0**	1	1	1	0	1	3	**1**
$t = 5$	**0**	**1**	1	1	1	0	1	5	**0**
$t = 6$	**1**	**1**	0	0	0	1	1	4	**0**
$t = 7$	**1**	0	1	1	1	1	1	2	**1**
$t = 8$	0	0	1	1	1	0	1	6	**1**
$t = 9$	0	0	0	0	0	0	1	5	**1**
$t = 10$	0	0	0	0	0	0	1	1	**0**

Table 2. Number of spikes in each neuron of Π_{Sub}, and number of spikes sent to the environment, at each time step during the computation of the subtraction $1101100_2 - 110011_2 = 111001_2$

As an example let us calculate $108 - 51 = 57$, that in binary form can be written as $1101100_2 - 110011_2 = 111001_2$. Table 2 reports the number of spikes that occur in each neuron of Π_{Sub}, at each time step during the computation. Note that at each step only one rule is active in the subtraction neuron, and thus

the computation is deterministic. At time $t = 0$, the eight neurons of Π_{Sub} are empty and the system has not yet emitted any spike to the environment. At time $t = 1$, the minuend and the subtrahend start to be supplied to σ_{Input_1} and σ_{Input_2}, respectively. From this moment, σ_{aux_5} always contains one spike. At $t = 4$ the first digit of the output is emitted to the environment. The computation continues until the binary sequence 111001_2, which is the binary representation of 57, has been emitted.

4 Checking Equality

Checking the equality of two numbers is a different task with respect to computing addition or subtraction. When comparing two numbers the output should be a binary mark, which indicates whether they are equal or not. Since an SN P system produces a spike train, we will encode the output as follows: starting from an appropriate instant of time, at each computation step the system will emit a spike if and only if the two corresponding input bits (that were inserted into the system some time steps before) are equal. So doing, the system will emit no spike to the environment if the input numbers are equal, and at least one spike if they are different. Stated otherwise, if we compare two n-bit numbers then the output will also be an n-bit number: if such an output number is 0, then the input numbers are equal, otherwise they are different.

Bearing in mind these marks for equality and inequality, the design of the SN P system is trivial. It consists of three neurons: two input neurons, having $a \to a$ as the single rule, linked to a third neuron, the *checking neuron*. This checking neuron is also the output neuron, and it has only two rules: $a^2 \to \lambda$ and $a \to a$.

Formally, the SN P system for checking equality is defined as a structure:

$$\Pi_{Comp} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{Comp}, syn, in_1, in_2, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Comp} = (0, R_{Comp})$, with $R_{Comp} = \{a \to a, a^2 \to \lambda\}$;
- $syn = \{(Input_1, Comp), (Input_2, Comp)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Comp$.

The system is illustrated in Figure 3. Due to its simplicity, we just give an informal justification of its working and correctness.

Both σ_{Input_1} and σ_{Input_2} send the information of the corresponding input digits simultaneously. Such information consist of 0 or 1 spike for each of the two inputs, so at each computation step there can be 0, 1 or 2 spikes in neuron σ_{Comp}. If there are 0 or 2 spikes, then no difference has been found and no spike is sent to the

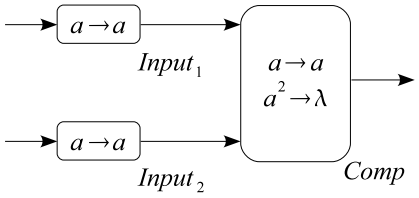

Fig. 3. An SN P system that compares two natural numbers of any length, expressed in binary form

environment. A spike is emitted only when a single spike is placed in σ_{Comp}. As explained above, this will be considered as a mark of inequality of the two binary numbers given as input, independent of any following bits.

5 Multiplication

In this section we present a first approach to the problem of computing the multiplication of two binary numbers by means of SN P systems. The main difference between multiplication and the addition or subtraction operations presented in the previous sections is that in addition and subtraction the n-th digit in the binary representation of the inputs is used exactly once, to compute the n-th digit of the output, and then it can be discarded. On the contrary, in the usual algorithm for multiplication the different digits of the inputs are reused several times; hence the design of a device that executes such algorithm needs some kind of memory. Other algorithms for multiplication, such as Booth's algorithm (see, for example, [2]) also need some kind of memory, to store the intermediate results.

We propose a family of SN P systems for performing the multiplication of two non-negative integer numbers. In these systems only one number, the multiplicand, is provided as input; the other number, the multiplier, is instead encoded in the structure of the system. The family thus contains one SN P system for each possible multiplier.

In the design of our systems, we exploit the following basic fact concerning multiplication by one binary digit: any number remains the same if multiplied by 1, whereas it produces a 0 if multiplied by zero. Bearing this fact in mind, an SN P system associated to a fixed multiplier only needs to add different copies of the multiplicand, by feeding such copies to an addition device with the appropriate delay. Before presenting this design, we extend the 2-addition SN P system from section 2 to an n-addition SN P system.

5.1 Adding n numbers

In this section we present a family $\{\Pi_{Add}(n)\}_{n\geq 2}$ of SN P systems which allows to add numbers expressed in binary form. Precisely, for any integer $n \geq 2$ the

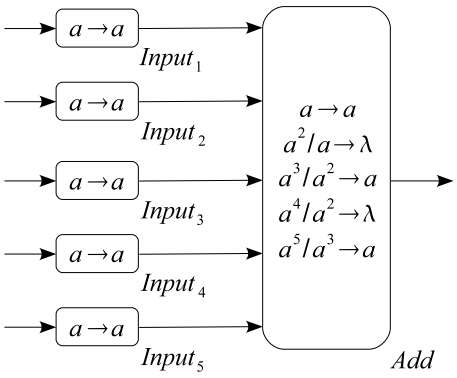

Fig. 4. An SN P system that performs the addition among five natural numbers expressed in binary form

system $\Pi_{Add}(n)$ computes the sum of n natural numbers. In what follows we will call $\Pi_{Add}(n)$ the SN P system for n-addition. For $n = 2$ we will obtain the SN P system for 2-addition that we have described in Section 2.

The system $\Pi_{Add}(n)$ consists of $n+1$ neurons: n *input* neurons and one *addition neuron*, which is also the output neuron. Each input neuron has only one rule, $a \to a$, and is linked to the addition neuron. This latter neuron computes the result of the computation by means of n rules r_i, $i \in \{1, \ldots, n\}$, which are defined as follows:

$$\begin{array}{ll} r_i \equiv a^i/a^{k+1} \to a & \text{if } i \text{ is odd and } i = 2k+1 \\ r_i \equiv a^i/a^k \to \lambda & \text{if } i \text{ is even and } i = 2k \end{array}$$

Formally, the SN P system for n-addition is defined as a structure:

$$\Pi_{Add}(n) = (O, \sigma_{Input_1}, \ldots, \sigma_{Input_n}, \sigma_{Add}, syn, in_1, \ldots, in_n, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_i} = (0, R_{Input_i})$, with $R_{Input_i} = \{a \to a\}$ for all $i \in \{1, 2, \ldots, n\}$;
- $\sigma_{Add} = (0, R_{Add})$, with $R_{Add} = \bigcup_{i=1}^{n} \{r_i\}$, where:
  - $r_i \equiv a^i/a^{k+1} \to a$ if i is odd and $i = 2k+1$;
  - $r_i \equiv a^i/a^k \to \lambda$ if i is even and $i = 2k$;
- $syn = \bigcup_{i=1}^{n} \{(Input_i, Add)\}$;
- $in_i = Input_i$, for all $i \in \{1, 2, \ldots, n\}$;
- $out = Add$.

As an example, Figure 4 shows $\Pi_{Add}(5)$, the SN P system for 5-addition. The following theorem holds.

Theorem 3. *The SN P system for n-addition outputs the addition in binary form of n non-negative integer numbers, provided to the neurons* $\sigma_{Input\_1}, \ldots, \sigma_{Input\_n}$ *in binary form.*

Proof. Let $A_1, \ldots, A_n$ be the n numbers to be added, and let $a_i^p a_i^{p-1} \ldots a_i^0$ be the binary expression of A_i, $1 \leq i \leq n$, padded with zeros on the left to obtain $(p+1)$-digit numbers (where $p+1$ is the maximum number of digits among the binary representations of $A_1, \ldots, A_n$). Hence we can write $A_i = \sum_{k=0}^{p} a_i^k 2^k$ for all $i \in \{1, 2, \ldots, n\}$.

For each $i \in \{1, \ldots, n\}$, let A'_i be the number with binary expression $a_i^p \ldots a_i^1$, i.e., $A'_i = \sum_{k=1}^{p} a_i^k 2^{k-1}$. Moreover, let $U = \sum_{i=1}^{n} a_i^0$ and let $k \in \mathbb{N}$ and $\alpha \in \{0, 1\}$ such that $U = 2k + \alpha$ ($\alpha = 1$ is U is odd and $\alpha = 0$ if U is even). The addition of $A_1, \ldots, A_n$ can be written as:

$$\sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \sum_{k=0}^{p} a_i^k 2^k = \left(\sum_{i=1}^{n} \sum_{k=1}^{p} a_i^k 2^k \right) + \sum_{i=1}^{n} a_i^0$$
$$= 2 \left(\sum_{i=1}^{n} \sum_{k=1}^{p} a_i^k 2^{k-1} \right) + 2k + \alpha = 2 \left(\sum_{i=1}^{n} A'_i + k \right) + \alpha$$

According to this formula, if $b_r \ldots b_0$ is the binary expression of $\sum_{i=1}^{n} A_i$, then $b_0 = \alpha$ and $b_r \ldots b_1$ is the binary expression of $\sum_{i=1}^{n} A'_i + k$.

Let us assume now that at the time instant t there are i spikes in neuron σ_{Add}. These spikes can come from the input neurons, or they may have remained from the previous computation step. Let us compute the t-th digit b_t of the output, dividing the problem in the following two cases.

- Let us assume that i is odd and $i = 2k + 1$. Then, according to the previous formula, $b_t = 1$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i / a^{k+1} \rightarrow a$. By applying this rule, one spike is sent to the environment ($b_t = 1$) and $k+1$ spikes are consumed, so that $i - (k+1) = 2k + 1 - (k+1) = k$ spikes remain for the next step.
- Let us assume that i is even and $i = 2k$. Then, according to the previous formula, $b_t = 0$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i / a^k \rightarrow \lambda$. By applying this rule, no spike is sent to the environment ($b_t = 1$) and k spikes are consumed, so that $i - k = 2k - k = k$ spikes remain for the next step. □

As an example, let us consider the addition of the numbers 3, 4, 2, 7 and 1, whose binary representations are 11_2, 100_2, 10_2, 111_2 and 1_2, respectively. Table 3 shows the evolution of the number of spikes in the neurons of the SN P system $\Pi_{Add}(5)$ (illustrated in Figure 4), as well as the number of spikes sent to the environment at each computation step, when performing such an addition. The input and the output sequences are written in bold. Note that the first instant of time for which the output is valid is $t = 3$. According with the computation, the result of the addition is $17 = 10001_2$.

5.2 Multiplication by a fixed multiplier

We now describe a family $\{\Pi_{Mult}(n)\}_{n \in \mathbb{N}}$ of SN P systems, one for each natural number n, that operate as multiplier devices. Precisely, the system $\Pi_{Mult}(n)$ takes

Time step	$Input_1$	$Input_2$	$Input_3$	$Input_4$	$Input_5$	Add	Output
$t = 1$	**1**	**0**	**0**	**1**	**1**	0	0
$t = 2$	**1**	**0**	**1**	**1**	0	3	0
$t = 3$	0	**1**	0	**1**	0	4	**1**
$t = 4$	0	0	0	0	0	4	**0**
$t = 5$	0	0	0	0	0	2	**0**
$t = 6$	0	0	0	0	0	1	**0**
$t = 7$	0	0	0	0	0	0	**1**

Table 3. Number of spikes in each neuron of $\Pi_{Add}(5)$ (the system illustrated in Figure 4) and number of spikes sent to the environment, at each time step during the computation of the addition $11_2 + 100_2 + 10_2 + 111_2 + 1_2 = 10001_2$

as input a number in binary form, and outputs the input multiplied by n. The output is also expressed in binary form.

Given a natural number n, the SN P system $\Pi_{Mult}(n)$ is described as follows. It consists of one *input* neuron, σ_{Input}, linked to k neurons $\sigma_{aux_{11}}, \ldots, \sigma_{aux_{k1}}$, where k is the number of occurrences of the digit 1 in the binary representation of n. For each $i \in \{1, \ldots, k\}$, neuron $\sigma_{aux_{i1}}$ is connected with a new neuron $\sigma_{aux_{i2}}$, which is connected with $\sigma_{aux_{i3}}$, etc. This sequence of neurons is a path of linked neurons that extends until reaching $\sigma_{aux_{ij_i}}$, where j_i is the number of order of the corresponding digit in the binary representation of n, where the first digit corresponds to 2^0, the second one corresponds to 2^1, and so on. All the last neurons of the k sequences are connected with a final neuron σ_{Add}, which is the same as the output neuron of the k-addition SN P system $\Pi_{Add}(k)$ described above. This neuron has the rules for the addition of k natural numbers. All the other neurons have only the rule $a \to a$.

For example, let us consider $n = 26$, whose binary representation is 11010_2. Such a representation has three digits equal to 1, at the positions 2, 4 and 5. The system $\Pi_{Mult}(26)$, illustrated in Figure 5, has 13 neurons: σ_{Input}, σ_{Add}, and three sequences of neurons associated with the three digits equal to 1:

- $\sigma_{aux_{11}}$ and $\sigma_{aux_{12}}$, corresponding to the 1 in the second position (corresponding to the power 2^1);
- $\sigma_{aux_{21}}$, $\sigma_{aux_{22}}$, $\sigma_{aux_{23}}$ and $\sigma_{aux_{24}}$, corresponding to the 1 in the fourth position (corresponding to the power 2^3);
- $\sigma_{aux_{31}}$, $\sigma_{aux_{32}}$, $\sigma_{aux_{33}}$, $\sigma_{aux_{34}}$ and $\sigma_{aux_{35}}$, corresponding to the 1 in the fifth position (corresponding to the power 2^4).

The last neurons of these sequences, namely $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$ and $\sigma_{aux_{35}}$, are linked to neuron σ_{Add}, which is also the output neuron. The rules of this neuron are $a \to a$, $a^2/a \to \lambda$ and $a^3/a^2 \to a$, which are the same as in the addition neuron of the 3-addition SN P system $\Pi_{Add}(3)$ described in the previous section.

Theorem 4. *The SN P system $\Pi_{Mult}(n)$ built as above takes as input a number m in binary form and outputs the result of the multiplication $m \cdot n$ in binary form.*

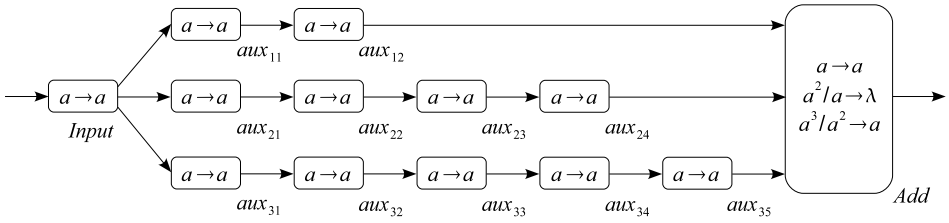

Fig. 5. An SN P system that computes the product among the natural number given as input (in binary form) and the fixed multiplier $26 = 11010_2$, encoded in the structure of the system

Proof. Since we already proved that the neuron σ_{Add} performs the addition of several numbers in binary form, it only remains to transform the multiplication $m \cdot n$ (where n is a fixed parameter) into an appropriate addition. To this aim, let $n = \sum_{j=0}^{q} n_j 2^j$. Then we can write:

$$m \cdot n = m \cdot \left(\sum_{j=0}^{q} n_j 2^j \right) = \sum_{j=0}^{q} \left(m \cdot 2^j \right) n_j$$
$$= \sum \left\{ m \cdot 2^j \mid j \in \{0, \ldots, q\} \wedge n_j = 1 \right\}$$

According to this expression, $m \cdot n$ can be calculated as the addition of as many copies of m as the number of digits n_j equal to 1 that appear in the binary representation of n. Such copies have to be padded with j zeros on the right (that is, they have to be multiplied by 2^j), to take into account the correct weight of n_j. Hence, if $k = \sum_{j=0}^{q} n_j$ then to compute $m \cdot n$ it suffices to provide k copies of m – each shifted in time of a number of steps that corresponds to the weight of a bit n_j equal to 1 – to a neuron that computes the addition of k natural numbers. □

We conclude this section with an example of multiplication. We will take $n = 26$ as the multiplier (hence the system $\Pi_{Mult}(26)$ illustrated in Figure 5) and we will supply the number $29 = 11101_2$ as input. Table 4 reports the number of spikes contained in neurons $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$, $\sigma_{aux_{35}}$ and σ_{Add} of $\Pi_{Mult}(26)$ during the computation, as well as the number of spikes sent to the environment. According to this computation, the output of the multiplication is 1011110010_2, which is the binary representation of 754.

6 Conclusion and Future Work

In this paper we have presented some simple SN P systems that perform the following operations on natural numbers: addition, multiple addition, comparison, and multiplication by a fixed factor. All the numbers given as inputs to these

Time step	*Input*	aux_{12}	aux_{24}	aux_{35}	*Add*	*Out*
$t = 1$	**1**	0	0	0	0	0
$t = 2$	**0**	0	0	0	0	0
$t = 3$	**1**	1	0	0	0	0
$t = 4$	**1**	0	0	0	1	**0**
$t = 5$	**1**	1	1	0	0	**1**
$t = 6$	0	1	0	1	2	**0**
$t = 7$	0	1	1	0	3	**0**
$t = 8$	0	0	1	1	3	**1**
$t = 9$	0	0	1	1	3	**1**
$t = 10$	0	0	0	1	3	**1**
$t = 11$	0	0	0	0	2	**1**
$t = 12$	0	0	0	0	1	**0**
$t = 13$	0	0	0	0	0	**1**

Table 4. Number of spikes in neurons $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$, $\sigma_{aux_{35}}$ and σ_{Add} of $\Pi_{Mult}(26)$ (the system illustrated in Figure 5) and number of spikes sent to the environment, at each time step during the computation of the multiplication $11101_2 \cdot 11010_2 = 1011110010_2$

systems are expressed in binary form, encoded as a spike train in which at each time instant the presence of a spike denotes 1, and the absence of a spike denotes 0. The outputs of the computations are also expelled to the environment in the same form.

The motivation for this work lies in the fact that we would like to implement a CPU using only spiking neural P systems. To this aim, the first step is to design the Arithmetic Logic Unit of the CPU, and hence to study a compact way to perform arithmetical and logical operations by means of spiking neural P systems. Ours is certainly not the unique possible way to approach the problem: other two possibilities, that we leave as two directions for future research, are:

- Simulating by means of SN P systems the widely known Boolean circuits that perform the desired arithmetical and logical operations. However, so doing we need one system for each possible input size: as an example, we need one system to add 2-bit numbers, another one to add 3-bit numbers, and so on. On the contrary, the systems presented in this paper are able to process inputs of any length;
- Simulating by means of SN P systems the programs of register machines that perform the desired arithmetical and logical operations. This solution would overcome the need to have one system for each possible input size. But, on the other hand, the simulating SN P systems would probably be much larger than those presented in this paper.

In any case, an interesting extension to the present work is to try to design an SN P system for the multiplication, where both the numbers m and n to be multiplied are supplied as inputs. And, of course, we would also need a system

to compute the integer division between two natural numbers; probably, this last system is the most difficult to be designed.

Acknowledgement

The first author wishes to acknowledge the support of the project TIN2006–13425 of Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. The second author was partially supported by the MIUR project "Mathematical aspects and emerging applications of automata and formal languages" (2007).

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds., *Fourth Brainstorming Week on Membrane Computing*, Vol. I RGCN Report 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 169–194.
2. M.J. Flynn: *Advanced Computer Arithmetic Design.* John Wiley Publisher, 2001.
3. M. García-Arnau, D. Peréz, A. Rodríguez-Patón, P. Sosík: Spiking neural P systems: stronger normal forms. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez, eds., *Fifth Brainstorming Week on Membrane Computing*, RGCN Report 01/2007, Research Group on Natural Computing, Sevilla University, Fénix Editora, 157–178.
4. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
6. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. In *DNA Computing,* 12^{th} *International Meeting on DNA Computing (DNA12)*, Revised Selected Papers, LNCS 4287, Springer, 2006, 1–16.
7. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical **NP**-complete problems with spiking neural P systems. In *Membrane Computing*, LNCS 4860, Springer, 2007, 336–352.
8. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: A new class of symbolic abstract neural nets: Tissue P systems. In *Proceedings of COCOON 2002*, LNCS 2387, Springer, 2002, 290–299.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
10. Gh. Păun: *Membrane Computing. An Introduction.* Springer–Verlag, 2002.
11. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Foundations of Computer Science*, 17, 4 (2006), 975–1002
12. The P systems website: `http://ppage.psystems.eu/`