International Journal of Neural Systems, Vol. 27, No. 8 (2017) 1750042 (13 pages)

DOI: 10.1142/S0129065717500423

Spiking Neural *P* Systems with Communication on Request

Linqiang Pan
Key Laboratory of Image Information Processing
and Intelligent Control of Education Ministry of China
School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, P. R. China

and Zhengzhou University of Light Industry
Zhengzhou 450002, Henan, P. R. China
lqpan@mail.hust.edu.cn

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
P. O. Box 1-764, RO-014700 Bucharest, Romania
gpaun@us.es

Gexiang Zhang*
Robotics Research Center, Xihua University
Chengdu 610039, P. R. China

Key Laboratory of Fluid and Power Machinery (Xihua University)
Ministry of Education, Chengdu 610039, P. R. China

School of Electrical Engineering
Southwest Jiaotong University, Chengdu 610031, P. R. China
zhgxdylan@126.com

Ferrante Neri
Centre for Computational Intelligence
School of Computer Science and Informatics
De Montfort University, The Gateway
Leicester LE1 9BH, England, UK
fneri@dmu.ac.uk

Accepted 10 August 2017
Published Online 5 October 2017

Spiking Neural *P* Systems are Neural System models characterized by the fact that each neuron mimics a biological cell and the communication between neurons is based on spikes. In the Spiking Neural *P* systems investigated so far, the application of evolution rules depends on the contents of a neuron (checked by means of a regular expression). In these *P* systems, a specified number of spikes are consumed and a specified number of spikes are produced, and then sent to each of the neurons linked by a synapse to the evolving neuron.

In the present work, a novel communication strategy among neurons of Spiking Neural *P* Systems is proposed. In the resulting models, called Spiking Neural *P* Systems with Communication on Request, the spikes are requested from neighboring neurons, depending on the contents of the neuron (still checked by means of a regular expression). Unlike the traditional Spiking Neural *P* systems, no spikes are consumed or created: the spikes are only moved along synapses and replicated (when two or more neurons request the contents of the same neuron).

*Corresponding author.

The Spiking Neural P Systems with Communication on Request are proved to be computationally universal, that is, equivalent with Turing machines as long as two types of spikes are used. Following this work, further research questions are listed to be open problems.

Keywords: Bio-inspired computing; membrane computing; P system; artificial neural network; spiking neural network.

1. Introduction

Membrane Computing is a branch of Natural Computing that abstracts computational models from the structure and the functioning of biological cells,[1,2] with a particular emphasis on their parallel and distributed computational features. These models are known under the name of *P Systems*.

A P system can be seen as a multicompartmental computing model (with the compartments delimited by *membranes*) characterized by the following points[3–6]:

- Its *structure* (i.e. membrane structure) which can be a hierarchical arrangement of membranes (thus, represented by means of a tree) or as a net of membranes and thus, represented by a general graph,
- its *multisets*, i.e. the molecules placed inside each membrane, with their multiplicity,
- its evolutionary *rules* that govern the operations on the multisets or the passage of the molecules across membranes.

Among the P systems, a *Neural-like P System* is a construct where the cells correspond to neurons, linked by synapses,[1] with a strong analogy between Neural-Like P systems[7] and neural networks.[8–12]

This paper focuses on a subclass of Neural-Like P systems, namely Spiking Neural (SN) P Systems.[13,14] SN P systems are a Membrane Computing version of the Spiking Neural Networks (SNNs).[15,16] In SNNs the communication among neurons is triggered by means of impulses of identical shape (spikes) or by sequences of spikes.[17–20] The training of a SNN is usually a complex task.[21]

SNNs have a wide application potential. Modern interesting applications of SNN include, e.g. epilepsy examination,[22–24] medical diagnostics,[25] pattern recognition,[26,27] neurosurgery,[28] information processing,[29] and liquid-state machine circuitry.[30]

SN P systems have some common features with SNNs: a neuron fires only when its potential or the number of spikes inside it reaches a specific value; the concept of time is incorporated into the information encoding and processing. In terms of features of models, SN P systems fall into the third generation of neural network models.

Briefly, SN P systems have the following structure and functioning. Neurons (in the form of a membrane) are placed in the nodes of a graph (whose edges are called synapses) and they contain a number of spikes. Identical objects denoted by a evolve by means of rules of the form $E/a^c \to a^p$: if the contents of the neuron are described by the regular expression E (over the alphabet $\{a\}$), then c spikes are consumed and p spikes are produced. The produced spikes are sent to all neurons: the synapse of each neuron points from the evolving neuron to each neighbor neuron. The p spikes are replicated in such a way that each destination neuron receives p spikes. In each time unit, each neuron that can use a rule should use one, while neurons in the system function in parallel with each other. When a computation halts, i.e. no further rule in the system can be applied, a result of the computation is obtained. In this work, the computation result is defined in the form of the number of spikes present in a specified neuron in the halting configuration.

Many variants of SN P systems have been considered in recent years, based on biological facts,[31–33] computer science motivations,[34,35] and mathematical motivations.[36–38] Most of the obtained classes of SN P systems are computationally universal, that is, they can simulate any Turing machine. Furthermore, several small computationally universal SN P systems have been constructed.[39] It has also been shown that, under certain conditions, solutions to computationally hard problems can be obtained in a polynomial time within this framework.[40,41] Successful applications of SN P systems have been presented in the areas of optimization[7] and fault diagnosis.[42–45] The interested reader can consult the above mentioned bibliography or the chapter[46] dedicated to SN P systems.

The aforementioned "standard" SN P systems and their variants are communicating *on command*: the initiative for communication belongs to the emitting neuron. Taking the inspiration from the area of Parallel-Cooperating Grammar Systems,[47] it is natural to consider also the reverse case: the communication *on request*. The spikes should be moved from a neuron to another one when the receiving neuron requests that.

Request-response is an important concept in software engineering. A request–response interaction (also called request-reply) is one of the three-event-based interaction types in an event-based system. The interactions among the agents in an event based system are governed by events, principally those interactions that are request-response, message-passing, or publish-subscribe.[48] A request-response interaction happens between two agents. Agent A makes a request to agent B by sending agent B a request, indicating the type of request along with the details of the request. Agent B processes the request and responds by sending a reply back to agent A. In a request-response interaction, there are potentially four events[48]: (1) the act of sending the request by agent A; (2) the receipt of the request by agent B; (3) the act of sending the reply by agent B; and (4) the receipt of the reply by agent A. For synchronous request-response interactions, especially those that occur over short periods of time, these four events are normally all combined together and considered one event. There are several particular cases.

Request-response is one of the basic methods computers used to communicate with each other, in which the first computer sends a request for some data and the second computer responds to the request.[49] For example, browsing a web page is an example of request-response communication. Request-response can be seen as a telephone call, in which someone is called and they answer the call. Request-response is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response.[49]

The class of SN P systems, we introduce here, namely, SN P Systems with Communications by Request (shortly called *SNQ P systems*), have only rules for requesting spikes from the neighboring neurons, the action being again dependent on the contents of the neuron. Basically, the rules are of the form $E/Q(a^{n_1}, j_1)\cdots(a^{n_m}, j_m)$, with the meaning that, if the neuron where this rule resides (say, neuron i) has a number of spikes described by the regular expression E, then it asks (this is the meaning of Q) n_1 spikes from neuron j_1, n_2 spikes from neuron j_2, and so on. If the neurons $j_1, \ldots, j_m$ cannot satisfy the requests (they contain less spikes than requested), then the rule cannot be applied. Also queries of the form (a^∞, j) can be formulated, with the meaning that all spikes from neuron j are requested, no matter how many they are, maybe none. When several neurons request simultaneously spikes from the same neuron, the queries should be identical, and the requested spikes are replicated. Details will be given in the next section.

We want to stress an important feature of this variant of SN P systems, *SNQ P systems*: no spike is consumed, they are only moved from a neuron to another one (from this point of view, they remind P systems with symport/antiport rules[50]). The only way to increase the number of spikes in the systems is by replicating the spikes in neurons which receive multiple queries.

In this work, the computational power of SNQ P systems is investigated. Specifically, the universality of SNQ P systems is obtained, if we extend the definition by considering two types of spikes. In addition, a small universal SNQ P system is obtained, composed of 49 neurons.

The remainder of the paper is organized in the following way. Section 2 describes the details of the proposed SN P Systems with Communications by Request. Section 3 contains some preliminary results about simple cases of SNQ P Systems, containing one, two, and three neurons. Section 4 discusses the universality of the proposed P systems. Section 5 provides an example of a small universal SN P system with communication by request. Further research questions are highlighted in Sec. 6. Section 7 gives the conclusions to this work.

2. Definition of SN P Systems with Communication by Request

This section formally defines the devices briefly described above. The reader is assumed to be familiar with basic elements of membrane computing,[46] as well as with some basic notions and notations from language and automata theory.[51] We only mention

that V^* denotes the free monoid generated by the alphabet V under the operation of concatenation with the null element λ (the empty string), and that the family of sets of natural numbers computed by Turing machines is denoted by NRE (they are the length sets of recursively enumerable languages, hence the notation).

We directly introduce the systems that we investigate, in the general form (with several types of spikes).

Definition 1 (SNQ *P* Systems). A *spiking neural P system with communication by request* (shortly, SNQ P system) is a construct

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, a_{i_0}, \text{out}),$$

where

(1) $O = \{a_1, a_2, \ldots, a_k\}$ is an alphabet (a_i is a type of spikes), where $k \geq 1$ is the number of types of spikes,
(2) $\sigma_1, \ldots, \sigma_m$ are *neurons* of the form $\sigma_i = (u_i, R_i)$, $1 \leq i \leq m$, m is the number of neurons, where:
   (a) u_i is a multiset over the alphabet O;
   (b) R_i is a finite set of rules of the form E/Qw, where E is a regular expression over O and w is a finite of *queries* of the forms (a_s^p, j) and (a_s^∞, j), $1 \leq s \leq k$, $p \geq 0$, $1 \leq j \leq m$;
(3) $a_{i_0}, 1 \leq i_0 \leq k$, is the *type of output spikes* and out $\in \{1, 2, \ldots, m\}$ indicates the *output neuron*, which is used to store the computation result.

The meaning of a query (a_s^p, j) is that neuron σ_i requests p copies of a_s from neuron σ_j, while (a_s^∞, j) means that all spikes of type a_s from σ_j, No matter how many they are, are requested by σ_i.

It must be noted that the set of *synapses* has not been specified in the definition since the synapses are implicitly defined by the rules.

A configuration C_t at an instant t of an SNQ P system $\Pi = (O, \sigma_1, \ldots, \sigma_m, a_{i_0}, \text{out})$, where $\sigma_i = (u_i, R_i)$ and $u_i = a_1^{n(i,1)} \ldots a_k^{n(i,k)}$, $1 \leq i \leq m$, is described by the number of spikes of each type present in each neuron in the beginning of the computation, that is, $C_t = ((n(1,1) \ldots n(1,k)), \ldots, (n(m,1) \ldots n(m,k)))$. A rule E/Qw, with a finite of queries w of the form (a_s^p, j), in neuron σ_i is applicable to a configuration C_t at time t if the following holds: (1) the contents of neuron σ_i considered as a string belongs to the language generated by E, and (2) all queries formulated in w are satisfied, that is, if (a_s^p, j) is a query in w then neuron σ_j at least p spikes (all query (a_s^∞, j) in w is always satisfiable because all spikes from σ_j are requested for neuron σ_i, no matter how many they are, maybe none). There could exist *conflicting queries* between two rules $r_1 \equiv E_1/Qw_1$ and $r_2 \equiv E_2/Qw_2$ associated with neurons σ_{i_1} and σ_{i_2} verifying conditions (1), (2), and such that by means of a query in w_1 and a query in w_2, different numbers of occurrences of the same spike a_s of neuron σ_j are requested by σ_{i1} and σ_{i2}. In this case, one of the rules r_1, r_2, nondeterministically chosen, can be used.

A delicate point appears when defining the result of a computational step because of the interplay of the queries. A *computation step* consists of the following *sub-steps*:

- **Sub-step 1.** In each neuron, we choose a rule to apply, and check its applicability. This means checking three conditions: (i) that the regular expression in the rule corresponds to the contents of the neuron, (ii) that the queries in the rule can be satisfied by the indicated neurons, and (iii) that there are no conflicting queries among the selected rules. If any of these conditions is not satisfied, then the rules should be changed, or, in the case of the third condition, some of the rules involved in conflicting queries should be omitted. However, the set of selected applicable rules should be maximal, in the sense that no rule can be added to the set without losing the applicability (each neuron which can evolve, should do it).
- **Sub-step 2.** The requested spikes are removed from the neurons where they were present. For each neuron we have three cases: (i) no spike a_s was requested by any other neuron (and then the existing number of spikes a_s remains unchanged), (ii) all spikes of some kind a_s were requested, by at least one other neuron (and then no spike of this kind remains here), or (iii) p spikes of type a_s are requested, by at least one other neuron (and then p is deduced from the number of copies of a_s present in the neuron). Note that because of the fact that the requests are not conflicting, we know precisely how many spikes of each type we have to deduce from each neuron.
- **Sub-step 3.** The queries are satisfied, the requested spikes are moved to the requesting

neurons. To the result of Sub-step 2, we add the requested spikes, with the following meaning: if two (or more) neurons request spikes from the same other neuron, then the number of spikes to be submitted to the two (or more) neurons is the same (say, p copies of some a_s), but only p spikes are removed from the emitting neuron, the p spikes are replicated and exactly p spikes are moved to each of the requesting neurons. The same in the case of two or more queries of the form (a_s^∞, j), all spikes present in σ_j are replicated as many times as the number of other neurons having submitted queries to σ_j.

It can be observed the three sub-steps together form a step, which lasts for one time unit.

After a computation step as illustrated above, the system passes to a new configuration. A sequence of such *transitions* from a configuration to another one, starting from the initial configuration, is called a *computation*. A computation *halts* if it reaches a configuration where no rule can be applied. The *result* of a halting computation is the number of copies of spike a_{i_0} present in neuron σ_{out} in the halting configuration.

2.1. *Differences between standard SN P systems and the proposed SNQ P systems*

It must be noted that there exist several important differences of SNQ P systems in comparison with usual SN P systems: (1) we use several types of objects, and we still call all of them spikes, (2) there is no interaction with the environment, no spike is sent out, hence we have to consider the result of a computation only in the internal mode (no spike train is defined here), (3) there is no other way to increase the number of spikes than the replication in the case of multiple queries from the same neuron (this corresponds to the case when a neuron in a usual SN P system sends spikes to several neurons to which it has synapses).

It is also worth mentioning the difference from the systems, we consider here and those in Ref. 52, where the request is done only from the environment, for cell-like SN P systems,[53] using (in the skin region only) rules of the form $E/\lambda \leftarrow a^p$, with the meaning that p spikes are brought from the environment. Besides these rules, usual spiking rules are used in Ref. 52.

3. Preliminary Theoretical Findings

Let us start the study of SNQ P systems by examining the computational power of small systems, which have a small number of neurons and of types of spikes. This is also an opportunity to illustrate the previous definitions by means of some examples.

For an SNQ P system Π, let us denote by $N(\Pi)$ the set of numbers generated by Π. Let us also denote by $NSN_kP_m(Q)$ the family of sets $N(\Pi)$ generated by SNQ P systems using at most k types of spikes and at most m neurons. When the numbers k or m can be arbitrary, the corresponding parameter is replaced with $*$.

Directly from the definition of SNQ P systems, it follows that the two parameters k and m induce a double hierarchy of families of sets of numbers:

$$NSN_kP_m(Q) \subseteq NSN_{k+1}P_{m+1}, \quad \text{for all } k \geq 1, m \geq 1.$$

As we will see in the next section, the hierarchy on the number of types of spikes collapses at the second level (SNQ P systems with two types of spikes are already universal). Because of the universality, the other hierarchy on the number of neurons cannot be infinite, but we do not know its precise height.

The following two lemmas analyze oversimplified SNQ P systems, composed of one and two neurons, respectively. It is shown that, as for other P systems, systems with only one neuron cannot apply any rule, hence they only generate singleton sets.

Lemma 1. *$NSN_kP_1 = NSN_*P_1 = SING, k \geq 1$, where SING denotes the family of singleton sets.*

Systems with two neurons have also a rather limited power.

Lemma 2. *$NSN_kP_2 = NSN_*P_2 = FIN, k \geq 1$, where FIN denotes the family of finite sets of numbers.*

Proof. In systems with two neurons, spikes cannot be replicated, hence the initial number of spikes cannot be increased, and $NSN_*P_2 \subseteq FIN$.

On the other hand, $FIN \subseteq NSN_1P_2$: consider a finite set of numbers, arranged in the increasing

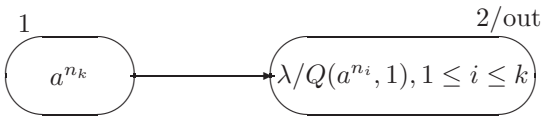

Fig. 1. An SNQ P system generating a finite set.

order, $n_1 < n_2 < \cdots < n_k$, and consider the SNQ P system from Fig. 1. We use the standard style in representing SN P systems; we also explicitly represent the synapses defined by the queries, as well as the initial spikes present in neurons (if no spike is specified, this means that no spike is present in that neuron in the initial configuration). Also as usual in the area of SN P systems, we identify a neuron with its label, thus equivalently saying "neuron σ_i" and "neuron i".

Initially, neuron σ_1 contains n_k spikes, while σ_2 is empty, corresponding to the empty string λ. Each computation takes only one step, neuron σ_2 nondeterministically chooses rule $\lambda/Q(a^{n_i},1)$ to apply, requesting n_i spikes from neuron σ_1. In this way, neuron σ_2 has n_i spikes, thus the number n_i, $1 \leq i \leq k$, is generated. □

The passage from two neurons to three neurons entails a rather large increase of the generative power, and the explanation resides in the possibility to have replication of spikes, not only permitting the increase of number of spikes, but even an exponential increase.

Lemma 3. *The family $NSN_1P_3(Q)$ contains any arithmetical progression.*

Proof. Let us take an arithmetical progression $L = \{n_0 + i \cdot n_1 \mid i \geq 1\}$, for some $n_0 \geq 0$ and $n_1 \geq 1$, and construct the SNQ P system Π from Fig. 2, which consists of three neurons with labels 1, 2 (out, output neuron), and 3.

Formally, the system is:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, a, 2), \quad \text{where}$$

$$\sigma_1 = (n_1, \{a^{n_1}/Q(a^{n_1},3),\ a^{n_1}/Q(a^{n_1-1},3)\}),$$

$$\sigma_2 = (n_0, \{a^{n_0}(a^{n_1})^*/Q(a^{n_1},1)\}),$$

$$\sigma_3 = (n_1, \{a^{n_1}/Q(a^{n_1},1)\}).$$

In each step, neurons σ_1 and σ_3 repeatedly exchange n_1 spikes, while neuron σ_2 also requests n_1 spikes from neuron σ_1 (hence the n_1 spikes of neuron σ_1 are duplicated), thus going along the terms of the arithmetical progression. The computation can stop at any moment, by using the second rule of neuron σ_1: neuron σ_1 brings only $n_1 - 1$ spikes inside (hence the query of neuron σ_2 cannot be satisfied) and neuron σ_3 remains with one spike inside, which, together with the n_1 spikes brought from neuron σ_1, do not allow the use of the rule in neuron σ_3. Clearly, $N(\Pi) = L$. □

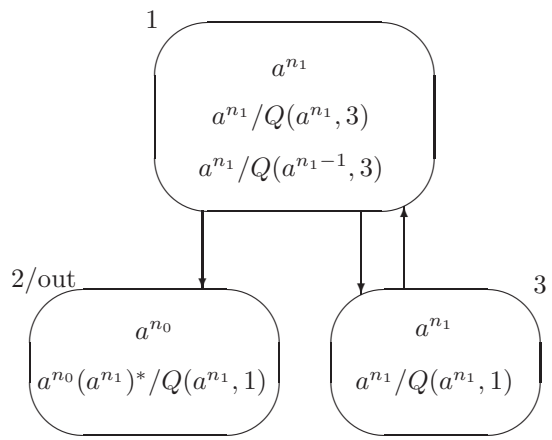

Fig. 2. An SNQ P system generating an arithmetical progression.

Lemma 4. *The family $NSN_1P_3(Q)$ contains non-semilinear sets of numbers.*

Proof. Let us consider the SNQ P system Π in Fig. 3.

The neurons $\sigma_1, \sigma_2, \sigma_3$ can use a rule only if they are empty. This is the case in the beginning with neurons σ_2 and σ_3, hence they request all spikes

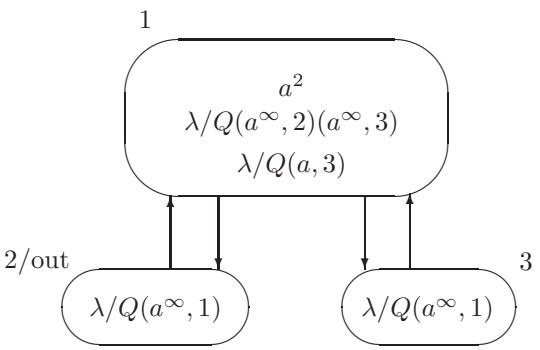

Fig. 3. An SNQ P system generating a non-semilinear set.

of neuron σ_1. Now neuron σ_1 can request back the spikes from neurons σ_2, σ_3, getting 4 spikes inside. As long as the neurons use their rules asking for all spikes of the partner neurons, the number of spikes present in neuron σ_1 is doubled, and this also happens with the contents of neurons σ_2 and σ_3.

At some step, $\lambda/Q(a,3)$ is used in neuron σ_1, simultaneously with neurons σ_2, σ_3 requesting the spikes of neuron σ_1. The computation halts, because all neurons have at least one spike inside, hence they can use no rule (all rules have the empty string λ). The number of spikes in neuron σ_1 is doubled after each move of the contents of neurons σ_2 and σ_3 to neuron σ_1, after neuron σ_2 requests all spikes from neuron σ_1, the number of spikes in the output neuron σ_2 is a power of 2, $N(\Pi) = \{2^n \mid n \geq 1\}$. □

Therefore, the increase of the number of neurons from 1 to 2 and to 3 induces a strict increase of the computing power of SNQ P systems. It remains to be investigated whether the strict increase of computing power is also true for the next levels of the hierarchies $NSN_kP_m(Q) \subseteq NSN_kP_{m+1}$.

4. The Universality of SNQ *P* Systems

This section gives the main result of the paper, that is the universality of SNQ P systems with two types of spikes (without a bound on the number of neurons). The proof will use the characterization of NRE by means of register machines.

Such a device is a construct $M = (n, H, l_1, l_h, I)$, where n is the number of registers, H is the set of instruction labels, l_1 is the start label (for simplicity, we may assume that l_1 labels an `ADD` instruction, but this is not essential; note that in many places the start instruction is labeled with l_0, but here we prefer to start from 1), 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 nonempty, 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 starts with all registers empty (i.e. storing the number zero), applies the instruction with label l_1 and proceeds to apply instructions as indicated by labels (and made possible by the contents of registers); if the machine reaches 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 $N(M)$. If the computation never halts, then no number is generated. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE (see, e.g. Ref. 54).

Theorem 1 (Universality of NSQ *P* Systems). *SNQ P Systems are computationally universal:* $NRE = NSN_2P_*(Q)$.

Proof. In order to prove the theorem, we only prove the inclusion $\subseteq$, the opposite one can be obtained through a straightforward but cumbersome construction of a Turing machine simulating an SNQ P system, or we can invoke the Turing–Church thesis.

Starting from a register machine $M = (n, H, l_0, l_h, I)$, we construct an SNQ P system Π with two types of spikes, which we denote by a and b, hence $O = \{a, b\}$. We associate one neuron with each register of M (denoted by $1, 2, \ldots, n$), one neuron σ_l with each label $l \in H$, as well as a second neuron, $\sigma_{l_i'}$ with each instruction of M of the form $l_i : (\texttt{SUB}(r), l_j, l_k)$. We also consider the neurons $\sigma_i, 1 \leq i \leq 5$, as mentioned below. Therefore, the number of neurons depends on the number of registers and labels of M, that is why we cannot bound it in advance.

If the value of a register r is m, then the corresponding neuron σ_r contains m spikes a. That is, a is the spike indicating the result of the computation, and the output neuron is σ_1 (associated with register 1).

Let us assume that H contains t elements, $l_1, l_2, \ldots, l_t$. Initially, each neuron contain $2t$ copies of b and no copy of a, with the exception of neurons $\sigma_{c_i}, 1 \leq i \leq 5$, which contain the spikes a, b, b, b, b, respectively (see also the construction below).

`ADD` **module:** For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M, we construct the module represented in Fig. 4.

In order to simplify the proof, we first assume that we are allowed to also use a query of the form (a, env), with the meaning that one copy of a is

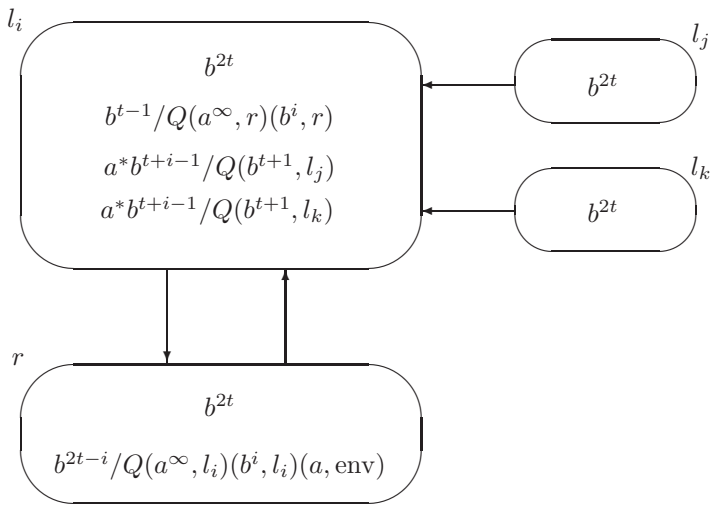

Fig. 4. The ADD module.

requested from the environment — with the environment supposed to contain arbitrarily many copies of a. Later we will remove this kind of rules.

Such a module is activated when $t+1$ spikes b from neuron l_i are requested by another neuron. In this construction, the neuron which asks $t+1$ copies of b from σ_{l_i} is σ_{c_1}. If we would accept the query (a, env), hence neurons c_1, c_2, c_3 will be absent, then we have to start with only $t-1$ spikes b in neuron l_1, the starting one. The neuron (with label) l_i becomes active and it requests from neuron r all copies of a as well as i copies of b. Note that i precisely identifies the label l_i, which precisely identifies the instruction (hence neurons r, l_j, l_k).

In the next step, both neurons l_i and r can apply a rule. In this way, the previous contents of neuron r returns to neuron r, at the same time neuron r requesting one copy of a from the environment, which corresponds to the fact that the register was increased by 1. Simultaneously, neuron l_i uses one of the rules $a^* b^{t+i-1}/Q(b^{t+1}, l_j)$, $a^* b^{t+i-1}/Q(b^{t+1}, l_k)$, nondeterministically chosen, which means that one of the neurons l_j, l_k is activated, while l_i ends with $2t$ copies of b inside, as it was the case at the beginning. The instruction of M is correctly simulated, and one of the instructions with label l_j, l_k will be simulated in the next steps.

It is important to note that, in spite of the fact that several instructions ADD can refer to the same register r (as well as several instructions SUB), this does not lead to wrong computations (i.e. computations in Π not corresponding to computations in M), because the regular expression b^{2t-i} of the rules in neuron r precisely identifies the neuron l_i to which a query is addressed from neuron r.

Let us see now how to avoid the query (a, env). Instead of the query (a, env), we put in neuron r the query (a, c_1), and then we consider the module consisting of three neurons given in Fig. 5. Their role is to produce arbitrarily many copies of spike a, keeping them available to neurons with label r corresponding to ADD instructions, then to request $t+1$ copies of spike b from neuron l_1, thus triggering the simulation of the first instruction in M.

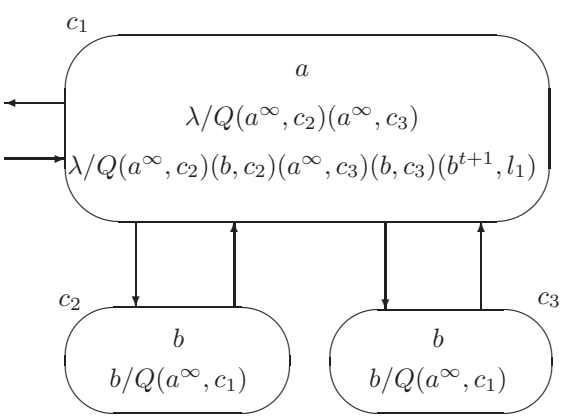

Fig. 5. The module producing arbitrarily many copies of spike a.

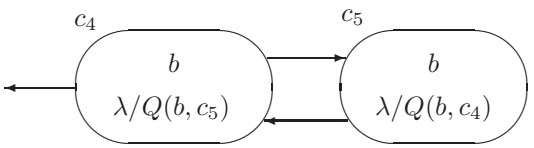

Fig. 6. The trap module.

This module functions in a way similar to the system in Fig. 3: as long as neurons c_2, c_3 contains one copy of spike b, they can request the contents of c_1, which then can bring back, doubled, the number of spikes a, repeatedly, until nondeterministically choosing to use the second rule, $\lambda/Q(a^\infty, c_2)(b, c_2)(a^\infty, c_3)(b, c_3)(b^{t+1}, l_1)$. This rule blocks the functioning of neurons c_2, c_3, and also activates neuron l_1.

It is easy to see that this module substitutes the use of the query (a, env), with only one exception: if the module in Fig. 5 stops "too early" and there are not enough copies of spike a, as necessary for the simulation of the computation in M. To avoid this situation, we also consider a "trap module": we add the rule $b^{2t-i}/Q(b, c_4)$ to neuron r. If the rule $b^{2t-i}/Q(a^\infty, l_i)(b^i, l_i)(a, c_1)$ cannot be used because neuron c_1 contains no copy of spike a, then this new rule should be used, requesting one copy of spike b from neuron c_4. This neuron is a part of the module in Fig. 6.

With spikes b inside, neurons c_4 and c_5 cannot use any rule, but after removing the spike b from c_4, neurons c_4 and c_5 will repeatedly ask to each other the remaining spike b, and the computation never stops.

`SUB` **module:** For each instruction $l_i : (\texttt{SUB}\ (r), l_j, l_k)$ of M we construct the module given in Fig. 7.

When neuron l_i "looses" $t+1$ spikes b, it becomes active, and can absorb all spikes a and i spikes b from neuron r. In the next step, both neurons r and l_i can use one rule. If there is no spike a present (corresponding to the fact that register r was empty), then neuron l_i has to use the rule $b^{t+i-1}/Q(b^{t+1}, l_k)$, and neuron l_k is activated. In parallel, neuron r returns to its previous contents (no subtraction was made), neurons l'_i, l_j are not modified.

If there is at least one copy of spike a present, the subtraction is performed by activating first neuron l'_i (in parallel, neuron r returns to its previous contents). Neuron l'_i decreases by one the contents of neuron r and activates neuron l_j. The copies of spike a requested by neuron l'_i during a computation remain in this neuron, they are "accepted" by the regular expression of the rule in l'_i.

Again, no unwanted interferences between `SUB` modules appear, because the label l_i precisely identifies the instruction (hence the module). In this way, the `SUB` instruction is also correctly simulated.

The simulation of the computation in M continues until the halt instruction is reached. In the neuron associated with l_h there is no rule, hence after activating this neuron, the computation in Π halts. The number of copies of spike a in neuron 1 is the result of computation, hence $N(M) = N(\Pi)$. □

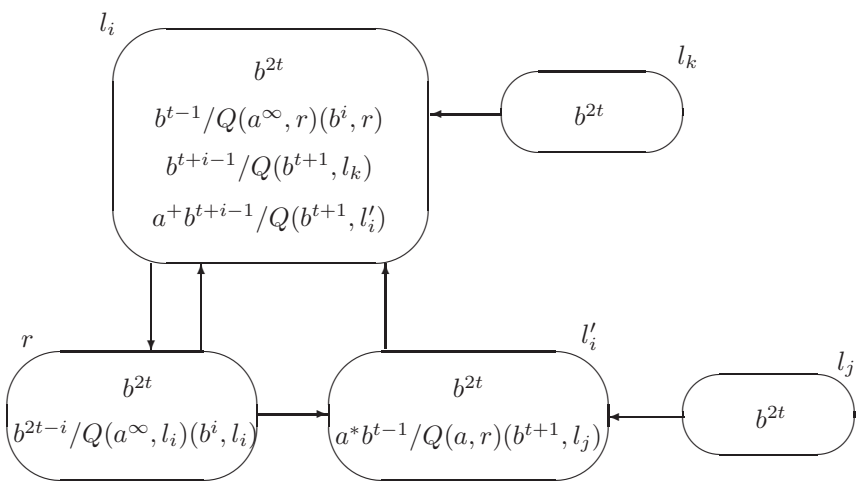

Fig. 7. The `SUB` module.

5. A Small Universal SNQ P System

In this section, starting from a universal register machine, as those presented in Ref. 55, a universal SNQ P system will be obtained.

In Ref. 55, the register machines are used for computing functions, with the universality defined as follows. Let $(\varphi_0, \varphi_1, \ldots)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y, we have $\varphi_x(y) = M_u(g(x), y)$. In Ref. 55, several universal register machines are constructed, with the input (the couple of numbers $g(x)$ and y) introduced in specified input registers and the result obtained in another specified register, the output one.

The machine from Ref. 55 used in Ref. 39 is given in Fig. 8. It has 8 registers and 23 instructions. Without loss of generality, l_0 labels the start instruction, which has no effect over the assumption about the definition of register machines specified in the last section. Because here, we do not work with numbers encoded in the spike train, as the distance in time between consecutive spikes, but with the multiplicity of spike a in specified neurons, we can have the input and the output of a computation in an SNQ P system defined in the same way as in register machines, hence no input and output module as in Ref. 39 is necessary.

Therefore, a direct counting on the modules constructed in the previous proof (8 registers + 23 labels + 13 SUB instructions + 5 neurons in Figs. 5 and 6 means a total of 49 neurons) leads to the following result.

$l_0 : (\texttt{SUB}(1), l_1, l_2)$, $l_1 : (\texttt{ADD}(7), l_0)$,
$l_2 : (\texttt{ADD}(6), l_3)$, $l_3 : (\texttt{SUB}(5), l_2, l_4)$,
$l_4 : (\texttt{SUB}(6), l_5, l_3)$, $l_5 : (\texttt{ADD}(5), l_6)$,
$l_6 : (\texttt{SUB}(7), l_7, l_8)$, $l_7 : (\texttt{ADD}(1), l_4)$,
$l_8 : (\texttt{SUB}(6), l_9, l_0)$, $l_9 : (\texttt{ADD}(6), l_{10})$,
$l_{10} : (\texttt{SUB}(4), l_0, l_{11})$, $l_{11} : (\texttt{SUB}(5), l_{12}, l_{13})$,
$l_{12} : (\texttt{SUB}(5), l_{14}, l_{15})$, $l_{13} : (\texttt{SUB}(2), l_{18}, l_{19})$,
$l_{14} : (\texttt{SUB}(5), l_{16}, l_{17})$, $l_{15} : (\texttt{SUB}(3), l_{18}, l_{20})$,
$l_{16} : (\texttt{ADD}(4), l_{11})$, $l_{17} : (\texttt{ADD}(2), l_{21})$,
$l_{18} : (\texttt{SUB}(4), l_0, l_h)$, $l_{19} : (\texttt{SUB}(0), l_0, l_{18})$,
$l_{20} : (\texttt{ADD}(0), l_0)$, $l_{21} : (\texttt{ADD}(3), l_{18})$,
$l_h : \texttt{HALT}$.

Fig. 8. The universal register machine from Ref. 55.

Theorem 2. *There is a computing universal SNQ P system with* 49 *neurons.*

It is highly possible that the number 49 can be slightly improved (by looking to other universal register machines in Ref. 55, by possibly saving some neurons by carefully examining the structure of the starting universal register machine, or by using a different construction). This task is left as an open problem to the reader.

6. Further Research Questions

Several questions naturally remain unaddressed in this preliminary phase of the study. These questions open multiple unexplored research directions. The following list illustrates some these questions and research directions to continue the research on SNQ P systems.

(1) Can the universality be obtained also for SNQ P systems using only one type of spikes? (Do we have $NRE = NSN_1P_*(Q)$?) We expect a negative answer, and a confirmation of this conjecture would be rather interesting, as not so many classes of P systems are known which are not universal (but are able to compute more than semilinear sets — see the example from Sec. 3).
(2) We have seen that the replication can grow exponentially the number of spikes in linear time. Can this be used in order to solve **NP**-complete problems in polynomial time? We again expect a negative answer — prove a *Milano theorem* version for SNQ P systems (prove that an SNQ P system can be simulated by a Turing machine with a polynomial slow-down, as done in Ref. 56 for multiset processing P systems and in Ref. 57 for usual SN P systems).
(3) Without duplication (and without bringing spikes from the environment) the number of spikes present in the system remains constant, hence only regular sets of numbers can be generated. Can *all* regular sets be generated in this way?
(4) Look for normal forms, e.g. in terms of the number of neurons from which a rule can request spikes. In the proof of Theorem 1, we have rules requesting spikes from 1, 2 or even 3 neurons (the case of neuron c_1). Is the universality lost if we bound this number to 1 or 2?

(5) An interesting kind of queries seems to be those of the form $(a^{\infty-s}, j)$: take all but s spikes a from neuron σ_j. Of course, if σ_j contains less than s spikes, then the query cannot be satisfied, the rule cannot be applied. Are such queries useful (for instance, in avoiding the use of the second type of spikes in the universality proof)?

(6) Although we do not have a spike train associated with a computation in an SNQ P system, we can associate a language to such a system in terms of traces, as in Ref. 58: follow the path of a designated spike from a neuron to another one. The family of these trace languages remains to be investigated.

(7) A natural question would be: how can we use SNQ P systems and what would be the advantages/differences with respect to traditional SN P systems? Although we do not have a definite answer to this question yet, like the traditional SN P systems, SNQ P systems could be used to sort, and to represent fuzzy knowledge and diagnose faults occurring in an electric power system by combing with fuzzy set theory. In addition, the promising applications of SNQ P systems might be used to construct arithmetic or logic operators and to detect network intrusion by making full use of the characteristic of communication on request.

(8) Finally, we point out another natural question: can we remove the regular expressions from the rules and replace them with polarizations associated with the membranes, as done in Ref. 59 for standard SN P systems? This made the universality proof in Ref. 59 much more difficult, so this is expected also for SNQ P systems — or maybe they will no longer be universal in this case.

7. Conclusion

This paper proposes a new class of spiking neural P systems, where the spikes are not spontaneously emitted but the communication is initiated *on request*, by the requesting neurons. The obtained computing devices are briefly called SNQ P systems.

These SNQ P systems are proved to be computationally universal, that is, they can simulate the Turing machines — provided that two types of spikes are used. As a consequence of the proof of this result, a small universal SNQ P system is obtained, composed of 49 neurons.

Several research questions remain unaddressed and require further investigation. Eight open questions have been identified but several others can be considered. Of a particular interest is the question of solving computationally hard problems, e.g. **NP**-complete problems, in a polynomial time. As mentioned in the introduction, SNNs have a wide application potential. As a membrane computing version of SNNs, SNQ P systems are deserved to be investigated for solving real-world problems.

Acknowledgments

The work of L. Pan was supported by National Natural Science Foundation of China (61320106005 and 61772214) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012). The work of G. Zhang was supported by National Natural Science Foundation of China (61373047, 61672437, 61702428 and 51641506) and the Research Project of Key Laboratory of Fluid and Power Machinery (Xihua University), Ministry of Education, P. R. China (JYBFXYQ-1).

References

1. G. Păun, *Membrane Computing — An Introduction* (Springer-Verlag, New York, 2002).
2. G. Zhang, M. J. Pérez-Jiménez and M. Gheorghe, *Real-life Applications with Membrane Computing* (Springer International Publishing, Berlin, 2017).
3. X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate and R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, *Integr. Comput.-Aided Eng.* **23**(1) (2015) 15–30.
4. G. Zhang, J. Cheng, M. Gheorghe and Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Appl. Soft Comput.* **13**(3) (2013) 1528–1542.
5. H. Peng, J. Wang, P. Shi, M. J. Pérez-Jiménez and A. Riscos-Núñez, An extended membrane system with active membranes to solve automatic fuzzy clustering problems, *Int. J. Neural Syst.* **26**(3) (2016) 1650004.
6. X. Liu and J. Xue, A cluster splitting technique by Hopfield networks and P systems on simplices, *Neural Process. Lett.* **46**(1) (2017) 171–194.

7. G. Zhang, H. Rong, F. Neri and M. J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, *Int. J. Neural Syst.* **24**(5) (2014) 1440006.
8. S. Iliya and F. Neri, Towards artificial speech therapy: A neural system for impaired speech segmentation, *Int. J. Neural Syst.* **26**(6) (2016) 1650023.
9. M. Ahmadlou and H. Adeli, Enhanced probabilistic neural network with local decision circles: A robust classifier, *Integr. Comput.-Aided Eng.* **17**(3) (2010) 197–210.
10. N. Wang and H. Adeli, Self-constructing wavelet neural network algorithm for nonlinear control of large structures, *Eng. Appl. Artifi. Intell.* **41** (2015) 249–258.
11. A. Rigos, G. E. Tsekouras, M. I. Vousdoukas, A. Chatzipavlis and A. F. Velegrakis, A chebyshev polynomial radial basis function neural network for automated shoreline extraction from coastal imagery, *Int. Comput.-Aided Eng.* **23**(2) (2016) 141–160.
12. Y. Zeinali and B. A. Story, Competitive probabilistic neural network, *Integr. Comput.-Aided Eng.* **24**(2) (2017) 105–118.
13. M. Ionescu, G. Păun and T. Yokomori, Spiking neural P systems, *Fundam. Informa.* **71** (2006) 279–308.
14. L. Pan, T. Wu and Z. Zhang, A bibliography of spiking neural P systems, *Bull. IMCS* **1** (2016) 63–78. Available at: http://membranecomputing.net/IMCSBulletin/.
15. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
16. W. Maass, Networks of spiking neurons: The third generation of neural network models, *Neural Netw.* **10**(9) (1997) 1659–1671.
17. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integr. Comput.-Aided Eng.* **14**(3) (2007) 187–212.
18. S. Nobukawa and H. Nishimura, Enhancement of spike-timing-dependent plasticity in spiking neural systems with noise, *Int. J. Neural Syst.* **26**(5) (2016) 1550040.
19. Z. Wang, L. Guo and M. Adjouadi, A generalized leaky integrate-and-fire neuron model with fast implementation method, *Int. J. Neural Syst.* **24**(05) (2014) 1440004.
20. C. Savin, P. Joshi and J. Triesch, Independent component analysis in spiking neurons, *Plos Comput. Biol.* **6**(4) (2010) e1000757.
21. X. Zhang, G. Foderaro, C. Henriquez and S. Ferrari, A scalable weight-free learning algorithm for regulatory control of cell activity in spiking neuronal networks, *Int. J. Neural Syst.* (2017) 1750015.
22. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Netw.* **22**(10) (2009) 1419–1431.
23. C. Luo, Y. Zhang, W. Cao, Y. Huang, F. Yang, J. Wang, S. Tu, X. Wang and D. Yao, Altered structural and functional feature of striato-cortical circuit in benign epilepsy with centrotemporal spikes, *Int. J. Neural Syst.* **25**(6) (2015) 1550027.
24. L. Guo, Z. Wang, M. Cabrerizo and M. Adjouadi, A cross-correlated delay shift supervised learning method for spiking neurons with application to interictal spike detection in epilepsy, *Int. J. Neural Syst.* **27**(3) (2017) 1750002.
25. A. Geminiani, C. Casellato, A. Antonietti, E. D'Angelo and A. Pedrocchi, A multiple-plasticity spiking neural network embedded in a closed-loop control system to model cerebellar pathologies, *Int. J. Neural Syst.* (2017) 1750017.
26. J. A. Garrido, N. R. Luque, S. Tolu and E. D'Angelo, Oscillation-driven spike-timing dependent plasticity allows multiple overlapping pattern recognition in inhibitory interneuron networks, *Int. J. Neural Syst.* **26**(5) (2016) 1650020.
27. A. Morro, V. Canals, A. Oliver, M. Alomar, F. Galán-Prado, P. Ballester and J. Rosselló, A stochastic spiking neural network for virtual screening, *IEEE Trans. Neural Netw. Learn. Syst.* (2017), available http://ieeexplore.ieee.org/document/7845709/.
28. S. Knieling, K. S. Sridharan, P. Belardinelli, G. Naros, D. Weiss, F. Mormann and A. Gharabaghi, An unsupervised online spike-sorting framework, *Int. J. Neural Syst.* **26**(5) (2016) 1550042.
29. J. L. Rossello, V. Canals, A. Oliver and A. Morro, Studying the role of synchronized and chaotic spiking neural ensembles in neural information processing, *Int. J. Neural Syst.* **24**(05) (2014) 1430003.
30. J. L. Rosselló, M. L. Alomar, A. Morro, A. Oliver and V. Canals, High-density liquid-state machine circuitry for time-series forecasting, *Int. J. Neural Syst.* **26**(5) (2016) 1550036.
31. F. G. C. Cabarle, H. N. Adorna and M. J. Pérez-Jiménez, Sequential spiking neural P systems with structural plasticity based on max/min spike number, *Neural Comput. Appl.* **27**(5) (2016) 1337–1347.
32. T. Song, L. Pan and G. Păun, Asynchronous spiking neural P systems with local synchronization, *Inf. Sci.* **219** (2013) 197–207.
33. J. Wang, H. J. Hoogeboom, L. Pan, G. Păun and M. J. Pérez-Jiménez, Spiking neural P systems with weights, *Neural Comput.* **22**(10) (2010) 2615–2646.
34. Y. Yu, T. Wu, J. Xu, Y. Wang and J. He, A note on spiking neural P systems with homogenous neurons and synapses, *Fundam. Inform.* **150**(2) (2017) 231–240.
35. M. Cavaliere and I. Mura, Experiments on the reliability of stochastic spiking neural P systems, *Nat. Comput.* **7**(4) (2008) 453–470.

36. M. Cavaliere, O. H. Ibarra, G. Păun, O. Egecioglu, M. Ionescu and S. Woodworth, Asynchronous spiking neural P systems, *Theor. Comput. Sci.* **410**(24–25) (2009) 2352–2364.
37. O. H. Ibarra, A. Păun, G. Păun, A. Rodríguez-Patón, P. Sosík and S. Woodworth, Normal forms for spiking neural P systems, *Theor. Comput. Sci.* **372**(2–3) (2007) 196–217.
38. M. García Arnau, D. Peréz, A. Rodríguez Patón and P. Sosík, Spiking neural P systems: Stronger normal forms, *Int. J. Unconv. Comput.* **5** (2009) 411–425.
39. A. Păun and G. Păun, Small universal spiking neural P systems, *BioSystems* **90** (2007) 48–60.
40. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng and X. Zhang, Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with precomputed resources, *Theor. Comput. Sci.* **411** (2010) 2345–2358.
41. L. Pan, G. Păun and M. Pérez-Jiménez, Spiking neural P systems with neuron division and budding, *Sci. China Inf. Sci.* **58** (2011) 1596–1607.
42. T. Wang, G. Zhang and M. J. Pérez-Jiménez, Fuzzy membrane computing: Theory and applications, *Int. J. Comput. Commun. Control* **10** (2015) 904–935.
43. H. Peng, J. Wang, M. J. Pérez-Jiménez, H. Wang, J. Shao and T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, *Inf. Sci.* **235** (2013) 106–116.
44. J. Wang, P. Shi, H. Peng, M. J. Pérez-Jiménez and T. Wang, Weighted fuzzy spiking neural P systems, *IEEE Trans. Fuzzy Syst.* **21**(2) (2013) 209–220.
45. T. Wang, G. Zhang, J. Zhao, Z. He, J. Zhao, J. Wang and M. J. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, *IEEE Trans. Power Syst.* **30**(3) (2015) 1182–1194.
46. G. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing* (Oxford University Press, Oxford, 2010).
47. E. Csuhaj-Varjú, J. Dassow, J. Kelemen and G. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation* (Gordon and Breach, London, 1994).
48. G. Mühl, L. Fiege and P. Pietzuch, *Distributed Event-Based Systems* (Springer-Verlag, New York, 2006).
49. Available at: https://en.wikipedia.org/wiki/Request.
50. A. Păun and G. Păun, The power of communication: P systems with symport/antiport, *New Gener. Comput.* **20** (2002) 295–306.
51. G. Rozenberg and A. Salomaa, *Handbook of Formal Languages* (Springer-Verlag, Berlin, 1997).
52. T. Song and L. Pan, Spiking neural P systems with request rules, *Neurocomputing* **193** (2016) 193–200.
53. T. Wu, Z. Zhang, G. Păun and L. Pan, Cell-like spiking neural P systems, *Theor. Comput. Sci.* **623** (2016) 180–189.
54. M. Minsky, *Computation — Finite and Infinite Machines* (Prentice Hall, Englewood Cliffs, NJ, 1967).
55. I. Korec, Small universal register machines, *Theor. Comput. Sci.* **168** (1996) 267–301.
56. C. Zandron, A model for molecular computing: Membrane systems, PhD thesis, Univ. degli Studi di Milano (2001).
57. A. Leporati, C. Zandron, C. Ferretti and G. Mauri, On the computational power of spiking neural P systems, *Int. J. Unconv. Comput.* **5**(5) (2009) 459–473.
58. M. Ionescu, A. Păun, G. Păun and M. Pérez-Jiménez, Computing with spiking neural P systems: Traces and small universal systems, in *DNA Computing. 12th International Meeting on DNA Computing*, *Lecture Notes in Computer Science*, Vol. 4287 (Seoul, Korea, 2006), pp. 1–16.
59. T. Wu, A. Păun, Z. Zhang and L. Pan, Spiking neural P systems with polarizations, *IEEE Trans. Neural Netw. Learn. Syst.* (2017), doi:10.1109/TNNLS.2017.2726119.