ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **19**, Numbers 1-2, 2016, 5–16

Four Recent Research Topics on Numerical and Spiking Neural P Systems

Linqiang PAN[1], Gheorghe PĂUN[2], Tingfang WU[1], Zhiqiang ZHANG[1]

[1]Key Laboratory of Image Information Processing and Intelligent Control
School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: {lqpan,tfwu,zhiqiangzhang}@hust.edu.cn

[2]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: gpaun@us.es

Abstract. Two research directions on numerical P systems and two on spiking neural P systems which were recently explored are shortly discussed. They deal with very natural versions of these classes of P systems which were not investigated before. Besides definitions, one recalls a few of the results obtained about these new classes of P systems and one points out open problems and directions for further research.

1. Introduction

Although membrane computing is considered as a young branch of natural computing (it was initiated in [5]), it is already a well developed research area, both at the theoretical level (a comprehensive presentation, at the level of the beginning of 2009, can be found in [8]) and in what concerns applications (see, *e.g.*, the volumes [1], [2], [12]).

In particular, in membrane computing there are numerous open problems and research topics in circulation, many of them are also collected in systematic lists and compiled, for instance, for the yearly Brainstorming Week on Membrane Computing, see http://ppage.psystems.eu and www.gcn.us.es. Also the "trace" of research about some of these problems can be found, *e.g.*, in [6].

Two of the much investigated classes of P systems, the spiking neural (in short, SN) P systems and the numerical P systems, were both introduced ten years ago, in [3] and [7], respectively. Bibliographies of each of these areas can be found in *Bulletin of the International Membrane Computing Society*, June 2016, `http://membranecomputing.net/IMCSBulletin/index.php`. However, many research ideas about these classes of P systems are still not completely explored, while new ideas appear continuously. Four research topics are briefly discussed here, which arose during a three months stay of the third and the fourth authors in Curtea de Argeş, Romania, in the fall of 2015.

Three of these research topics (cell-like SN P systems, numerical P systems with target indications or generating strings) are very natural extensions of the basic definitions, while the fourth one – SN P systems with polarizations associated with neurons, but without any other control on the use of the rules – is both significantly more restrictive than the standard definition and challenging from a computational point of view, because of its minimalistic definition.

As the reader of this note is supposed to be a researcher in membrane computing, we do not give basic definitions, but we provide full details for the new classes of P systems considered. As a general reference we refer to the Handbook [8].

2. Cell-like SN P Systems

In SN P systems investigated so far, the "neurons" are placed in the nodes of an arbitrary graph, while many classes of P systems considered in membrane computing use a cell-like arrangement of membranes, hence described by a tree. SN P systems with a cell-like membrane structure may look "non-natural" from a biological point of view, but they are mathematically interesting, especially in view of the children-parent membrane interaction; recalling that circularity is not allowed in standard SN P systems.

This version of SN P systems was investigated in [11]; we recall some details.

A *cell-like SN P system* (in short, a cSN P system), of degree $m \geq 1$, is a construct

$$\Pi = (O, \mu, n_1, \ldots, n_m, R_1, \ldots, R_m, i_o),$$

where $O = \{a\}$, μ is a hierarchical membrane structure with m membranes, $n_i, 1 \leq i \leq m$, is the number of spikes present in compartment i of μ at the beginning of the computation, $R_i, 1 \leq i \leq m$, is the finite set of rules from compartment i, and i_o indicates the output region (this is the environment if $i_o = env$).

Besides forgetting rules of the form $a^s \to \lambda$, $s \geq 1$, the sets R_i contain *spiking rules* of the (extended) form $E/a^c \to u$, where E is a regular expression over O, $c \geq 1$, and u is a sequence of couples of the form (a^p, tar), where $p \geq 1$ and tar is a target indication specifying the destination of the p associated spikes. This target can be *here, out, in*, in_j, where j is the label of a membrane, with the usual meaning in cell-like P systems, or in_{all}, with the meaning that the p spikes will be sent, replicated, to all immediately inner membranes (each of them will receive p spikes). Of course, in

the case of non-extended rules, when only one spike is produced by a rule, only one couple of the form (a, tar) will be used.

The computations in a cSN P system are defined as in usual SN P systems: (at most) one rule in each compartment is applied, but the compartments work in parallel, synchronously. The result can be obtained as the number of the spikes in region i_o in the moment when the computation halts, and this can be inside the system or outside, when $i_o = env$. We denote by $N_{in}(\Pi)$ the set of numbers computed (generated) by the system Π in the internal mode. We will not consider here the external output in the form of the number of spikes sent out, as this is a direct dual of the inner mode, but, like in SN P systems, we also consider as the result of a computation the time distance between the first two steps when the system sends spikes out; this can be done by rules introducing couples (a^p, out) used in the skin region of Π, hence in this case the indication of i_o is omitted. We denote by $N_2(\Pi)$ the set of numbers computed by Π in this sense, by means of halting or non-halting computations. (By convention, number 0 is computed by a computation which sends out spikes only once.)

It is important to note that in the previous definition we have imposed no restriction on the number of produced spikes, that is, it can be greater than the number of consumed spikes. Actually, we need rules for producing more spikes than consumed, otherwise we cannot increase the number of spikes in the system – unless if we use the replication target command in_{all}.

We denote by $N_\alpha cSNP_m(forg, here, in_t, in_{all})$, $\alpha \in \{2, in\}$, the family of sets of numbers $N_\alpha(\Pi)$ computed by cSN P systems Π with at most m membranes, using forgetting rules and target indications of the types $here, in_t, in_{all}$, together with indications *in, out*. We explicitly write only *forg* and the indications $here, in_t, in_{all}$ because these features are powerful and they can be avoided in certain cases (this also happens in standard SN P systems with *forg*). When all spiking rules $E/a^c \to u$ have c greater than or equal to the number of spikes in u we write $N_\alpha cSN'P_m(...)$ instead of $N_\alpha cSNP_m(...)$. When the number of membranes is not bounded, we replace the subscript m with $*$.

Here are the results reported in [11]: all the following families

$$N_{in}cSNP_4(here, in_t),\ N_{in}cSNP_7,\ N_2cSNP_4(here, in_t),$$
$$N_{in}cSN'P_*(in_t, in_{all}),\ N_2cSN'P_*(here, in_t, in_{all})$$

are equal to NRE (the family of Turing computable sets of numbers).

Several open problems and research topics were formulated in [11]. First, a large research area is open just by checking whether the results obtained for usual SN P systems can be extended to cSN P systems. Many questions are of interest: looking for small (as the number of membranes) universal cSN P systems, adding anti-spikes, working in a parallel way also in the membranes, working asynchronously, and so on and so forth. In [11] one starts directly with extended systems (without delay). What is the power of non-extended cSN P systems? In this case we need a way to replicate spikes. In [11] in_{all} is used to that aim; what else can be imagined? Look for restrictions which lead to characterizations of sub-universal families of numbers (such as $SLIN$, the family of semilinear sets of numbers) or of languages (in the case

of the external output; note that the spike train can be also a sequence of symbols over an arbitrary alphabet).

The languages generated by cSN P systems were investigated in [10] – many results were obtained, but also many questions remain to be further examined.

3. Numerical P Systems with Migrating Variables

In numerical P systems, the variables are associated with compartments and their values are changed according to the *repartition protocols* of *evolution programs* from variables regions and from the neighboring regions. As in basic classes of transition P systems, we can also move variables across membranes by associating with them the usual target indications *here, in, out.*

Numerical P systems *with migrating variables* were considered in [14] in the following form:

$\Pi = (m, H, \mu, Var, (Pr_1, Var_1(0)), \ldots, (Pr_m, Var_m(0)), (x_{i_0}, j_0))$, where:

- $m \geq 1$ is the number of membranes;
- H is an alphabet (of labels for membranes in μ);
- μ is a hierarchical (cell-like) membrane structure with m membranes labeled with the elements of H;
- $Var = \{x_1, x_2, \ldots, x_n\}$ is a set of variables for the system;
- $Var_i \subset Var, 1 \leq i \leq m$, is a set of variables initially present in region i;
- $Var_i(0)$, $1 \leq i \leq m$, is the vector of the initial values of variables from region i;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i; each program has the following form: $F_{j,i}(x_{p_1}, \ldots, x_{p_k}) \rightarrow c_{j,1}|(x_{r_1}, tar_1) + \ldots + c_{j,q}|(x_{r_q}, tar_q)$, where $F_{j,i}(x_{p_1}, \ldots, x_{p_k})$ is the production function, $c_{j,1}|(x_{r_1}, tar_1) + \ldots + c_{j,q}|(x_{r_q}, tar_q)$ is the repartition protocol of the program and $tar_1, \ldots, tar_q \in \{here, out, in\}$; the symbols *here, out, in* are called *target commands* or *target indications*; all the variables $x_{p_1}, \ldots, x_{p_k}$ and $x_{r_1}, \ldots, x_{r_q}$ are from Var.
- $x_{i_0} \in Var$, $j_0 \in H$.

The variables initially placed in membrane i have non-zero values specified by $Var_i(0), 1 \leq i \leq m$. A variable equal to zero is simply supposed not to be present in a membrane. This is called the NZP assumption. A program $F_{j,i}(x_{p_1}, \ldots, x_{p_k}) \rightarrow c_{j,1}|(x_{r_1}, tar_1) + \ldots + c_{j,q}|(x_{r_q}, tar_q)$ can be applied only when all variables $x_{p_1}, \ldots, x_{p_k}$ ("production variables") are present in membrane i at that time with non-zero values. By using the production function, the system computes a *production value* which is distributed to variables specified by the repartition protocol. Variables involved in the production function are reset to zero after computing the production value.

The application of programs is as usual in numerical P systems, with the following specific points. After the application of the program, the variables involved in the repartition protocol are moved to the region indicated by the target command associated with them. Specifically, *here* means the variable will be placed in the same region i where the program is applied; *out* means the variable will be moved to the region immediately outside membrane i – this region can be the environment in the case when i is the skin membrane; *in* means the variable should be moved to a membrane immediately inside membrane i, non-deterministically chosen.

When a program is applied, for a variable involved in the program there are five cases to consider: i) if the variable appears in the production (it must be present in the membrane for the program to be applied) but not also in the repartition protocol, then this variable is zeroed and removed from the membrane; ii) if the variable appears both in the production function (with a non-zero value) and in the repartition, then it is first zeroed, then the variable with the contribution received from the repartition protocol is moved to the membrane indicated by the associated target; iii) if the variable appears in the repartition protocol and is not present in the membrane (hence it must not appear in the production function), then the variable with its contribution received from the repartition protocol is moved to the membrane as the associated target indicates; iv) if the variable appears in the repartition protocol and it was initially present in the membrane but not in the production, then the initial value plus the contribution it receives is moved to the membrane indicated by its associated target; v) if the variable appears in several repartition protocols, then, in order to avoid any conflicts/complications, we restrict to applying programs where the same variable has associated the same target indication in all programs; then, each program separately changes the variable as stated above and the variable, with the summed value, is moved to the associated target.

After moving variables to the target membranes, all the values of the same variable received from different membranes are added up, and the sum is the value of this variable in the destination membrane. If the sum is zero, then the variable is simply removed from the membrane.

A system can evolve in the *all-parallel* mode (at each step, in each membrane, all programs which can be applied are applied, allowing that several programs share the same variable), in the *sequential* mode (at each step, only one program is applied in each membrane; if more than one program in a membrane can be used, then one of them is non-deterministically chosen), or in the *one-parallel* mode (apply programs in the all-parallel mode with the restriction that one variable can appear in only one of the applied programs; in the case of multiple choices, the programs to be applied are chosen in the non-deterministic way). In the one-parallel mode, where more than one program can be applied in a membrane, one can also impose the restriction that there is no conflict between the targets associated with variables in the repartition protocols of the applied programs.

Besides programs as above (called non-enzymatic), numerical P systems can also have enzymatic programs of the form $F_{j,i}(x_{p_1},\ldots,x_{p_k})|_{e_{j,i}} \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q)$, where $e_{j,i}$ is a variable present in membrane i and different from $x_{p_1},\ldots,x_{p_k}$ and $x_{r_1},\ldots,x_{r_q}$. Such a program is applied at time t only if $e_{j,i}(t) >$

$min(x_{p_1}(t), \ldots, x_{p_k}(t))$. Note that $e_{j,i}(t)$ remains unchanged in the program where it appears as an enzymatic variable; in other programs, $e_{j,i}$ can appear as a usual variable in production functions or repartition protocols, and it can be "consumed" or receive "contributions".

If every program is enzymatic, we call the system *purely enzymatic*.

It is worth mentioning that the enzymatic control was introduced in the framework of using numerical P systems in robot control, see details in [4].

A *configuration* of Π at time $t \in N$ is given by the values of all the variables present in each membrane at that time. Using the programs in the way mentioned above, we obtain *transitions* among configurations. A sequence of such transitions forms a *computation*. If no program can be applied in the current configuration, we say that the system *halts*. When the system halts, the value taken by the special variable x_{i_0} in membrane j_0 is the number generated by the computation.

The set of natural numbers generated by a system Π working in the *one-parallel* or *sequential* mode is denoted by $N_\alpha(\Pi), \alpha \in \{one, seq\}$, where *one* stands for one-parallel, *seq* stands for sequential. We use $N_\alpha M^0 \beta NP_m^D(poly^n(r), Var_{k_1}, Pro_{k_2})$, to denote the family of all sets $N_\alpha(\Pi)$, $\alpha \in \{one, seq\}$, $\beta \in \{E, pE, -\}$, of numbers generated by β numerical P systems Π with migrating variables (E = enzymatic, pE = purely enzymatic; if the system is non-enzymatic, then β is omitted), with at most m membranes, at most k_1 variables, and at most k_2 programs, with production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial; D indicates the use of deterministic systems (we remove it when the systems may also be non-deterministic); the superscript 0 means the system works under the NZP assumption. If this assumption is removed, hence the variables can be present also with value zero, and the programs can be applied if the variables are present in the membrane, does not matter whether or not their values are zero (we say that we work without the NZP assumption), then the superscript 0 is removed. If one of the parameters m, n, r, k_1, k_2 is not bounded, then we replace it with $*$.

Here are part of the results proved in [14]:

1. $N_\alpha M^0 NP_1(poly^1(1), Var_2, Pro_2) - SLIN \neq \emptyset, \alpha \in \{one, seq\}$.
2. $SLIN \subset N_\alpha M^0 NP_1(poly^1(1), Var_*, Pro_*), \alpha \in \{one, seq\}$.
3. The following families are equal to NRE:
   $N_{one} M^0 NP_1(poly^1(3), Var_*, Pro_*)$, $N_{seq} M^0 NP_2(poly^1(3), Var_*, Pro_*)$,
   $N_{one} MENP_1(poly^1(3), Var_*, Pro_*)$, $N_{seq} MENP_2(poly^1(3), Var_*, Pro_*)$.

The possibility to generate strings with these systems was also explored in [14].

The systems considered in [14] work in the one-parallel and sequential mode. The case of all-parallel mode remains open.

4. Numerical P Systems Generating Strings

Instead of considering numbers as results of computations in a numerical P system, we can associate strings, in a way similar to that explored, *e.g.*, for SN P systems, by

considering an *external output*: we add a variable also to the environment, which gets parts of the production of the skin compartment. When this variable gets a value i, we can consider that a symbol b_i was generated and with a halting computation we then associate a string. Several technical details appear in this framework. We recall here the proposals made in [13].

Let us consider a special variable *out* placed in the environment, which can appear in the repartition protocol of programs in the skin region of a numerical P system. At each step its value is first reset to zero, then it receives a new value. If at a given step it receives several values from several programs, all these values are added up and the sum is the value it receives at this step. If the value is a number i between 1 to q, for some constant q, then the symbol b_i is added to the generated string. If at any step variable *out* receives a value which is greater than q or smaller than 0, then this computation aborts, no result is associated with it.

In order to define the generated string, we need to define its end. This is clear in the case when the computation halts (no further program can be applied), and this can be taken as a definition of successful computations in purely enzymatic P systems. In non-enzymatic and in (non-purely) enzymatic systems the computations never halt, and then we define the end of the string by means of a *signal*, e.g., the step when the system sends out value 0. Because for purely enzymatic systems we have halting at our disposal, in this case we avoid sending out value 0, that is, this case is simply ignored. (For a general definition, however, a decision should be made also for value 0 sent out. A possibility is to proceed as in spiking neural P systems, where in such a case a special symbol, b_0, is added to the string.)

It still remains a case not covered: the steps when the system sends no value to variable *out*. We have two choices: to forbid such steps, by the definition of correct computations, or to proceed as in the case of SN P systems and to associate the empty string λ to the generated string (the string is not increased, the system can continue working).

In this way, we define two languages generated by a numerical P system Π. If at each step a positive value is sent to variable *out* (with the exception of the last step, for non-enzymatic and for enzymatic P systems, when value 0 is sent out, marking the end), then we denote the generated language by $L^{res}(\Pi)$ (with res coming from *restricted*). If in the steps when no value is sent out (neither 0) we interpret that the system adds λ to the generated string, then the generated language is denoted by $L^{\lambda}(\Pi)$.

For an easier remembering, we synthesize the previous conventions/definitions in a table:

Sending out	Non-enzymatic & Enzymatic	Purely enzymatic
$1, 2, \ldots, q$	b_i	b_i
< 0 or $> q$	abort	abort
0	end signal	ignored here
nothing	λ or forbidden (res)	λ or forbidden (res)

We denote by $L^{\alpha}\beta NP_m^{\gamma}(poly^n(r), Var_{k_1}, Pro_{k_2})$, $\alpha \in \{res, \lambda\}$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$, the family of languages $L^{\alpha}(\Pi)$, generated by β numerical P systems

Π (E = enzymatic, pE = purely enzymatic; if the system is non-enzymatic, then β is omitted) with at most m membranes, at most k_1 variables, and at most k_2 programs, with production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial; the superscript $\gamma = hal$ is used for purely enzymatic systems, to indicate that the result is obtained when the system reaches a halting configuration; in the case when the end of the computation is defined by means of a signal (sending value 0 out), then we replace *hal* by *fin*. If one of the parameters m, n, r, k_1, k_2 is not bounded, then we replace it with $*$.

Here are some of the results proved in [13]:

1. $L^{res}\beta NP_*^{\gamma}(poly * n(*), Var_*, Pro_*) \subseteq REC$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$.
2. $REG \subseteq L^{res}NP_1^{fin}(poly^1(1), Var_*, Pro_*)$.
3. $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_4) - REG \neq \emptyset$..
4. $L^{res}NP_1^{fin}(poly^1(4), Var_7, Pro_6) - CF \neq \emptyset$.
5. $FIN \subset L^{res}pENP_1^{hal}(poly^1(1), Var_*, Pro_*)$.
6. $REG \subseteq L^{res}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.
7. $L^{res}pENP_1^{hal}(poly^1(1), Var_6, Pro_4) - REG \neq \emptyset$.
8. $L^{res}pENP_1^{hal}(poly^1(1), Var_9, Pro_6) - CF \neq \emptyset$.
9. The families $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_7)$, $L^{res}pENP_1^{hal}(poly^1(2), Var_7, Pro_6)$, and $L^{res}ENP_1^{hal}(poly^1(2), Var_4, Pro_6)$ contain non-semilinear languages.
10. $RE = L^{\lambda}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.

The external variable can be useful also for considering a numerical P system as a decidability device: an instance of a decision problem is encoded in the values of certain variables, and the values of a specified variable – maybe the external one (which is not used in any production function) – at a well defined moment (in a halting configuration, if halting can be defined and ensured) is the yes/no answer to the problem instance. Using numerical P systems in this way, as decidability devices, is a general research topic of definite interest. What is the efficiency of this approach? Can **NP**-complete problems be solved in polynomial time in this framework? If not, which further ingredients (a candidate is membrane division) can help?

5. SN P Systems with Polarizations

We end with a very promising new class of SN P systems, which are no longer using regular expressions for controlling the application of rules, but instead *polarizations* are associated with the neurons and the rules. The idea was explored in [9]. For the convenience of readers, we recall the definition with full details.

A *spiking neural P systems with polarizations* (in short, a PSN P system) of degree $m \geq 1$ 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. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (\alpha_i, n_i, R_i), 1 \leq i \leq m$, where:
   (a) $\alpha_i \in \{+, 0, -\}$ is the *initial polarization* of neuron σ_i;
   (b) n_i is the *initial number of spikes* contained in σ_i;
   (c) R_i is a finite set of *rules* of the following two forms:
      (i) $\alpha/a^c \to a; \beta$, for $\alpha, \beta \in \{+, 0, -\}, c \geq 1$ (*spiking rules*);
      (ii) $\alpha/a^s \to \lambda; \beta$, for $\alpha, \beta \in \{+, 0, -\}, s \geq 1$ (*forgetting rules*);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn, 1 \leq i, j \leq m$ (synapses between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

Note that the definition of PSN P systems is the same as the usual definition of SN P systems given in the literature, with two differences: the applicability of a rule is not determined by checking the total number of spikes contained in the neuron against a regular expression associated with the rule, but the neurons have charges and a rule can be applied only if the neuron has the charge indicated in the left hand side of the rule. Of course, in order to use a rule, the total number of spikes inside the neuron should not be less than the number of spikes consumed by the rule. Moreover, the neurons not only send out spikes, but also charges, even when using forgetting rules.

A spiking rule $\alpha/a^c \to a; \beta$ is used as follows. If the neuron σ_i has the charge α and it contains at least c spikes, then the rule can be applied, and its application means that c spikes are consumed, the neuron fires and produces a spike, which carries the charge β. The spike is replicated and each neuron σ_j such that $(i, j) \in syn$ receives the spike and the charge β.

The output neuron also sends spikes out of the system, but no electrical charge is sent out (it is "lost" in the environment).

A forgetting rule $\alpha/a^s \to \lambda; \beta$ is applied when the neuron has the charge α and contains at least s spikes; s spikes are removed from the neuron and the charge β is sent to all neurons σ_j such that $(i, j) \in syn$. (Note that we do not necessarily forget all spikes, as in the case of usual SN P systems, where exactly s spikes should be present in order to use a forgetting rule $a^s \to \lambda$.)

After a neuron receives charges from other neurons, we perform a computation of charges inside the neuron as described below:

1. several positive charges (+), several neutral charges (0), several negative charges (−) lead to one positive charge (+), one neutral charge (0), one negative charge (−), respectively;

2. a positive (+) and a negative charge (−) cancel each other and give the neutral charge (0);

3. a positive (+) or a negative charge (−) is not changed by a neutral charge (0).

We stress that (i) the computation of charges takes no time; (ii) step 1 of the above computation of charges is done first. For example, if a given neuron which is initially neutral receives two positive charges and one negative charge, then first the two positive charges lead to one positive charge, after that the positive charge and the negative charge cancel each other, thus the neuron remains neutral.

As usual in SN P systems, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. If several rules can be used at the same time in a neuron, then the one to be applied is chosen non-deterministically. Thus, the rules are used in a sequential manner in each neuron, but the neurons function in parallel with each other.

The *configuration* of the system is described by both the number of spikes and the charge of each neuron; thus, the *initial configuration* of the system is $C_0 = \langle n_1, n_2, \ldots, n_m; \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$. Using the rules as described above, one can define *transitions* among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Rightarrow C_2$. Any sequence of transitions starting from the initial configuration is called a *computation*. A computation is *successful* if it reaches a configuration where no rule can be used in any neuron of the system. We say that the computation is *halting*.

A PSN P system can be used as a generative, an accepting, or a computing device.

With any computation, halting or not, we associate a *spike train*, the sequence of symbols 0 and 1 describing the behavior of the output neuron: 1 indicates a spiking step, 0 indicates a step when no spike exits the system. With a spike train, a *result of a computation* can be defined in several ways. For instance, the result of a computation can be defined as usual in general SN P systems: we only consider the first two time instances t_1 and t_2 that neuron *out* spikes and we say that the number $t_2 - t_1$ is computed/generated by Π. The set of all numbers generated in this way by a PSN P system Π is denoted by $N_2(\Pi)$.

In the generative case, the neuron with label *in* is ignored. In the accepting mode, the neuron with label *out* is ignored. A number n is introduced in the system, by introducing a sequence $10^{n-1}1$ in neuron *in* (two spikes are introduced, at a time distance of n steps) and this number is accepted if the computation halts.

When both an input and an output neuron are considered, the PSN P systems can be used to compute numerical functions. In order to compute a function $f : \mathbf{N}^k \to \mathbf{N}$, k natural numbers $n_1, \ldots, n_k$ are introduced into the system by "reading" from the environment a spike train of the form $z = 10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$. Note that exactly $k + 1$ spikes are "read", that is, after the last spike, it is assumed that no further spike is sent to the input neuron. The result of the computation is also encoded as the distance between the first two spikes emitted by the output neuron with the restriction that the system outputs exactly two spikes and halts (maybe some further steps after the second spike), hence it produces a spike train of the form $0^b10^{r-1}10^d$

for some $b, d \geq 0$ with $r = f(n_1, n_2, \ldots, n_k)$. The system outputs no spike in the $b \geq 0$ steps from the beginning of the computation until the first spike.

We denote by $N_2PSNP(ch_p)$ the family of all sets of numbers $N_2(\Pi)$ generated by PSN P systems with at most p charges.

The two results proved in [9] are the following:

1. $NRE = N_2PSNP(ch_3)$.
2. There exists a universal PSN P system (with three charges) for computing functions, having 164 neurons.

The proofs are rather complicated, at least in comparison with the proofs of the corresponding results for usual SN P systems, and this is due to the fact that the polarizations provide a much weaker control on the applicability of the rules in neurons than the regular expressions.

Again, many research topics remain to be explored. Practically the whole program of investigation carried on usual SN P systems has to be explored also for the new type of spiking neural P systems: normal forms (can we get rid of forgetting rules?), using extended rules (producing more than one spike can help, e.g., in simplifying the proofs?), generating strings or infinite sequences, considering asynchronous computations or a parallel/exhaustive use of spiking rules in each neuron, adding astrocytes or other biology inspired ingredients, and so on and so forth. Another idea is to consider cell-like PSN P systems; polarized cell-like SN P systems seem to be challenging to investigate (maybe not universal).

There also appear specific open problems. Of a definite interest is the question whether or not the number of electrical charges considered in the universality proof from [9], three, can be decreased. What is the power of PSN P systems with 2 charges, or even without any charge? Is any of the corresponding classes of computing devices sub-universal? If so, what are the properties (size, closure, decidability) of the corresponding family of sets of numbers or of languages generated? Finally: can the number of neurons in universal PSN P systems be (significantly) decreased? (We are pessimistic about this, as clever codifications in terms of the number of spikes, as usual for standard SN P systems, do not seem to help in the absence of regular expressions.)

Definitely, we believe that SN P systems with polarizations deserve further research efforts.

Acknowledgments. This work of L. Pan, T. Wu and Z. Zhang was supported by the National Natural Science Foundation of China (61033003, 91130034, and 61320106005), Ph.D. Programs Foundation of Ministry of Education of China (2012014213008), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

[1] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*, Springer-Verlag, Berlin, 2006.

[2] P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing in Systems and Synthetic Biology*, Springer-Verlag, Berlin, 2014.

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

[4] A.B. Pavel, C. Buiu, C.: Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, 11, 3 (2012), 387–393.

[5] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).

[6] Gh. Păun: Tracing some open problems in membrane computing, *ROMJIST*, 10, 4 (2007), 303–314.

[7] Gh. Păun, R. Păun: Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, 73 (2006), 213–227.

[8] Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[9] T. Wu, A. Păun, Z. Zhang, L. Pan: Spiking neural P systems with polarizations. Submitted, 2015.

[10] T. Wu, Z. Zhang, L. Pan: On string languages generated by cell-like spiking neural P systems. Submitted, 2015.

[11] T. Wu, Z. Zhang, Gh. Păun, L. Pan: Cell-like spiking neural P systems. *Theoretical Computer Science*, 2016, DOI: 10.1016/j.tcs.2015.12.038.

[12] G. Zhang, J. Cheng, T. Wang, X. Wang, J. Zhu: *Membrane Computing. Theory and Applications*, Science Press, Beijing, 2015 (in Chinese).

[13] Z. Zhang, T. Wu, L. Pan, Gh. Păun: On string languages generated by numerical P systems. Submitted, 2015.

[14] Z. Zhang, T. Wu, A. Păun, L. Pan: Numerical P systems with migrating variables. Submitted, 2015.