Open Problems, Research Topics, Recent Results on Numerical and Spiking Neural P Systems (The "Curtea de Argeş 2015 Series")

Gheorghe Păun[1], Tingfang Wu[2], Zhiqiang Zhang[2]

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

[2] Key Laboratory of Image Information Processing and Intelligent Control
School of Automation, Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
whutwutf@163.com, zhiqiangzhang@hust.edu.cn

Summary. A series of open problems and research topics are formulated, about numerical and spiking neural P systems, initially prepared as a working material for a three months research stage of the second and the third co-author in Curtea de Argeş, Romania, in the fall of 2015. Further problems were added during this period, while certain problems were addressed in this time; some details and references are provided for such cases.

1 Introduction

In membrane computing there are numerous open problems and research topics in circulation, many of them also collected in systematic lists, compiled, for instance, for the yearly Brainstorming Week on Membrane Computing, see http://ppage.psystems.eu and www.gcn.us.es. Because the present list had initially a working material character, we do not provide here complete references. Furthermore, as the reader is supposed to be a researcher in membrane computing, we do not give basic definitions either (but we give details for the new classes of P systems considered). As a general reference we refer to the Handbook [8].

The list deals only with numerical P systems (in short, NP systems, [7]) and spiking neural P systems (in short, SNP systems, [3]). Some problems are more specific, many others are just general ideas, so that the first step in approaching them implies a formal definition (possibly, of new classes of NP or SNP systems). We recall some details from several papers written in Curtea de Argeş during the mentioned period of time.

2 Bridging NP and SNP

The two "exotic" classes of P systems (their "biochemistry" is not very closely related to the biological one) have many common and, also, many different characteristics. Of the first type is the fact that both of them process numbers and a specific "production" of a cell is distributed among neighboring cells. Thus, it is just natural to check whether features of one class can be extended to the other class, and conversely. This proves to be a very fruitful idea.

Here are a few more precise suggestions of this kind.

A. NP with SNP features. Three main ingredients of SNP systems can be exported also to NP systems: the tissue-like arrangement of membranes, the regular expressions which control the application of spiking rules, and the replication of the production to all the adjacent cells. However, further features can be considered – only a few suggested below, so this is already a general research issue.

B. Passing from the cell-like membrane structure of usual NP systems to tissue-like NP systems is just an extension which will also extend all computing power results and computational complexity properties. As the distribution of the *production* of a compartment is done with a precise identification of the target variables, a simple (real-time?) mutual simulation between cell-like and tissue-like NP systems is expected.

C. What about using, in the tissue-like NP systems, weights on "synapses" instead of repartition coefficients? The simulation of NP systems with weights by systems with repartition coefficients and conversely is a natural research topic. What about using both repartition coefficients and weights on "synapses"? (This might also have economic interpretations, thus bringing the model closer to the initial motivation, the economics.)

D. Associating a regular expression with each evolution program seems to be a non-trivial extension of the enzymatic control from [6]. (Note that we can compare the values of two variables, even in terms of regular expressions, in the following way: $(x_1x_2)^*x_2^*$ can be interpreted as $x_2(t) \geq x_1(t)$, meaning that the value of x_2 at time t is greater than or equal to the value of x_1 at time t.) In what cases can this intuition be confirmed? A good candidate is the descriptional complexity of various NP systems, for instance, constructed for controlling robots.

E. While considering a regular expression looks like adding computing power, distributing the production of a cell to all neighboring variables, replicated, on the one hand, increases the total values of variables in the system, on the other hand, removes the control possibilities provided by the repartition coefficients. Which is the power (and the efficiency) of NP systems with such a repartition protocol remains to be checked.

F. Directly related to the previous idea is the following problem: what about NP systems with an "egalitarian" repartition, i.e., with all distribution coefficients equal in each evolution program/in each compartment/in the whole system? A

particular case is that of considering all distribution coefficients equal to 1. Are such restricted NP systems still universal?

G. Continuing with the restrictions, what about considering NP systems with only $k \geq 1$ variables in the distribution protocols? Is any difference in power between NP systems with k variables and those with $k+1$ variables, for various values of k? (It is expected that $k = 1$ is, indeed, a special case.) What about NP systems with both egalitarian distribution and at most k variables in each distribution protocol?

H. SNP with NP features. This is the "reverse" of problem **A**, again with (at least) three basic directions: considering SNP systems with a cell-like membrane structure, using a production function instead of spiking rules, and considering a distribution protocol for communicating the produced spikes. What else, it remains to imagine.

I. SNP systems with a cell-like membrane structure look "non-natural" from a biological point of view, but it is mathematically interesting, especially in view of the children-parent membrane interaction; remember that circularity is not allowed in standard SNP systems.

This idea was explored in [14]; 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 also 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 distance in time 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 [14]:

1. $N_{in}cSNP_4(here, in_t) = NRE$.
2. $N_{in}cSNP_7(in_t) = NRE$.
3. $N_{in}cSNP_7 = NRE$.
4. $N_{in}cSN'P_*(in_t, in_{all}) = NRE$.
5. $N_2cSNP_4(here, in_t) = NRE$.
6. $N_2cSN'P_*(here, in_t, in_{all}) = NRE$.

Several open problems and research topics were formulated in [14].

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 [14] one starts directly with extended systems (without delay). Which is the power of non-extended cSN P systems? In this case we need a way to replicate spikes. In [14] in_{all} it 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 $NREG$) 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 cell-like SN P systems were investigated in [13] – many results were obtained, but also many questions remain to be further examined. We do not enter into details.

J. Replacing the spiking rules with a production function (of one variable if only spikes are considered, of two variables if also anti-spikes are used; the interplay

with the annihilation rule is also of interest – useful seems to be to apply the annihilation rule after computing the production, of both spikes and anti-spikes). The production function can be a polynomial, as in usual NP systems, but we can try to capture also other neural ingredients, such as the sigmoid function on which the functioning of the biological neuron is based.

K. Using a distribution protocol, for SNP systems with "standard" spiking rules looks easy: just associate distribution coefficients to synapses. This add "programming" possibilities, hence simpler proofs than for usual SNP systems are expected.

3 Further Problems for NP

The investigations on NP systems reported so far only deal with the basic systems and with the enzymatic ones, but there are many possibilities for considering further classes. Some ideas were mentioned also before, a few others will be suggested below, but the reader can imagine many more.

L. For instance, we can consider NP systems with restricted communication, in the sense that the production of a compartment is distributed only to variables from one or two levels out of the three used so far: *here, down, up*. For "one-way" systems it is expected to obtain rather restricted families of numbers generated in this way. Which cases still lead to universality?

M. A very natural idea is, instead of having the variables associated with compartments, to move variables across variables by associating with them the usual target indications *here, in, out*.

Numerical P systems *with migrating variables* (in short, MNP systems) were considered in [17] 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 from Var, initially present in region i;
- $Var_i(0), 1 \leq i \leq m$, is a vector which indicates the values of the initial variables in region i;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i; each program has the following form:

$$F_{j,i}(x_{p_1},\ldots,x_{p_k}) \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q),$$

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 $F_{j,i}(x_{p_1},\ldots,x_{p_k}) \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q)$ 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. An important observation is that 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) and 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. (Another possibility is to immediately move variables with the value received from each program to the associated targets and to sum the values at the destination.)

MNP systems 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 apply 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 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*.

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 [17]:

1. $N_\alpha M^0 NP_1(poly^1(1), Var_2, Pro_2) - SLIN_1^+ \neq \emptyset, \alpha \in \{one, seq\}$.
2. $SLIN_1^+ \subset N_\alpha M^0 NP_1(poly^1(1), Var_*, Pro_*), \alpha \in \{one, seq\}$.
3. $N_{one} M^0 NP_1(poly^1(3), Var_*, Pro_*) = NRE$.
4. $N_{seq} M^0 NP_2(poly^1(3), Var_*, Pro_*) = NRE$.
5. $N_{one} MENP_1(poly^1(3), Var_*, Pro_*) = NRE$.
6. $N_{seq} MENP_2(poly^1(3), Var_*, Pro_*) = NRE$.

Also the possibility to generate strings with these systems was explored in [17].

N. Associate a language to a computation in an NP system. For instance, the values of a variable can form a string – in general, over an infinite alphabet (like in [2]), or on a finite alphabet. For instance, we can consider the binary string obtained by marking with 0 and 1 the odd and the even values of the distinguished variable. We can also "read" the natural numbers modulo a given constant $k \geq 2$, so that we can obtain strings over an alphabet with k letters.

O. In particular, we can associate a language to an NP system by considering an *external output*: we add a variable also to the environment, which gets parts of the production of the skin compartment. This can be used both for computing numbers and strings (in the latter case, following the suggestions from the previous question).

NP systems used as string generators were considered in [16]. A string is associated with a computation in a way somewhat similar to that adopted for spiking neural P systems: one just considers a special variable *out* 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 one 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 spiking neural P systems and to associate the 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 [16]:

1. $L^{res}\beta NP_*^{\gamma}(poly * n(*), Var_*, Pro_*) \subseteq REC$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$.
2. $L^{res}NP_1^{fin}(poly^1(1), Var_1, Pro_2) - FIN \neq \emptyset$.
3. $REG \subseteq L^{res}NP_1^{fin}(poly^1(1), Var_*, Pro_*)$.
4. $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_4) - REG \neq \emptyset$..
5. $L^{res}NP_1^{fin}(poly^1(4), Var_7, Pro_6) - CF \neq \emptyset$.
6. The family $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_7)$ contain non-semilinear languages.
7. $L^{res}pENP_1^{hal}(poly^1(1), Var_2, Pro_2) - FIN \neq \emptyset$.
8. $FIN \subset L^{res}pENP_1^{hal}(poly^1(1), Var_*, Pro_*)$.
9. $REG \subseteq L^{res}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.
10. $L^{res}pENP_1^{hal}(poly^1(1), Var_6, Pro_4) - REG \neq \emptyset$.

11. $L^{res}pENP_1^{hal}(poly^1(1), Var_9, Pro_6) - CF \neq \emptyset$.
12. The family $L^{res}pENP_1^{hal}(poly^1(1), Var_7, Pro_6)$ contains non-semilinear languages.
13. $RE = L^{\lambda}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.
14. The family $L^{res}ENP_1^{hal}(poly^1(2), Var_4, Pro_6)$ contains non-semilinear languages.

P. The external variable can be useful also for considering an NP 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 NP systems in this way, as decidability devices, is a general research topic of definite interest. Which is the efficiency of this approach? Can **NP**-complete problems be solved in polynomial time in this framework? If not, which ingredients can help?

Q. In general, what about NP systems with "active membranes", i.e., with possibilities of dissolving, creating, dividing membranes? Are these operations useful for speeding-up the computations?

R. Related also to the previous questions is the natural one of looking for interesting sequences of numbers and for interesting functions which can be computed by NP systems. Are there *hard* sequences/functions (hard with respect to Turing machines) which can be computed in a more efficient way with NP systems (maybe endowed with membrane manipulating rules)?

S. The answer to the previous question can have a practical interest, e.g., for robot controllers. In this context, an exercise is natural: passing from robots acting in a 2D space, as those considered so far in membrane computing area, to 3D robots (drones, satellites). This is, expectedly, only a programming issue/exercise, but of interest in view of the popularity of 3D machineries which need an automatic (maybe intelligent) controller.

T. In robot control there were useful numerical P systems with *enzymes* controlling the applicability of programs. A natural idea is to count the variables used as enzymes, then to try to keep this number as small as possible without diminishing the computing power (without losing the universality). The numbers of enzymes used so far in proofs is surprisingly large: For instance, the result in [11] can be written as

$$\begin{aligned} NRE &= N_{gen}E_*NP_*(poly^1(2), oneP) \\ &= N_{gen}E_{776}NP_{254}(poly^2(253), allP) \end{aligned}$$

(the subscript of E indicates the number of enzymes used) whereas the improvement of the last equality given in [10] can be written as

$$NRE = N_{gen}E_{427}NP_4(poly^1(6), allP).$$

The improvements of the above results in [4] are also not concerned with keeping under control the number of variables used as enzymes.

In [18], the following – again surprising – results were obtained:

$$\begin{aligned} NRE &= N_{acc}E_1NP_1(poly^1(2), allP) \\ &= N_{gen}E_2NP_1(poly^1(2), oneP) \\ &= N_{acc}E_1NP_1(poly^1(2), oneP). \end{aligned}$$

What other results about enzymatic numerical P systems remain to be improved from this point of view? (What about small universal numerical P systems?)

U. The previous problem is related to another way to control the use of programs, namely by means of *thresholds*, constants associated with programs, compared with the current values of variables in the production function or with the value of the production itself. See precise definitions in [19] and [15]. Universality results with a small number of thresholds are obtained in these papers.

4 Further Problems for SNP

V. In the same way as NP systems can compute (also for robot controllers) functions $f : \mathbf{N}^n \longrightarrow \mathbf{N}^m$, such a function can be computed also by SNP systems. Can such systems be used for designing robot controllers? Which is the (practical and theoretical) efficiency of such an approach?

W. On the one hand, the brain is supposed to be a non-Turing "computer", on the other hand, it is supposed to have a deterministic conscious part and a non-deterministic unconscious part, the first one problems problems to the latter, this one proposing solutions, which are evaluated by the conscious part, and the process is iterated until either finding the right solution, or the problem is abandoned. Can such a strategy be implemented in terms of SNP systems? Is it possible to devise such a hybrid SNP system able of computing beyond Turing?

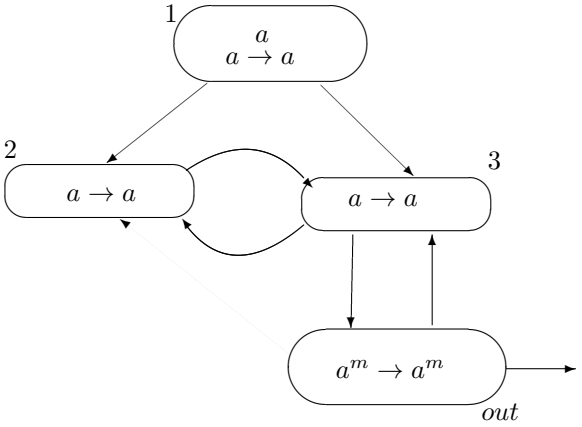
X. What about extending to SNP systems other ideas currently explored in hypercomputing, see, e.g., [9]? Can they be formulated for SNP systems in such a way to make them compute beyond Turing (as – again – the human brain is supposed to do)?

For instance, what about *accelerated SNP systems*, where the neurons "learn" during the functioning of the system. First time when a neuron uses a rule, the application of the rule lasts one time unit (the time is measured by an external clock, the *user clock*). Next time (does not matter how many steps the neuron is not working in between), the rule is applied in half of a user time unit – and so on, always half of the duration of the previous rule application.

Thus, a neuron which works each step, in two external time units will perform an infinity of steps.

The example in figure below shows an SN P system with only one spike inside, with neurons 1 and 4 working only once, but with neurons 2 and 3 working each step from step 2 on. Thus, in at most 2 external time units, neurons 2 and 3 send to neuron 4 any number of spikes. When m spikes are present in neuron 4, neuron 4 fires and the computation halts.

Thus, irrespective how large is m, starting with only one spike inside, this system will produce m spikes (sent outside) in at most 4 external time units (but working internally a number of steps which depends on m).



Conjectures:

1. Using the acceleration, we can solve **NP**-complete problems in polynomial time. (The first step is to find a suitable problem to be addressed in this framework.)
2. Accelerated P systems can go beyond Turing (can solve the halting problem – see the example of [1]); can this result be extended to SN P systems?

Both these conjectures are, metaphorically, supported by the fact that the brain is efficient and "non-Turing".

Y. Add to SNP systems further biologically-inspired features, to get closer to the brain. Ideally, bring enough further features to the SNP systems so that processes taking place in the "real" brain can be modeled/simulated (at the level of biologists interest).

Z. We have left to the end a very promising new class of SNP systems, which are no longer using regular expressions for controlling the application of spiking rules, but instead *polarizations* are associated with the neurons and the rules. The idea was explored in [12]. For the reader convenience, 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

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out),$$

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 charge (+) and a negative charge (−) cancel each other and give the neutral charge (0);
3. a positive charge (+) 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 $N_2(\Pi)$ (the subscript 2 indicates that the computation result is encoded by the time distance between the first two spikes of any computation).

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 $N_2PSNP(ch_p)$ 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 [12] 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 complex, 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 [12], three, can be decreased. Which 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, which 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 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. C. Calude, Gh. Păun: Bio-steps beyond Turing, *CDMTCS Research Report* 226, Univ. of Auckland (November 2003), and *BioSystems*, 77 (2004), 175–194
2. J. Dassow, G. Vaszil: P finite automata and regular languages over countable infinite alphabets. *Proc. WMC 2006, Leiden, The Netherlands* (H.J. Hoogeboom et al., eds.), LNCS 4361, Springer, 2006, 367–381.
3. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
4. A. Leporati, A.E. Porreca, C. Zandron, G. Mauri: Improving universality results on parallel enzymatic numerical P systems. *Proceedings of 11th Brainstorming Week on Membrane Computing*, Sevilla, February 2013, Fenix Editora, Sevilla, 2013, 177–200.
5. A. Leporati, A.E. Porreca, C. Zandron, G. Mauri: Enzymatic numerical P systems using elementary arithmetic operations. *Membrane Computing. Proc. 14th Intern. Conf., CMC2013, Chişinău, August 2013*, LNCS 8340 (A. Alhazov et al., eds.), Springer, Berlin, 2014, 249–264.
6. A.B. Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. *Proc. Conf. Living Machines 2012*, LNCS 7375, Springer, 2012, 204–215.
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. A. Syropoulos: *Hypercomputation: Computing Beyond the Church-Turing Barrier*. Springer, Berlin, 2008.
10. C.I. Vasile, A.B. Pavel, I. Dumitrache: Universality of enzymatic numerical P systems. *International Journal of Computer Mathematics*, 90(4) (2013), 869–879.
11. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the power of enzymatic numerical P systems. *Acta Informatica*, 49(6) (2012), 395–412.
12. T. Wu, A. Păun, Z. Zhang, L. Pan: Spiking neural P systems with polarizations. Submitted, 2015.
13. T. Wu, Z. Zhang, L. Pan: On string languages generated by cell-like spiking neural P systems. Submitted, 2015.
14. T. Wu, Z. Zhang, Gh. Păun, L. Pan: Cell-like spiking neural P systems. Submitted, 2015.
15. Z. Zhang, L. Pan: Numerical P systems with production thresholds. Submitted, 2015.
16. Z. Zhang, T. Wu, L. Pan, Gh. Păun: On string languages generated by numerical P systems. Submitted, 2015.
17. Z. Zhang, T. Wu, A. Păun, L. Pan: Numerical P systems with migrating variables. Submitted, 2015.
18. Z. Zhang, T. Wu, A. Păun, L. Pan: Universal enzymatic numerical P systems with a small number of enzymatic variables. Submitted, 2015.
19. Z. Zhang, J. Xu, L. Pan: Numerical P systems with thresholds. Submitted, 2015.