New Normal Forms for Spiking Neural P Systems

Linqiang Pan[1,2], Gheorghe Păun[2,3]

[1] Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, lqpan@us.es
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro, gpaun@us.es

Summary. We consider a natural restriction in the architecture of a spiking neural P system, namely, to have neurons of a small number of types (i.e., using a small number of sets of rules), and we prove that three types of neurons are sufficient in order to generate each recursively enumerable set of numbers as the distance between the first two spikes emitted by the system or as the number of spikes in a specified neuron, in the halting configuration. The case we investigate is that of spiking neural P systems with standard rules, with delays, but without using forgetting rules; similar normal forms remain to be found for other types of systems.

1 Introduction

The spiking neural P systems (in short, SN P systems) were introduced in [3], and then investigated in a large number of papers. We refer to the respective chapter of [4] for general information in this area, and to the membrane computing website from [5] for details.

In this note, the SN P systems are considered as generators of sets of numbers, with the numbers obtained as the distance in time between the first two spikes emitted by the output neuron of the system, or with the generated number given as the number of spikes present in a given neuron in the end of the computation. Systems with standard rules are used (i.e., with only one spike produced by each rule), with the rules used sequentially in each neuron, but the whole system working in the maximally parallel way (i.e., each neuron which can use a rule has to do it).

Several normal forms were imposed to SN P systems – see, e.g., [2]. A natural restriction suggested by biology but also natural by itself is to restrict the number

of types of neurons used in a system, where by "type" we understand the set of rules present in a neuron. An SN P system whose neurons are of at most k types is said to be in the kR-normal form.

We prove that each recursively enumerable set of natural numbers can be generated by an SN P system (without forgetting rules, but using delays) in the $3R$-normal form when the number is obtained as the distance between the first two spikes sent out by the system or when the number is given by the number of spikes from a specified neuron. Slightly bigger values are obtained when we also consider the number of spikes initially present in a neuron for defining the "type" of a neuron. We do not know whether or not these results can be improved, or how they extend to other classes of SN P systems. (Do the forgetting rules help? What about extended rules, about asynchronous SN P systems, systems with exhaustive use of rules, etc?) What about SN P systems used in the accepting mode? (The systems can then be deterministic, which usually brings some simplifications.)

2 Prerequisites

We assume the reader to be familiar with basic elements about SN P systems, e.g., from [4] and [5], and we introduce here only a few notations, as well as the notion of register machines, used later in the proofs of our results. We also assume familiarity with very basic elements of automata and language theory, as available in many monographs.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The family of Turing computable sets of natural numbers is denoted by NRE. For a regular expression E we denote by $L(E)$ the regular language identified by E.

A *register machine* is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), 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 non-empty, 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 computes (generates) a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach 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 $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents. In the proofs of our results we assume that the register machines which we simulate have these properties.

We can also use a register machine in the accepting mode: a number is stored in the first register (all other registers are empty); if the computation starting in this configuration eventually halts, then the number is accepted. Again, all sets of numbers in NRE can be obtained, even using deterministic register machines, i.e., with the ADD instructions of the form $l_i : (\texttt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$).

Convention: when evaluating or comparing the power of two number generating/accepting devices, number zero is ignored.

3 Spiking Neural P Systems

In order to have the paper self-contained, we recall here the definition of an SN P system and of the set of numbers generated or accepted by it.

An SN P system of degree $m \geq 1$ is a construct of the form

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

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \rightarrow a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
      (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means removing c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost). In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$).

The rules of type (2) are *forgetting* rules and they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E/a^c \to a; d$ of type (1) has $E = a^c$, then we will write it in the following simplified form: $a^c \to a; d$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. Since two firing rules, $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case, only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron. During a computation, the "state" of the system is described by both by the number of spikes present in each neuron, and by the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

An SN P system can be used in various ways. In the generative mode, we start from the initial configuration and we define the result of a computation (i) either as the number of steps between the first two spikes sent out by the output neuron, or (ii) as the number of spikes present in neuron σ_{out} when the computation halts (note that in the first case we do not request that the computation halts after sending out two spikes). We denote by $N_2(\Pi)$ the set of numbers computed in the first way by an SN P system Π and by $N_{gen}(\Pi)$ the set of numbers generated

by Π in the second case. We can also use Π in the accepting mode: a number n is introduced in the system in the form of a number $f(n)$ of spikes placed in neuron σ_{in}, for a well-specified mapping f, and the number n is accepted if and only if the computation halts. (Alternatively, we can introduce the number to be recognized as the distance in time between two spikes entering neuron σ_{in} from the environment.) We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π.

In the generative case, the neuron (with label) *in* is ignored, in the accepting mode the neuron *out* is ignored (in most cases below, we identify the neuron σ_i with its label i, so we say "neuron i" understanding that we speak about "neuron σ_i"). We can also use an SN P system in the computing mode, introducing a number in neuron *in* and obtaining a result in neuron *out*, but we do not consider this case here.

A neuron σ_i (in the initial configuration of an SN P system) is characterized by n_i, the number of spikes present in it, and by R_i, its associated set of rules. An SN P system is said to be in the kR-normal form, for some $k \geq 1$, if there are at most k different sets $R_1, \ldots, R_k$ of rules used in the m neurons of the system. An SN P system is said to be in the knR-normal form, for some $k \geq 1$, if there are at most k different pairs $(n_1, R_1), \ldots, (n_k, R_k)$ describing the m neurons of the system.

We denote by $N_\alpha SNP(k\beta, forg, dley)$ the families of all sets $N_\alpha(\Pi)$ computed by SN P systems in the $k\beta$-normal form, for $\alpha \in \{2, gen, acc\}$, $\beta \in \{R, nR\}$, and $k \geq 1$; if no forgetting rules are used, then we remove the indication *forg* from the notation; if all rules have delay $d = 0$, then we remove the indication *dley* from the notation.

4 A 3*R*-Normal Form Result

We are going now to prove the main result mentioned in the Introduction: SN P systems with only three different sets of rules are universal when generating numbers encoded in the first two spikes of the spike train.

Theorem 1. $NRE = N_2SNP(3R, dley)$.

Proof. We show that $NRE \subseteq N_2SNP(3R, dley)$; the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). Let us consider a register machine $M = (m, H, l_0, l_h, I)$ with the properties specified in Section 2. We construct an SN P system Π which simulates M in the way somewhat standard already when proving that a class of SN P systems is universal. Specifically, we construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN which provides the result (in the form of a suitable spike train). Each register r of M will have a neuron r in Π, and if the register contains the number n, then the associated neuron will contain $2n$ spikes.

The modules will be given in a graphical form, indicating their initial configuration, the synapses, and, for each neuron, the associated set of rules; all neurons

are initially empty, with the exception of the neuron associated with the initial label, l_0, of M, which contains one spike, and with exception of a few other neurons, as shown in the following figures.

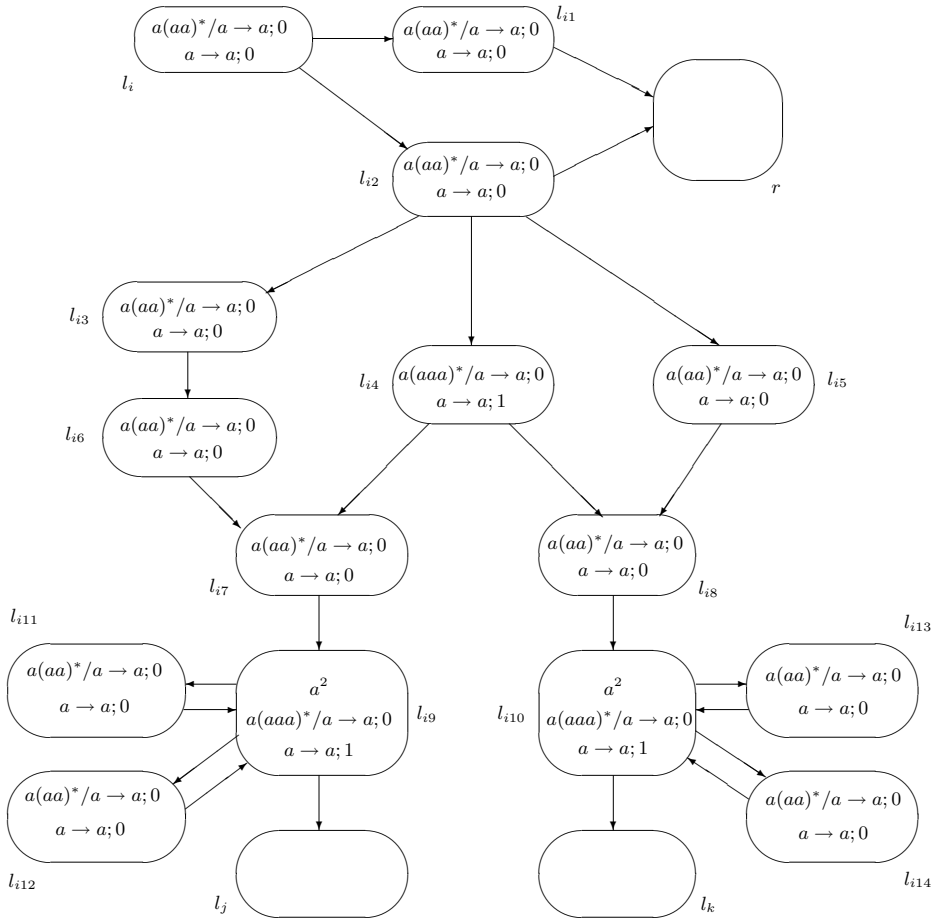

Fig. 1. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

We consider the following three sets of rules:

$$R_1 = \{a(aa)^*/a \to a; 0, \quad a \to a; 0\},$$
$$R_2 = \{a(aa)^*/a^3 \to a; 0, \quad a \to a; 1\},$$
$$R_3 = \{a(aaa)^*/a \to a; 0, \quad a \to a; 1\}.$$

The *ADD* module used to simulate an addition instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is indicated in Figure 1. No rule in R_1 can be applied in the presence of an even number of spikes. If a spike enters the neuron (with the label) l_i, then this neuron starts using its rules; initially, this is the case with neuron l_0. Neuron l_i spikes and one spike is sent to both neuron l_{i1} and neuron l_{i2}, which also spike in the next step. In this way, two spikes are sent to neuron r, and this represents the increment of register r by one. Neuron l_{i2} also sends a spike to neurons l_{i3}, l_{i4}, and l_{i5}. Neurons l_{i3} and l_{i5} spike immediately, while neuron l_{i4} can non-deterministically choose either rule to use as both of them are enabled by the existence of a single spike – this ensures the non-deterministic passage to one of the instructions l_j or l_k..

Assume that $\sigma_{l_{i4}}$ uses the rule $a(aa)^*/a \to a; 0$. This means that in the next step $\sigma_{l_{i8}}$ receives two spikes, hence no rule here can be used. Simultaneously, neurons l_{i6} and l_{i7} receive one spike each, and both of them spike. In this way, $\sigma_{l_{i9}}$ receives one spike and $\sigma_{l_{i7}}$ continues having one spike. Neuron l_{i9} contains now a number of spikes of the form $3n + 3$, for some $n \geq 0$ (initially we had two spikes here, hence $n = 0$) and no rule is enabled. In the next step, this neuron receives one further spike, and the first rule is fired (the number of spikes is now $3(n + 1) + 1$). All neurons l_j and l_{i11}, l_{i12} receive one spike. The last two neurons send back to $\sigma_{l_{i9}}$ one spike each, hence the number of spikes in this neuron will be again congruent with 2 modulo 3, as at the beginning. Thus, the neuron associated with the label l_j has been activated.

If neuron l_{i4} uses the rule $a \to a; 1$, then $\sigma_{l_{i7}}$ receives two spikes at the same (after one time unit) time and this branch remains idle, while neurons $l_{i8}, l_{i10}, l_{i13}, l_{i14}$ behave like neurons $l_{i7}, l_{i9}, l_{i11}, l_{i12}$, and eventually σ_{l_k} is activated and the number of spikes from $\sigma_{l_{i10}}$ returns to the form $3s + 2$, for some $s \geq 0$.

The simulation of the ADD instruction is correctly completed.

The SUB module used to simulate a subtraction instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is shown in Figure 2. Because the reader has the experience of examining the work of the ADD module, this time we do not write explicitly the rules, but the sets R_1, R_2, R_3 as defined above. Like in the case of the ADD module, the SUB module starts to work when a spike enters the neuron with the label l_i. The functioning of each neuron is similar to the previous case (the rules to be used are chosen in the same way and eventually the neurons remain with a number of spikes like that in the starting configuration).

The neuron l_i sends a spike to the neurons l_{i1}, and r. If register r is not empty, then the rule $a(aa)^*/a^3 \to a; 0$ of R_2 will be applied.

Assume that this is the case. This means that σ_r spikes immediately, hence $\sigma_{l_{i2}}$ receives two spikes (one from $\sigma_{l_{i1}}$) and is doing nothing, while neuron l_{i3} receives one spike and it fires. A spike is sent to each of the three neurons l_{i7}, l_{i8}, and l_{i9}. This last neuron will send a spike to $\sigma_{l_{i11}}$, which will spike, thus activating the neuron associated with label l_j. The two spikes sent by $\sigma_{l_{i7}}, \sigma_{l_{i8}}$ to $\sigma_{l_{i10}}$ wait here, as no rule is enabled for a number of spikes of the form $3n + 2$. In the next step, a spike comes from neuron l_{i11}, hence $\sigma_{l_{i10}}$ ends with a number of spikes which is a multiple of 3, hence no rule is activated.

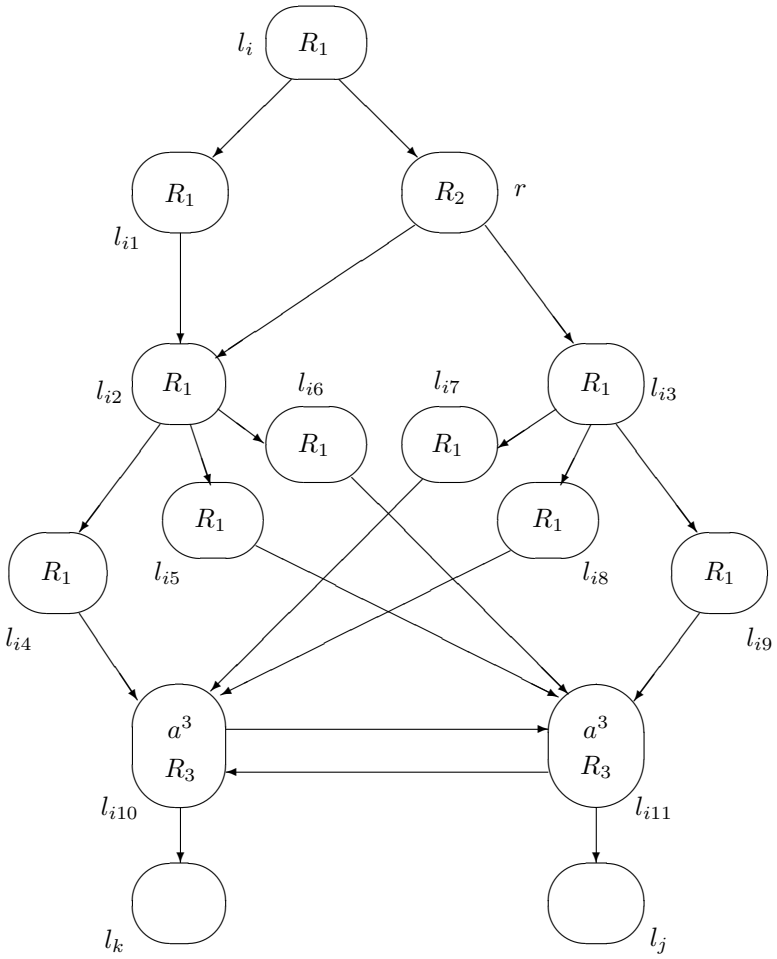

Fig. 2. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

If the register r is empty, then in σ_r we have to use the rule $a \to a; 1$. The neuron l_{i2} receives a spike from $\sigma_{l_{i1}}$ and in the next step it fires, at the same time with the move of spikes produced at the previous step in σ_r and kept there because of the delay. In this moment, all neurons $l_{i2}, l_{i3}, l_{i4}, l_{i5}$, and l_{i6} contains one spike. Neurons l_{i4}, l_{i5}, l_{i6} send their spikes to $\sigma_{l_{i10}}$ and $\sigma_{l_{i11}}$, but they immediately receive one spike from $\sigma_{l_{i2}}$. This also happens with neurons l_{i7}, l_{i8}, l_{i9}, which receive one spike each from $\sigma_{l_{i3}}$. Neuron l_{i10} spikes and activated l_k, sending at the same time one spike to $\sigma_{l_{i11}}$, thus completing here the number of spikes to a multiples of three. Similarly, in the next step, $\sigma_{l_{i10}}$ (resp., $\sigma_{l_{i11}}$) receives three spikes each, from neurons l_{i4}, l_{i7}, l_{i8} (respectively, l_{i5}, l_{i6}, l_{i9}). The simulation of the SUB instruction

is correctly completed, with the neurons containing numbers of spikes of the same parity as in the beginning.

The modules have different neurons, precisely identified by the label of the respective instruction of M. Modules ADD do not interfere. However, a problem appears with modules SUB: when simulating an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, neuron σ_r send one spike to all neurons l_{s2}, l_{s3} from modules associated with instructions $l_s : (\texttt{SUB}(r), l_u, l_v)$ (that is, subtracting from the same register r). However, no undesired effect appears: the spikes arrive simultaneously in neurons l_{s2}, l_{s3}, hence they send one spike to each of the three neurons "below" them, which, in turn, send their spikes to neurons l_{s10}, l_{s11}; each of these neurons gets three spikes, hence no rule can be used here, the spikes are just accumulated (in a number which continues to be a multiple of 3).

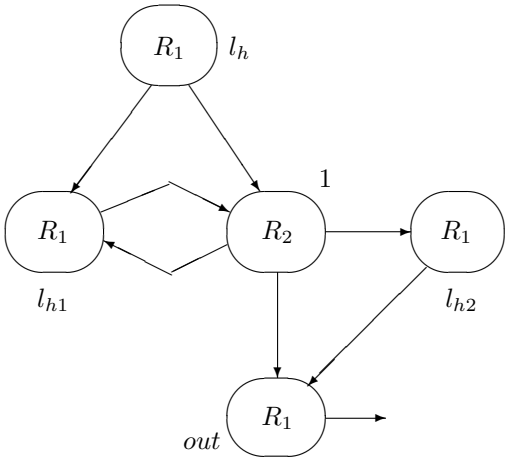

Fig. 3. The FIN module

The addition and subtraction modules simulate the computation of M. In order to produce the number generated by M as the distance between the first two spikes sent out by the system Π we use the module FIN from Figure 3. It is triggered when M reaches the l_h : HALT instruction. At this point a single spike is sent to neuron 1, and at the same time to $\sigma_{l_{h1}}$. Neuron σ_1 sends a spike to each neuron l_{h1}, l_{h2}, and out. The output neuron spikes (for the first time). Neurons σ_1 and $\sigma_{l_{h1}}$ continuously exchange spikes, hence at each step from now on neuron σ_1 contains an odd number of spikes and fires. Neuron out gets two spikes in each step, one from σ_1 and one from $\sigma_{l_{h2}}$, hence nothing happens. When the content of σ_1 is exhausted, the rule $a \to a; 1$ must be used here. The neuron is closed, the spike of $\sigma_{l_{h1}}$ is lost, σ_{out} receives only one spike, from $\sigma_{l_{h2}}$, and spikes for the second time. The work of the system continues forever, because of the interchange of spikes between $\sigma_{l_{h1}}$ and σ_1, but we are interested only in the distance between the first

two spikes emitted by σ_{out}, and this distance is equal to the number stored in register 1 in the end of the computation of M. Consequently, $N(M) = N_2(\Pi)$ and this concludes the proof. □

In the previous figures one can see that the set R_1 appears in neurons having zero or one spike (the case of σ_{l_0}) in the initial configuration, R_2 only with zero spikes, and the set R_3 appears in neurons with zero, two, or three spikes. This means that, if we also consider the number of spikes present in a neuron in the initial configuration when defining the type of a neuron, then the previous $3R$-normal form becomes a $6nR$-normal form.

Corollary 1. $NRE = N_2SNP(6nR, dley)$.

When considering the generated number encoded in the number of spikes present in the output neuron, then several simplifications of the previous constructions are possible. First, the module FIN is no longer necessary; moreover, when a spike is sent to neuron l_h, the computation will halt. Because no instruction is performed in the register machine after reaching the instruction l_h : `HALT`, we provide no outgoing synapse for neuron l_h, so it does matter which rules are present in this neuron, no change is implied on the result of the computation. Then, we only write to register 1, hence to neuron 1 we do not have to apply SUB operations; this means that we can only add spikes to this neuron, namely, one at a time, while any rule can be used inside because no outgoing synapse is present for this neuron. However, always we have the same number of types of neurons, because three types are necessary in modules ADD and SUB. (Similarly, we can use for neuron 1 a FIN module which halves the number of spikes and sends them to another neuron, which leads back to a construction as above.) The results are again as above (the details are left to the reader):

Corollary 2. $NRE = N_{gen}SNP(3R, dley) = N_{gen}SNP(6nR, dley)$.

5 Final Remarks

The accepting case brings further simplifications: the ADD instructions are deterministic (hence only one type of neurons is necessary – see Figure 4, where R_1 is as above), and the FIN module is no longer necessary (we consider the input given as the number of spikes initially present in neuron σ_{in}, without taking into account this number when defining the types of neurons).

However, if we also take into consideration the number of spikes present in the neurons, then we get the following types: $(0, R_1), (1, R_1), (0, R_2), (3, R_3)$, that is, we have the next result:

Corollary 3. $NRE = N_{acc}SNP(3R, dley) = N_{acc}SNP(4nR, dley)$.

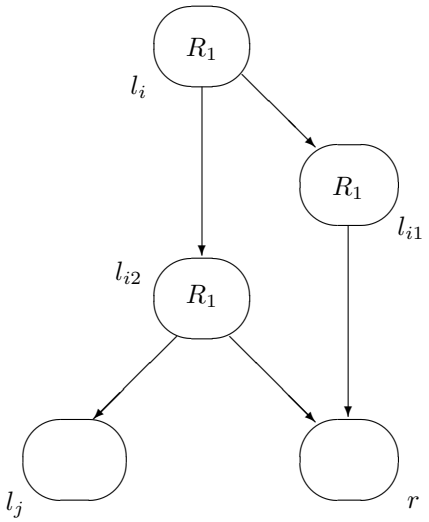

Fig. 4. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j)$

We do not have a construction for module SUB using only two types of neurons.

As mentioned in the Introduction, there are several open problems and research topics suggested by the previous results. We conclude by mentioning a few basic ones. Is the result in Theorem 1 optimal, or a $2R$-normal form or even a $1R$-normal form result is valid? Extend this study to other classes of SN P systems, with other types of rules or with other modes of using the rules.

Important remark: The proof of Theorem 1 implicitly shows that the universality of SN P systems can be obtained without using forgetting rules. The result was first stated in [2], but the construction of the SUB module as given in that paper has a bug: the interaction between neurons from different SUB modules acting on the same register is not examined and the undesired interactions avoided. This is done in the construction from Figure 2, as explicitly mentioned above. Note that our construction uses the delay feature; in [1] it is proved that both the forgetting rules and the delay feature can be avoided without losing the universality, but also there it seems that the interaction of neurons in different SUB modules is not carefully checked. Whether or not the idea in our module SUB can be used to obtain such a stronger normal form remains to be seen.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science

Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. García-Arnau, D. Pérez, A. Rodríguez-Patón, P. Sosík: Spiking neural P systems: Stronger normal forms. *Proc. Fifth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo et al., eds.), Fenix Editora, Sevilla, 2007, 157–178.
2. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Paton, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
3. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308
4. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010 (in press).
5. The P Systems Website: `http://ppage.psystems.eu`.