Spiking Neural P Systems with Anti-Spikes

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. Besides usual spikes employed in spiking neural P systems, we consider "anti-spikes", which participate in spiking and forgetting rules, but also annihilate spikes when meeting in the same neuron. This simple extension of spiking neural P systems is shown to considerably simplify the universality proofs in this area: all rules become of the form $b^c \to b'$ or $b^c \to \lambda$, where b, b' are spikes or anti-spikes. Therefore, the regular expressions which control the spiking are the simplest possible, identifying only a singleton. A possible variation is not to produce anti-spikes in neurons, but to consider some "inhibitory synapses", which transform the spikes which pass along them into anti-spikes. Also in this case, universality is rather easy to obtain, with rules of the above simple forms.

1 Introduction

The spiking neural P systems (in short, SN P systems) were introduced in [4], and then investigated in a large number of papers. We refer to the respective chapter of [7] for general information in this area, and to the membrane computing website from [9] for details.

In this note, we consider a variation of SN P systems which was suggested several times, i.e., involving inhibitory impulses/spikes or inhibitory synapses and investigated in a few papers under various interpretations/formalizations – see, e.g., [1], [2], [5], [8]. The definition we take here for such spikes – we call them *anti-spikes* (somewhat thinking to anti-matter) – considers having, besides usual "positive" spikes denoted by a, objects denoted by $\bar{a}$, which participate in spiking or forgetting rules as usual spikes, but also in implicit rules of the form $a\bar{a} \to \lambda$: if an anti-spike meets a spike in a given neuron, then they annihilate each other, and this happens instantaneously (the disappearance of one a and one $\bar{a}$ takes no time, it is like applying the rule $a\bar{a} \to \lambda$ without consuming any time for that).

This simple extension of SN P systems is proved to entail a surprising simplification of both the proofs and the form of rules necessary for simulating Turing machines (actually, the proofs here are based on simulating register machines) by means of SN P systems: all rules have a singleton regular expression, which, moreover, indicates precisely the number of spikes or anti-spikes to consume by the rule. (Precisely, we have rules of the forms $b^c \to b'$ or $b^c \to \lambda$, where b, b' are spikes or anti-spikes; such rules, having the regular expression E such that $L(E) = b^c$ are called *pure*; formal definitions will be given immediately.) This can be considered as a (surprising) normal form for this case; please compare with the normal forms from [3], especially with the simplifications of regular expressions obtained there.

Anti-spikes are produced from usual spikes by means of usual spiking rules; in turn, rules consuming anti-spikes can produce spikes or anti-spikes (actually, as we will see below, the latter case can be avoided). A possible variant is to produce always only spikes and to consider synapses which "change the nature" of spikes. Also in this case, universality is easily proved, using only pure rules.

2 Prerequisites

We assume the reader to be familiar with basic elements about SN P systems, e.g., from [7] and [9], 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 $\{a\}^*, \{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

The family of Turing computable sets of natural numbers is denoted by NRE.

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:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (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),
- $l_h : \mathtt{HALT}$ (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.

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 $l_i : (\mathtt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\mathtt{ADD}(r), l_j)$).

Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

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

3 Spiking Neural P Systems with Anti-Spikes

We recall first the definition of an SN P system in the classic form (without delays, because this feature is not used in our paper) and of the set of numbers generated or accepted by it.

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

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

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) $E/a^c \rightarrow a$, where E is a regular expression over a and $c \geq 1$;
(2) $a^s \rightarrow \lambda$, for some $s \geq 1$;

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 $E/a^c \rightarrow a$ 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 which is sent immediately to all neurons σ_j such that $(i, j) \in syn$.

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 $a^s \rightarrow \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

Note that we have not imposed here the restriction that for each rule $E/a^c \rightarrow a$ of type (1) and $a^s \rightarrow \lambda$ of type (2) from R_i to have $a^s \notin L(E)$.

If a rule $E/a^c \rightarrow a$ of type (1) has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a$ and we say that it is *pure*.

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, $E_1/a^{c_1} \rightarrow a$ and $E_2/a^{c_2} \rightarrow a$, 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. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The configuration of the system is described by the number of spikes present in each neuron. The initial configuration is $n_1, n_2, \ldots, n_m$. 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 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.

When using an SN P system in the generative mode, we start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron. We denote by $N_2(\Pi)$ the set of numbers computed by Π in this way. 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. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π. It is also possible to introduce the number n by means of a spike train entering neuron σ_{in}, as the distance between the first two spikes coming to σ_{in}.

In the generative case, the neuron (with label) *in* is ignored, in the accepting mode the neuron *out* is ignored (sometimes 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 (by means of) neuron *out*, but we do not consider this case here.

We denote by $N_\alpha SNP(rule_k)$ the families of all sets $N_\alpha(\Pi)$, $\alpha \in \{2, acc\}$, computed by SN P systems with at most $k \geq 1$ rules (spiking or forgetting) in each neuron.

Let us now pass to the extension mentioned in the Introduction. A further object, $\bar{a}$, is added to the alphabet O, and the spiking and forgetting rules are of the forms

$$E/b^c \to b', \; b^c \to \lambda,$$

where E is a regular expression over a or over $\bar{a}$, while $b, b' \in \{a, \bar{a}\}$, and $c \geq 1$. As above, if $L(E) = b^c$, then we write the first rule as $b^c \to b'$ and we say that it is pure.

Note that we have four categories of rules, identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$.

The rules are used as in a usual SN P system, with the additional fact that a and $\bar{a}$ "cannot stay together", they instantaneously annihilate each other: if in a neuron there are either objects a or objects $\bar{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^r and $\bar{a}^s$ inside, then immediately a rule of the form $a\bar{a} \to \lambda$ is applied in a maximal manner, so that either a^{r-s} or $\bar{a}^{s-r}$ remain, provided that $r \geq s$ or $s \geq r$, respectively.

We stress the fact that the mutual annihilation of spikes and anti-spikes takes no time, so that the neuron always contains either only spikes or anti-spikes. That is why, for instance, the regular expressions of the spiking rules are defined either on a or on $\bar{a}$, but not on both symbols. Of course, we can also imagine that the annihilation takes one time unit, when the explicit rule $a\bar{a} \to \lambda$ is used, but we do not consider this case here (if the rule $a\bar{a} \to \lambda$ has priority over other rules, then no essential change occurs in the proofs below).

The computations and the result of computations are defined in the same way as for usual SN P systems – but we consider the restriction that the output neuron produces only spikes, not also anti-spikes (again, this is a restriction which is only natural/elegant, but not essential). As above, we denote by $N_\alpha S_a NP(rule_k, forg)$ the families of all sets $N_\alpha(\Pi)$, $\alpha \in \{2, acc\}$, computed by SN P systems with at most $k \geq 1$ rules (spiking or forgetting) in each neuron, using also anti-spikes. When only pure rules are used, we write $N_\alpha S_a NP(prule_k)$.

4 Universality Results

We start by considering the generative case, for which we have the next result (universality is known for usual SN P systems, without anti-spikes, but now both the proof is simpler and the used rules are all pure):

Theorem 1. *$NRE = N_2 S_a NP(prule_2)$.*

Proof. We only have to prove the inclusion $NRE \subseteq N_2S_aNP(prule_2, forg)$.

Let us consider a register machine $M = (m, H, l_0, l_h, I)$ as introduced in Section 2. We construct an SN P system Π (with $O = \{a, \bar{a}\}$) which simulates M in the way already standard in the literature 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 σ_r in Π, and if the register contains the number n, then the associated neuron will contain n spikes, except for the neuron σ_1 associated with the first register (the neurons associated with registers will either contain occurrences of a, hence $\bar{a}$ disappears immediately, or only $\bar{a}$ is present, and it is consumed in the next step by a rule $\bar{a} \to a$). Two spikes are initially placed in the neuron σ_1 associated with the first register, so if the first register contains the number n, then neuron σ_1 will contain $n + 2$ spikes. These two spikes are used for outputting the computation result. Note that the number of spikes in the neuron σ_1 will not be smaller than two before the simulation reaches the instruction l_h and the output module FIN is activated, because we assume that the output register is never decremented during the computation. One neuron σ_{l_i} is associated with each label $l_i \in H$, and some auxiliary neurons $\sigma_{l_i^{(j)}}, j = 1, 2, 3, \ldots$, will be also considered, thus precisely identified by label l_i (remember that each $l_i \in H$ is associated with a unique instruction of M).

The modules will be given in a graphical form, indicating the synapses and, for each neuron, the associated set of rules. In the initial configuration, all neurons are empty, except for the neurons associated with label l_0 of M and the first register, which contain one spike and two spikes, respectively. In general, when a spike a is sent to a neuron σ_{l_i}, with $l_i \in H$, then that neuron becomes active and the module associated with the respective instruction of M starts to work, simulating the instruction.

The functioning of the module from Figure 1, simulating an instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, is obvious; the non-deterministic choice between instructions l_j and l_k is done by non-deterministically choosing the rule to apply in neuron $\sigma_{l_i^{(3)}}$.

The simulation of an instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is also simple – see the module from Figure 2. The neuron σ_{l_i} sends a spike to neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In the next step, neuron $\sigma_{l_i^{(2)}}$ sends an anti-spike to neuron σ_r, corresponding to register r; at the same time, $\sigma_{l_i^{(1)}}$ sends a spike to each neuron $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$. If register r is non-empty, that is, neuron σ_r contains at least one a, then $\bar{a}$ removes one occurrence of a, which corresponds to subtracting one from register r, and no rule is applied in σ_r. This means $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive only two spikes, from $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$, hence σ_{l_j} is activated and σ_{l_k} not. If register r is empty, then the rule $\bar{a} \to a$ is used in σ_r, hence $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive three spikes, and this leads to the activation of σ_{l_k}, which is the correct continuation also in this case.

Note that if there are several sub instructions l_t which act on register r, then σ_r will send one spike to neurons $\sigma_{l_t^{(5)}}$ and $\sigma_{l_t^{(6)}}$ while simulating the instruction

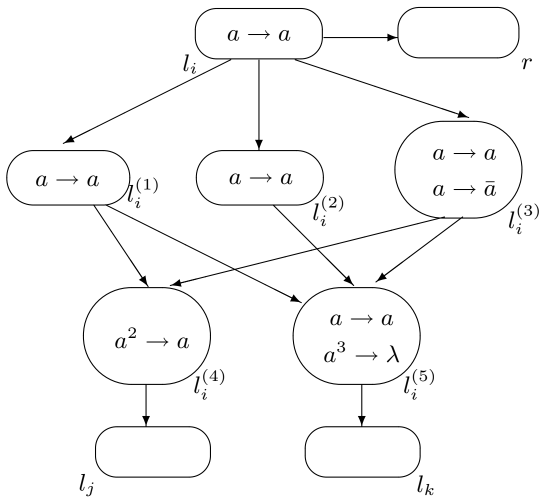

Fig. 1. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

$l_i : (\texttt{SUB}(r), l_j, l_k)$, but this spike is immediately removed by the rule $a \to \lambda$ present in all neurons $\sigma_{l_t^{(5)}}, \sigma_{l_t^{(6)}}$.

The module FIN, which produces a spike train such that the distance between the first two spikes equals the number stored in register 1 of M, is indicated in Figure 3. At some step t, the neuron σ_{l_h} is activated, which means that the register machine M reaches the halt instruction and the system Π starts to output the result. Suppose the number stored in register 1 of M is n. At step $t+2$, neurons σ_{h_1}, σ_{h_3} and σ_{h_4} contain a spike. Neurons σ_{h_1} and σ_{h_4} exchange spikes among them, and thus σ_{h_4} sends a spike to neuron σ_{h_5} continuously until neuron σ_1 spikes and neurons σ_{h_1}, σ_{h_4}, σ_{h_5} are “flooded”. At step $t+4$, neuron σ_{out} receives a spike, and in the next step σ_{out} sends a spike to the environment; at the same time, σ_1 receives an anti-spike that decreases by one the number of spikes from σ_1. At step $t+n+4$, the neuron σ_1 contains one spikes, and in the next step neuron σ_1 sends a spike to neuron σ_{out}. At step $t+n+6$, neuron σ_{out} spikes again. The distance between the first two spikes emitted by σ_{out} equals n, which is exactly the number stored in register 1 of M. The spike produced by neuron σ_1 “floods” neurons σ_{h_1}, σ_{h_4}, and σ_{h_5}, thus blocking the work of these neurons. After the system sends the second spike out, the whole system halts.

From the previous explanations we get the equality $N(M) = N_2(\Pi)$ and this concludes the proof. $\square$

Note that in the previous construction there is no rule of the form $\bar{a}^c \to \bar{a}$; is it possible to also avoid other types of rules? For instance, the rule $\bar{a} \to a$ only appears in the neurons associated with registers in module SUB. Is it possible to remove the $\bar{a} \to a$ by replacing it with the rules $a^c \to a$ and $a \to \bar{a}$?

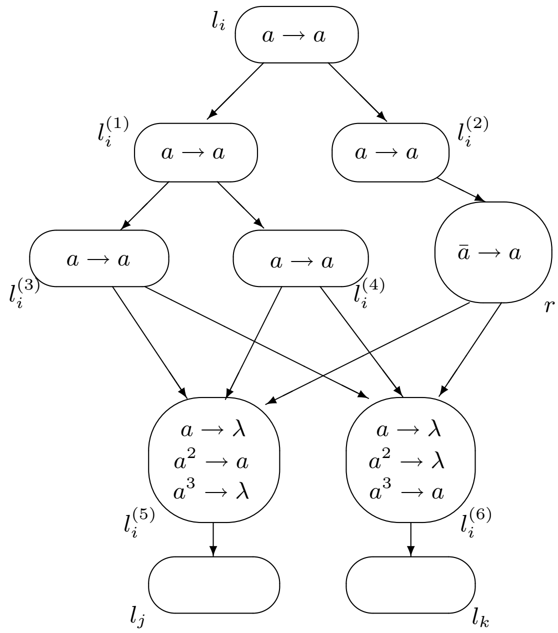

Fig. 2. Module SUB, simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$

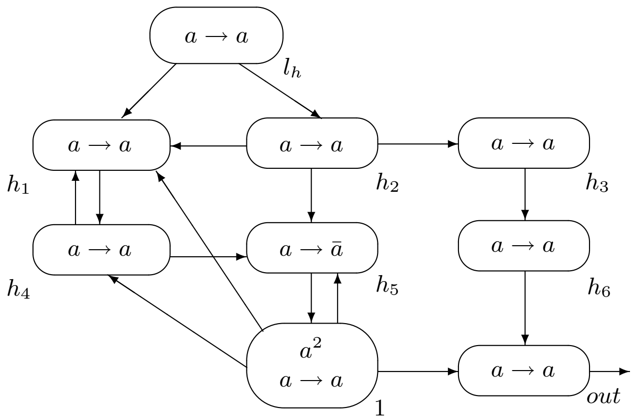

Fig. 3. The FIN module

If the SN P systems are used in the accepting mode, then a further simplification is entailed by the fact that the ADD instructions are deterministic. Such an instruction $l_i : (\mathtt{ADD}(r), l_j)$ can be directly simulated by a simple module as in Figure 4.

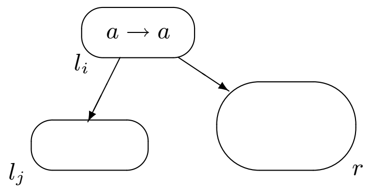

Fig. 4. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j)$

Together with SUB modules, this suffices in the case when the number to accept is introduced as the number of spikes initially present in neuron σ_1. If this number is introduced in the system as the distance between the first two spikes which enters the input neuron, then a input module is necessary, as used, for instance, in [3]. Note that the module INPUT from [3] uses only pure rules (involving only spikes, not also anti-spikes), hence we get a theorem like Theorem 1 also for the accepting case, for both ways of providing the input number.

It is worth mentioning that in the previous constructions we do not have spiking rules which can be used at the same time with forgetting rules.

5 Using Inhibitory Synapses

Let us now consider the case when no rule can produce an anti-spike, but there are synapses which transform spikes into anti-spikes. The previous modules ADD, SUB, FIN can be modified in such a way to obtain a characterization of NRE also in this case. We directly provide these modules, without any explanation about their functioning, in Figures 5, 6, and 7; the synapses which change a into $\bar{a}$ are marked with a dot.

Note that this time the non-determinism in the ADD instruction is simulated by allowing the non-deterministic choice among the spiking rule $\bar{a} \to a$ and the forgetting rule $\bar{a} \to \lambda$ of neuron $\sigma_{l_i^{(1)}}$, which is not allowed in the classic definition of SN P systems. Removing this feature, without introducing rules which are not pure or other ingredients, such as the delay, remains as an open problem.

Denoting by $N_\alpha S_a NP_s(prule_k)$ the respective families of sets of numbers (the subscript s in P_s indicates the use of inhibitory synapses, in the sense specified above), we conclude having the next result:

Theorem 2. $NRE = N_2 S_a NP_s(prule_2)$.

6 Final Remarks

There are several open problems and research topics suggested by the previous results. Some of them were already mentioned, but further questions can be formulated. For instance, can the proofs be improved so that less types of rules are

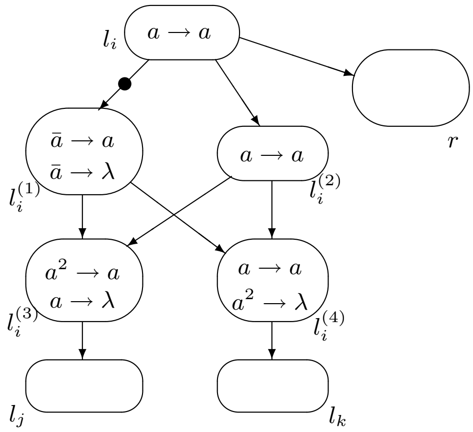

Fig. 5. Module ADD, simulating $l_i : (\mathtt{ADD}(r), l_j, l_k)$

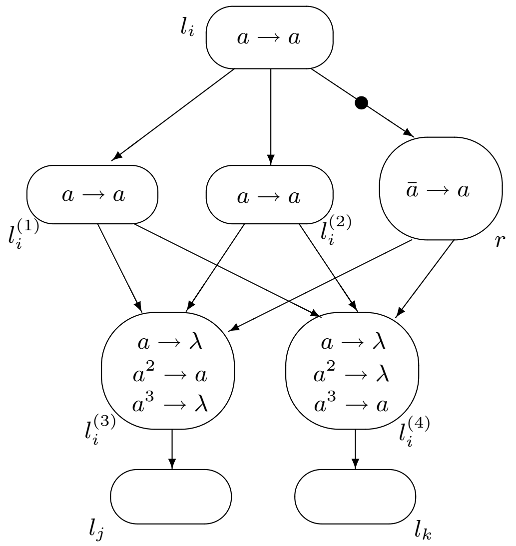

Fig. 6. Module SUB, simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$

necessary? We have avoided using rules $\bar{a}^c \to \bar{a}$, but not the other three types, corresponding to the pairs $(a, a), (a, \bar{a}), (\bar{a}, a)$. Then, following the idea from [6], can we decrease the number of *types* of neurons, in the sense of having a small number of sets of rules which are used in each neuron (three such sets are found in [6] to be sufficient for universality in the case of usual SN P systems; do the anti-spikes helps also in this respect?).

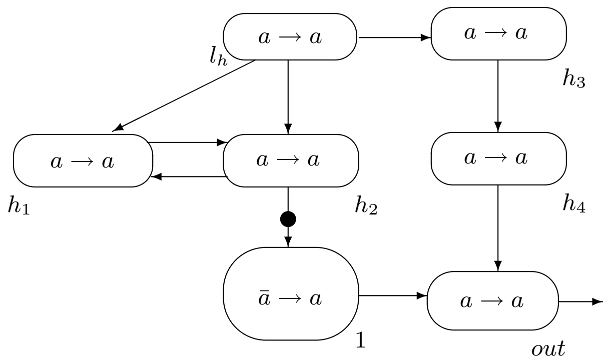

Fig. 7. Module FIN

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 Valia, de la Junta de Andalucia, grant P08 – TIC 04200. Useful remarks by Artiom Alhazov are gratefully acknowledged.

References

1. A. Binder, R. Freund, M. Oswald, L. Vock: Extended spiking neural P systems with excitatory and inhibitory astrocytes. Submitted, 2007.
2. R. Freund, M. Oswald: Spiking neural P systems with inhibitory axons. *AROB Conf.*, Japan, 2007.
3. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
4. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
5. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In the present volume.
6. L. Pan, Gh. Păun: New normal forms for spiking neural P systems. In the present volume.
7. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2009.

8. J. Wang, L. Pan: Excitatory and inhibitory spiking neural P systems. Submitted, 2007.
9. The P Systems Website: `http://ppage.psystems.eu`.