Spiking Neural P Systems with Minimal Parallelism

Yun Jiang[1,2(✉)], Fen Luo[1], and Yueguo Luo[3]

[1] Detection and Control of Integrated Systems Engineering Laboratory, Chongqing Technology and Business University, Chongqing 400067, China
[2] School of Computer Science and Information Engineering, Chongqing Technology and Business University, Chongqing 400067, China
jiangyun@email.ctbu.edu.cn
[3] College of Computer Engineering, Yangtze Normal University, Chongqing 408100, China

Abstract. This paper is an attempt to relax the condition of using the rules in a maximally parallel manner in the framework of spiking neural P systems with exhaustive use of rules. To this aim, we consider the minimal parallelism of using rules: if one rule associated with a neuron can be used, then the rule must be used at least once (but we do not care how many times). In this framework, we study the computational power of our systems as number generating devices. Weak as it might look, this minimal parallelism still leads to universality, even when we eliminate the delay between firing and spiking and the forgetting rules at the same time.

Keywords: Membrane computing · Spiking neural P system · Minimal parallelism · Computing universality

1 Introduction

Through millions of years human brain has evolved into a complex and enormous information processing system, where more than a trillion neurons work in a cooperation manner to perform various tasks. Therefore, brain is a rich source of inspiration for informatics. Specifically, it has provided plenty of ideas to construct high performance computing models, as well as to design efficient algorithm. Inspired from the biological phenomenon that neurons cooperate in the brain by exchanging spikes via synapses, various neural-like computing models have been proposed, such as artificial neural networks [1] and spiking neural networks [2]. These neural-like computing models have performed significantly well in solving real life problems.

In the framework of membrane computing, a kind of distributed and parallel neural-like computing model were proposed in 2006 [3], which is called spiking neural P systems (SN P systems for short). Briefly, an SN P system consists of a set of neurons placed in the nodes of a directed graph that are linked by synapses.

These neurons send signals (spikes) along synapses (edges of the graph). This is done by means of firing rules, which are of the form $E/a^c \to a^p; t$: if the number of spikes contained in the neuron is described by the regular expression E (over the alphabet $\{a\}$), then c spikes are consumed and p spikes are produced after a delay of t steps; the produced spikes are sent to all neurons to which there exist synapse leaving the neuron where the rule is applied. The second type of rules are forgetting rules, of the form $a^s \to \lambda$. The meaning is that $s \geq 1$ spikes are just removed from the neuron, provided that the neuron contains exactly s spikes. The system evolves synchronously: a global clock is assumed and in each time unit each neuron which can use a rule should do it. But the work of the system is sequential locally: only (at most) one rule is used in each neuron. When the computation halts, no further rule can be applied, and a result is obtained, which can be defined in different ways, e.g., in the form of the time elapsed between the first two consecutive spikes sent into the environment.

For the past decade, there have been quite a few research efforts put forward to SN P systems. Many variants of SN P systems have been considered [4–18]. Most of the obtained classes of SN P systems are computationally universal, equivalent in power to Turing machines [19–28]. An interesting topic is to find small universal SN P systems [29–35]. In certain cases, polynomial solutions to computationally hard problems can also be obtained in this framework [36, 37]. Moreover, SN P systems have been applied to solve real-life problems, for example, to design logic gates, logic circuits [38] and databases [39], to perform basic arithmetic operations [40,41], to represent knowledge [42], to diagnose fault [43–45], to approximately solve combinatorial optimization problems [46].

Besides using the rules of a neuron in the sequential mode introduced above, it is also possible to use the rules in a parallel way. An exhaustive manner of rule application is considered in [47]: whenever a rule is enabled in a neuron, it is used as many times as possible for the number of spikes from that neuron, thus exhausting the spikes it can consume in that neuron. Also in [47], SN P systems with exhaustive use of rules are proved to be universal, both in the accepting and the generative cases.

In this paper we propose a minimal parallelism [48] which, as far as we know, have not yet been considered in the framework of SN P systems: from each set of rules associated with a neuron, if a rule can be used, then the rule must be used at least once (maybe more times, without any restriction). Here we consider SN P systems with minimal parallelism function as generator of numbers, which is encoded in the distance between the first two steps when spikes leave the system. Weak as it might look, we prove the equivalence of our systems with Turing machines in the generative mode, even in the case of eliminating two of its key features – delays and forgetting rules – simultaneously. In the proof of this result, we play a trick (also useful in other similar cases, like [47]) of representing a natural number n (the content of a register) by means of $2n + 2$ spikes (in a neuron).

This work is started by the mathematical definition of SN P systems with minimal parallelism, and then the computational power of SN P systems with minimal parallelism is investigated as number generator, in the case of eliminating delays and forgetting rules at the same time. It is proved in a constructive way that SN P systems with minimal parallelism can compute the family of sets of Turing computable natural numbers. At last, the paper is ended with some comments.

2 SN P Systems with Minimal Parallelism

Before introducing SN P systems with minimal parallelism, we recall some prerequisites, including basic elements of formal language theory [49] and basic notions in SN P systems [3,50].

For a singleton alphabet $\Sigma = \{a\}$, $\Sigma^* = \{a\}^*$ denotes the set of all finite strings of symbol a; the empty string is denoted by λ, and the set of all nonempty strings over $\Sigma = \{a\}$ is denoted by Σ^+. We can simply write a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$. A regular expression over a singleton alphabet Σ is defined as follows: (i) λ and a is a regular expression, (ii) if E_1,E_2 are regular expressions over Σ, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over Σ, and (iii) nothing else is a regular expression over Σ. Each regular expression E is associated with a language $L(E)$, which is defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, (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 Σ.

In what follows, we give the formal definition of SN P systems with minimal parallelism.

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

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

where:

1. $O = \{a\}$ is a singleton alphabet (a is called spike);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form

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

   where:
   (1) $n_i \geq 0$ is the initial number of spikes placed in the neuron σ_i;
   (2) R_i is a finite set of rules of the following two forms:
   - Firing rule: $E/a^c \rightarrow a^p; d$, where E is a a regular expression over $\{a\}$, $c \geq 1$, $d \geq 0$, with the restriction $c \geq p$. Specifically, when $d = 0$, it can be omitted;
   - Forgetting rule: $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ is the set of synapses between neurons, with restriction $(i, i) \notin syn$ for $1 \leq i \leq m$ (no self-loop synapse);
4. $out \in \{1, 2, \ldots, m\}$ indicates the output neuron, which can emit spikes to the environment.

The firing rule of the form $E/a^c \rightarrow a^p; d$ with $c \geq p \geq 1$ is called an *extended rule*; if $p = 1$, the rule is called a *standard rule.* If $L(E) = \{a^c\}$, the rule can be simply written as $a^c \rightarrow a^p; d$. Specifically, if $d = 0$, it can be omitted and the rule can be simply written as $a^c \rightarrow a^p$.

The firing rule $E/a^c \rightarrow a^p; d \in R_i$ can be applied if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$. However, the essential we consider here is not the form of the rules, but the way they are used. Using the rule in a minimal parallel manner, as suggested in the Introduction, means the following. Assume that $k = sc + r$, for some $s \geq 1$ (this means that we must have $k \geq c$) and $0 \leq r < c$ (the remainder of dividing k by c). Then nc, $1 \leq n \leq s$ spikes are consumed, $k - nc$ spikes remain in the neuron σ_i, and np spikes are produced 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 produced spikes are emitted immediately, if $d = 1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is applied at step t, then in steps $t, t+1, \ldots, t+d-1$ the neuron is closed, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends a spike along it, then the spike is lost). In step $t + d$, the neuron fires and becomes open again, hence it can receive spikes (which can be used in step $t + d + 1$). The spikes emitted by a neuron are replicated and are sent to all neurons σ_j such that $(i, j) \in syn$.

The forgetting rules are applied as follows: if the neuron contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ can be used, and this means that all s spikes are removed from the neuron.

In each time unit, in each neuron which can use a rule we have to use a rule, either a firing or a forgetting one. Since two firing rules $E_1/a^{c_1} \rightarrow a^p; d_1$ and $E_2/a^{c_2} \rightarrow a^p; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that several rules can be applied in a neuron. This leads to a non-deterministic way of using the rules (but a firing rule cannot be interchanged with a forgetting rule, in the sense that $a^s \notin L(E)$).

The configuration of the system is described both by the numbers of spikes present in each neuron and by the number of steps to wait until it becomes open (if the neuron is already open this number is zero). Thus, the initial configuration is $\langle n_1/0, n_2/0, \ldots, n_m/0 \rangle$. Using the rules as described above, we can define *transitions* among configurations. Any sequence of transitions starting from the initial configuration is called a *computation.* The computation proceeds and a spike train, a sequence of digits 0 and 1, is associated with each computation by marking with 0 for the steps when no spike is emitted by the output neuron and marking with 1 when one or more spikes exit the system. A computation halts if it reaches a configuration where no rule in the neuron can be used.

The result of a computation can be defined in several ways. In this work, we consider SN P systems with minimal parallelism as number generators: assuming the first time and the second time spike(s) emitted from the output neuron is at step t_1 and t_2, the computation result is defined as the number $t_2 - t_1$. For an SN P system Π, the set of all numbers computed in this way is denoted by $N_2^{min}(\Pi)$, with the subscript 2 reminding that only the distance between the first two spikes of any computation is considered. Then, we denote by $Spik_2N^{min}P_m(rule_k, cons_r, forg_q)$ the family of all sets $N_2^{min}(\Pi)$ computed as above by SN P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^c \to a^p; d$ having $c \leq r$, and all forgetting rules $a^s \to \lambda$ having $s \leq q$. When one of the parameters m, k, r, q is not bounded, then it is replaced with $*$.

In order to clarify the definitions, we here discuss an example. In this way, we also introduce a standard way to pictorially represent a configuration of an SN P system, in particular, the initial configuration. Specifically, each neuron is represented by a "membrane" (a circle or an oval), marked with a label and having inside both the current number of spikes (written explicitly, in the form a^n for n spikes present in a neuron) and the evolution rules; the synapses linking the neurons are represented by arrows; besides the fact that the output neuron will be identified by its label, i_0, it is also suggestive to draw a short arrow which exits from it, pointing to the environment.

In the system Π_1 (Fig. 1) we have two neurons, labeled with 1 and 2 (with neuron σ_2 being the output one), which have 5 and 2 spikes, respectively, in the initial configuration. Both of the two neurons fire at the first step of the computation.

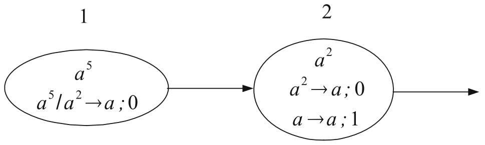

Fig. 1. A simple example of an SN P system with minimal parallel

In the output neuron σ_2, the rule $a^2 \to a; 0$ is used at the first step, a spike is sent out to the environment and no spike remains in σ_2. In neuron σ_1, one can notice that $5 = 2 \times 2 + 1$, hence we can non-deterministically choose the rule $a^5/a^2 \to a$ to be applied once or twice at the same time. If σ_1 uses the rule twice at the first step, it sends two spikes to neuron σ_2 immediately (one for each use of the rule) and neuron σ_2 spikes again, using the rule $a^2 \to a; 0$ at the second step of the computation. Thus, the result of the computation in this case is $2 - 1 = 1$.

If the rule $a^5/a^2 \to a$ is used only once at the first step by neuron σ_1, it sends one spike to neuron σ_2. At the second step, neuron σ_2 fires again, using the rule $a \to a; 1$ at the second step. At the third step the spike generated emits the system and the result of the computation is $3 - 1 = 2$.

Hence, Π_1 generates the finite set $\{1, 2\}$.

3 Universality Result

In this section we prove that SN P systems with minimal parallelism are still universal when eliminating both delays and forgetting rules at the same time.

Nevertheless, the elimination has a price in terms of the number of neurons in the modules. Our universality proof will use the characterization of NRE by means of register machine.

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 labeled instructions are of the following forms:

- $l_i : (ADD(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j),
- $l_i : (SUB(r), l_j, l_k)$ (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),
- $l_h : HALT$ (the halt instruction).

It is known that the register machine can compute all sets of numbers which are Turing computable, even with only three registers [51]. Hence, it characterizes *NRE*, i.e., $N(M) = NRE$ (*NRE* is the family of length sets of recursively enumerable languages – the family of languages recognized by Turing machines). We have the convention that when comparing the power of two number generating devices, number zero is ignored.

In the proof below, we use the characterization of NRE by means of register machine, with an additional care paid to the delay from firing to spiking, and forgetting rules. Because all rules we use have the delay 0, we write them in the simpler form $E/a^c \to a^p$, hence omitting the indication of the delay. Also a change is made in the notation below: we add $dley_t$ to the list of features mentioned between parentheses, meaning that we use SN P systems whose rules $E/a^c \to a^p; d$ have $d \leq t$ (the delay is at most t).

Theorem 1.

$$Spik_2N^{min}P_*(rule_3, cons_4, forg_0, dley_0) = NRE.$$

Proof. In view of the Turing-Church thesis, the inclusion in NRE can be proved directly, so we only have to prove the inclusion $NRE \subseteq Spik_2N^{min}P_*(rule_3, cons_4, forg_0, dley_0)$. The proof is achieved in a constructive way, that is, an SN P system with minimal parallelism is constructed to simulate the universal register machine.

Let $M = (m, H, l_0, l_h, I)$ be a universal register machine. Without lose of generality, we assume that the result of a computation is the number from register 1 and this register is never decremented during the computation.

We construct an SN P system Π as follows, simulating the register machine M and spiking only twice, at an interval of time which corresponds to the number computed by the register machine.

For each register r of M we consider a neuron σ_r in Π whose contents correspond to the contents of the register. Specifically, if the register r holds the number $n \geq 0$, then the neuron σ_r will contain $2n + 2$ spikes; therefore, any register with the value 0 will contain two spikes.

Increasing by 1 the contents of a register r which holds the number n means increasing by 2 the number of spikes from the neuron σ_r; decreasing by 1 the contents of a non-empty register means to decrease by 2 the number of spikes; checking whether the register is empty amounts at checking whether σ_r has two spikes inside.

With each label l of an instruction in M we also associate a neuron σ_l. Initially, all these neurons are empty, except for the neuron σ_{l_0} associated with the start label of M, which contains 2 spikes. This means that this neuron is "activated". During the computation, the neuron σ_l which receives 2 spikes will become active. Thus, simulating an instruction $l_i : (op(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by op, then introducing 2 spikes in one of the neuron σ_{l_j}, σ_{l_k}, which becomes active in this way. When the neuron σ_{l_h}, associated with the halting label of M, is activated, the computation in M is completely simulated in Π, and we have to output the result in the form of a spike train with the distance between the first two spikes equals to the number stored in the first register of M. Additional neurons will be associated with the registers and the labels of M, in a way which will be described immediately.

In what follows, the work of system Π is described (that is how system Π simulates the ADD, SUB instructions of register machine M and outputs the computation result).

Simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$
This instruction adds one to the register r and switches non-deterministically to label l_j or l_k. As seen in Fig. 2, this module is initiated when two spikes enter neuron σ_{l_i}. Then the neuron σ_{l_i} sends two spikes to neuron $\sigma_{i,1}$, $\sigma_{i,2}$ and σ_r, adding one to the content of the register. In the next step, the spikes emitted by $\sigma_{i,1}$ arrive in $\sigma_{i,3}$ (which will in turn be sent to $\sigma_{i,6}$ in the following step) and $\sigma_{i,4}$, while the spikes of $\sigma_{i,2}$ reaches $\sigma_{i,4}$ and $\sigma_{i,5}$. Neuron $\sigma_{i,4}$ will allow us to switch non-deterministically to either σ_{l_j} or σ_{l_k}. If $\sigma_{i,4}$ uses the rule $a^4/a^2 \rightarrow a$ only once, then two spikes will be blocked in $\sigma_{i,8}$ (those coming from $\sigma_{i,4}$ and $\sigma_{i,5}$), while just one will arrive in neuron $\sigma_{i,7}$, waiting for other spikes to come. In the next step, the neuron $\sigma_{i,4}$ fires again (this time applying the rule $a^2 \rightarrow a^2$), sending two spikes to $\sigma_{i,7}$ and $\sigma_{i,8}$. The two spikes are blocked in $\sigma_{i,8}$ again, while the two spikes from $\sigma_{i,4}$ and the spike from $\sigma_{i,6}$ reaches $\sigma_{i,7}$. Together with the one spike await, the neuron $\sigma_{i,7}$ can get fired, activating neuron σ_{l_j} one step later.

On the other hand, if $\sigma_{i,4}$ uses the rule $a^4/a^2 \rightarrow a$ twice, it consumes all of the four spikes inside, and produces two spikes, having no spike left. This means that, in the following step, $\sigma_{i,7}$ receives two spikes and $\sigma_{i,8}$ gets three (two from $\sigma_{i,4}$ and one from $\sigma_{i,5}$). Now $\sigma_{i,8}$ contains three spikes and fires, activating σ_{l_k} in the following step. One step later, $\sigma_{i,7}$ receives another spike from $\sigma_{i,6}$ and the three spikes together get blocked here.

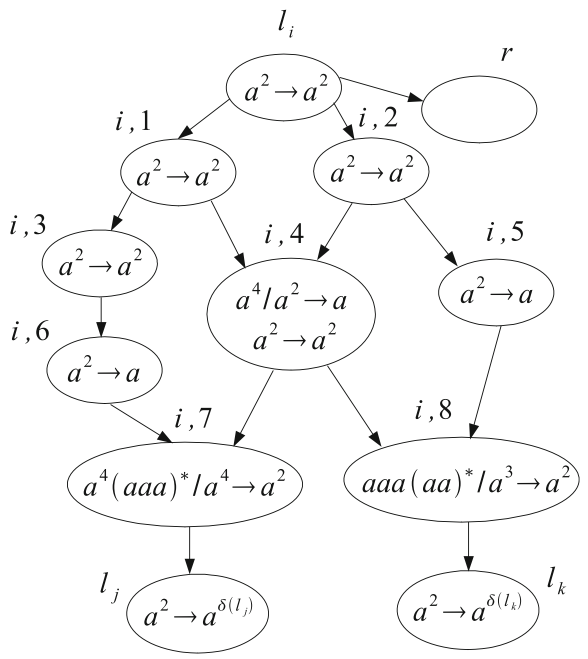

Fig. 2. Module ADD (simulating $l_i : (ADD(r), l_j, l_k)$)

It is clear that, after each ADD instruction, neuron $\sigma_{i,7}$ and $\sigma_{i,8}$ will hold three or two spikes, respectively, depending on the times of application of rule $a^4/a^2 \to a$ chosen non-deterministically in the neuron $\sigma_{i,4}$. Thanks to the regular expressions used in the rules of $\sigma_{i,7}$ and $\sigma_{i,8}$, this does not disturb further computations using this instruction.

Simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$
The SUB module (shown in Fig. 3) is initiated when two spikes are sent to neuron σ_{l_i}. This neuron fires and its spike reaches neurons $\sigma_{r,1}$, $\sigma_{r,2}$ and σ_r. The three rules of neuron σ_r allow us to differentiate whether the register is empty or not. As we have explained, a register containing the value n means the corresponding neuron holds $2n + 2$ spikes. If register r stores number $n > 0$, that is to say σ_r contains at least 4 spikes, the spike coming from σ_{l_i} makes the neuron fire (by the rule $aaa(aa)^+/a^3 \to a$) and send a spike to $\sigma_{r,6}$, which makes sure that the rule is used only once. In the next step, $\sigma_{r,6}$ gets fired and send a spike to $\sigma_{r,8}$, and another spike passing from $\sigma_{r,2}$ to $\sigma_{r,4}$ reaches $\sigma_{r,8}$ at the same time. These two spikes make $\sigma_{r,8}$ can not be fired. In parallel, two spikes will arrive in neuron $\sigma_{r,7}$, one along $\sigma_{r,1}$-$\sigma_{r,3}$-$\sigma_{r,5}$-$\sigma_{r,7}$, and the other σ_r-$\sigma_{r,6}$-$\sigma_{r,7}$. These two spikes make $\sigma_{r,7}$ get fire, then $\sigma_{r,9}$, and as we will explain later, eventually allow us to reach σ_{l_j}.

On the other hand, when register r stores number zero (σ_r contains 2 spikes), the spike received from σ_{l_i} fires the rule $a^3/a^2 \to a$. The neuron σ_r spikes,

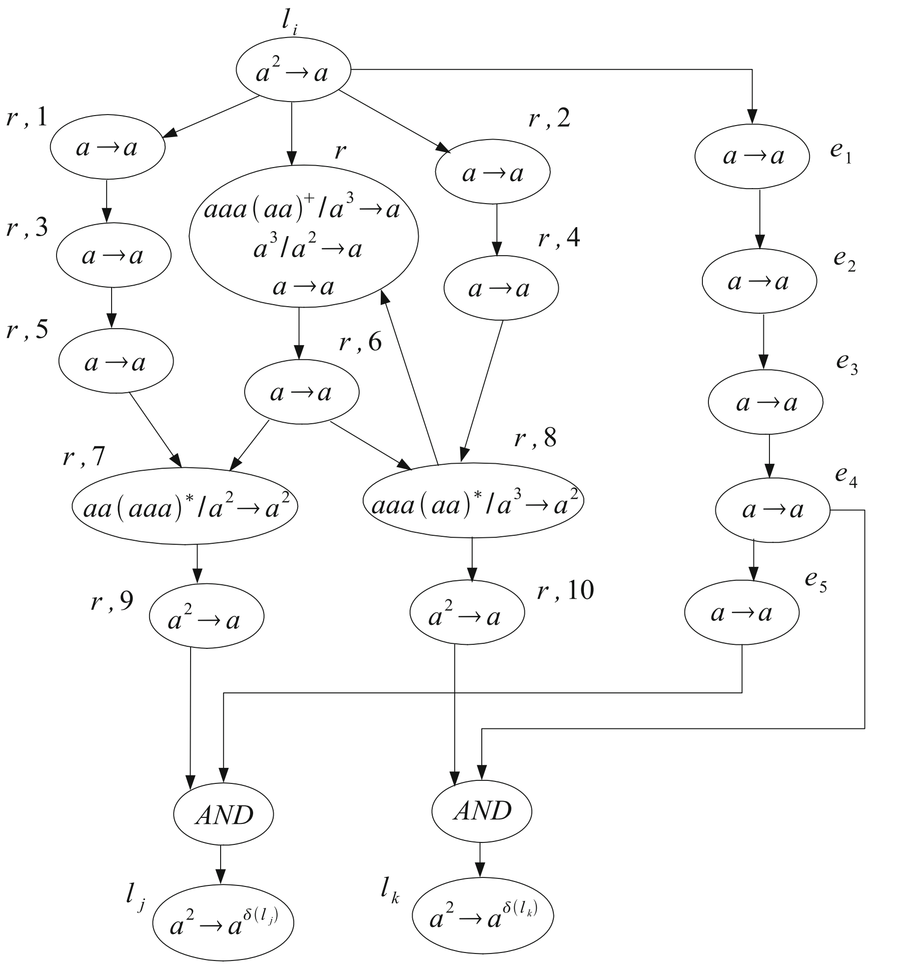

Fig. 3. Module SUB (simulating $l_i : (SUB(r), l_j, l_k)$)

consuming two of the three spikes it contains. This spike is sent to neuron $\sigma_{r,6}$, then to neurons $\sigma_{r,7}$ and $\sigma_{r,8}$. In the following step, σ_r fires again (using the rule $a \to a$), consuming its last spike and sending a new spike to $\sigma_{r,7}$ and $\sigma_{r,8}$ (the value of register r is now degraded and needs to be reconstituted). This spike reaches neuron $\sigma_{r,7}$ at the same time that the one coming from $\sigma_{r,5}$. Then $\sigma_{r,7}$ cannot get fired because it contains now three spikes. Meanwhile, $\sigma_{r,8}$ also contains now three spikes. It gets fired and spikes, allowing $\sigma_{r,10}$ to fire in the following step. It also emits two spikes to σ_r which reconstitute the value 0 in the register r before reaching σ_{l_k}.

The reader can check that two AND modules (the AND module is shown in Fig. 4) are embedded in the SUB module, making sure that further instructions associated with register r will not wrongly switches to l_i and l_j here. In this module, only if σ_{d_1} and σ_{d_2} get fired simultaneously, neuron σ_{d_6} receives three spikes and gets fired. Otherwise, it will receive two spikes and get blocked. Hence, σ_{l_j} or σ_{l_k} can get fired only if σ_{l_i} is activated, and other instructions associated with register r will not activate them wrongly. Also the spikes remained in the neurons $\sigma_{r,7}$ and $\sigma_{r,8}$ do not disturb further computations.

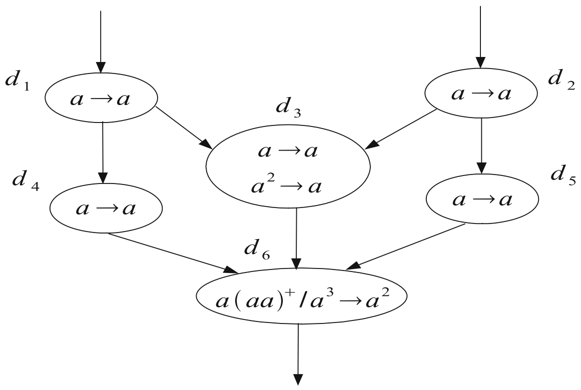

Fig. 4. Module AND

Because we do not know whether l_j and l_k are labels of ADD, SUB, or halting instructions, hence in both of the ADD and the SUB module the rules in the neurons σ_{l_j}, σ_{l_k} are written in the form $a^2 \to a^{\delta(l_s)}$, where the function $\delta : H \to \{1, 2\}$ is defined as below:

$$\delta(l) = \begin{cases} 2, \text{ if } l \text{ is the label of an ADD instruction,} \\ 1, \text{ otherwise.} \end{cases}$$

Outputting a computation

As shown in Fig. 5, when the computation in M halts, two spikes reach the neuron σ_{l_h} of Π_1. In that moment, register 1 of M stores value n and neuron σ_1 of Π_1 contains $2n + 2$ spikes. The spike emitted by l_h reaches neuron 1 (hence containing an odd number of spikes). Thanks to the neuron $\sigma_{h,2}$, it leads neuron σ_1 to fire continuously, consuming two spikes at each step. One step after receiving the spike from σ_{l_h}, neuron σ_1 fires and two spikes reach neuron $\sigma_{h,2}$. Next, neuron $\sigma_{h,4}$ simultaneously receives a spike from $\sigma_{h,3}$, gets fired and sends a spike to neuron σ_{out}, which spikes for the first time one step later. From then on, neuron $\sigma_{h,4}$ receives a couple of spikes from $\sigma_{h,2}$ that do not let it fire again, until two steps after neuron σ_1 fires for the last time (using the rule $a^3 \to a$). When neuron σ_1 stops spiking, neuron $\sigma_{h,4}$ will receive one spike, making σ_{out} fire again and emitting its second and last spike (exactly n steps after the first one).

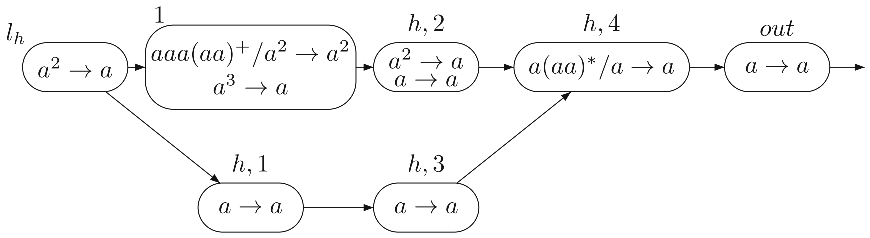

Fig. 5. Module FIN (ending the computation)

From the above description of the work of system Π, it is clear that the system Π_1 correctly simulates the register machine M and outputs exactly the value n computed by M, i.e., $N(M) = N_2^{min}(\Pi)$. This concludes the proof. $\square$

4 Conclusions and Discussions

In this work, we investigated the computational power of SN P systems with minimal parallelism using as number generator. Specifically, an SN P system with minimal parallelism is constructed to compute any set of Turing computable natural numbers. Many issues remain to be investigated for this new class of SN P systems.

It remains to consider the SN P systems with minimal parallelism as language generator. Moreover, it is worth investigating the acceptive case for SN P systems with minimal parallelism.

In the proof of Theorem 1, there are 10 auxiliary neurons in each ADD module, and 27 in each SUB module. So compared to the results in [29–35], the SN P system with minimal parallelism seems to have no advantage in the aspect of small universal SN P system. It is highly possible that this disadvantage can be made up by constructing the modules differently. This task is left as an open problem to the readers.

Also in the proof of Theorem 1, the feature of delay and forgetting rule are not used. It is of interests to check if the number of neurons in universal SN P systems with minimal parallelism can be reduced with using delay or forgetting rule. Moreover, from a point of view of computation power, the boundary between the universality and non-universality of SN P systems with minimal parallelism is still open.

Acknowledgments. This work was supported by National Natural Science Foundation of China (61502063).

References

1. Hagan, M.T., Demuth, H.B., Beale, M.H.: Neural Network Design. PWS Publishing, Boston (1996)
2. Ghosh-Dastidar, S., Adeli, H.: Spiking neural networks. Int. J. Neur. Syst. **19**, 295–308 (2009)
3. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. Fund. Inform. **71**, 279–308 (2006)
4. Cavaliere, M., Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems. Theor. Comput. Sci. **410**, 2352–2364 (2009)
5. Pan, L., Păun, G.: Spiking neural P systems with anti-spikes. Int. J. Comput. Commun. **4**, 273–282 (2009)
6. Ionescu, M., Păun, G., Pérez-Jiménez, M.J., Yokomori, T.: Spiking neural dP systems. Fund. Inform. **111**, 423–436 (2011)
7. Pan, L., Wang, J., Hoogeboom, H.J.: Spiking neural P systems with astrocytes. Neural Comput. **24**, 805–825 (2012)
8. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. Neural Process. Lett. **35**, 13–27 (2012)
9. Song, T., Pan, L., Păun, G.: Spiking neural P systems with rules on synapses. Theor. Comput. Sci. **529**, 82–95 (2014)
10. Wang, J., Hoogeboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with weights. Neural Comput. **22**, 2615–2646 (2014)
11. Song, T., Liu, X., Zeng, X.: Asynchronous spiking neural P systems with anti-spikes. Neural Process. Lett. **42**, 633–647 (2015)
12. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spiking strategy. IEEE Trans. Nanobiosci. **14**, 465–477 (2015)
13. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. IEEE Trans. Nanobiosci. **14**, 38–44 (2015)
14. Song, T., Gong, F., Liu, X., Zhao, Y., Zhang, X.: Spiking neural P systems with white hole neurons. IEEE Trans. Nanobiosci. **15**, 666–673 (2016)
15. Zhao, Y., Liu, X., Wang, W., Adamatzky, A.: Spiking neural P systems with neuron division and dissolution. PLoS ONE **11**, e0162882 (2016)
16. Wu, T., Zhang, Z., Păun, G., Pan, L.: Cell-like spiking neural P systems. Theor. Comput. Sci. **623**, 180–189 (2016)
17. Jiang, K., Chen, W., Zhang, Y., Pan, L.: Spiking neural P systems with homogeneous neurons and synapses. Neurocomputing **171**, 1548–1555 (2016)
18. Song, T., Pan, L.: Spiking neural P systems with request rules. Neurocomputing **193**, 193–200 (2016)
19. Ibarra, O.H., Păun, A., Rodríguez-Patón, A.: Sequential SNP systems based on min/max spike number. Theor. Comput. Sci. **410**, 2982–2991 (2009)
20. Neary, T.: A boundary between universality and non-universality in extended spiking neural P systems. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 475–487. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13089-2_40
21. Song, T., Pan, L., Jiang, K., Song, B., Chen, W.: Normal forms for some classes of sequential spiking neural P systems. IEEE Trans. Nanobiosci. **12**, 255–264 (2013)
22. Zhang, X., Zeng, X., Luo, B., Pan, L.: On some classes of sequential spiking neural P systems. Neural Comput. **26**, 974–997 (2014)

23. Wang, X., Song, T., Gong, F., Zheng, P.: On the computational power of spiking neural P systems with self-organization. Sci. Rep. **6**, 27624 (2016)
24. Chen, H., Freund, R., Ionescu, M.: On string languages generated by spiking neural P systems. Fund. Inform. **75**, 141–162 (2007)
25. Krithivasan, K., Metta, V.P., Garg, D.: On string languages generated by spiking neural P systems with anti-spikes. Int. J. Found. Comput. Sci. **22**, 15–27 (2011)
26. Zeng, X., Xu, L., Liu, X.: On string languages generated by spiking neural P systems with weights. Inform. Sci. **278**, 423–433 (2014)
27. Song, T., Xu, J., Pan, L.: On the universality and non-universality of spiking neural P systems with rules on synapses. IEEE Trans. Nanobiosci. **14**, 960–966 (2015)
28. Wu, T., Zhang, Z., Pan, L.: On languages generated by cell-like spiking neural P systems. IEEE Trans. Nanobiosci. **15**, 455–467 (2016)
29. Păun, G., Păun, A.: Small universal spiking neural P systems. Biosystems **90**, 48–60 (2007)
30. Zhang, X., Zeng, X., Pan, L.: Smaller universal spiking neural P systems. Fund. Inform. **87**, 117–136 (2008)
31. Pan, L., Zeng, X.: A note on small universal spiking neural P systems. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 436–447. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11467-0_29
32. Song, T., Jiang, Y., Shi, X., Zeng, X.: Small universal spiking neural P systems with anti-spikes. J. Comput. Theor. Nanosci. **10**, 999–1006 (2013)
33. Zeng, X., Pan, L., Pérez-Jiménez, M.J.: Small universal simple spiking neural P systems with weights. Sci. China. Inform. Sci. **57**, 1–11 (2014)
34. Metta, V.P., Raghuraman, S., Krithivasan, K.: Small universal spiking neural P systems with cooperating rules as function computing devices. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 300–313. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14370-5_19
35. Song, T., Pan, L.: A small universal spiking neural P systems with cooperating rules. Rom. J. Inf. Sci. Technol. **17**, 177–189 (2014)
36. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X.: Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. Theor. Comput. Sci. **411**, 2345–2358 (2010)
37. Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Sci. China Inform. Sci. **54**, 1596–1607 (2011)
38. Song, T., Zheng, P., Wong, M.L., Wang, X.: Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. Inform. Sci. **372**, 380–391 (2016)
39. Díaz-Pernil, D., Gutiérrez-Naranjo, M.J.: Semantics of deductive databases with spiking neural P systems. Neurocomputing (2017). doi:https://doi.org/10.1016/j.neucom.2017.07.007
40. Zeng, X., Song, T., Zhang, X., Pan, L.: Performing four basic arithmetic operations with spiking neural P systems. IEEE Trans. Nanobiosci. **11**, 366–374 (2012)
41. Liu, X., Li, Z., Liu, J., Liu, L., Zeng, X.: Implementation of arithmetic operations with time-free spiking neural P systems. IEEE Trans. Nanobiosci. **14**, 617–624 (2015)
42. Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M.J., Wang, T.: Weighted fuzzy spiking neural P systems. IEEE Trans. Fuzzy Syst. **21**, 209–220 (2013)

43. Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy reasoning spiking neural P systems for fault diagnosis. Inform. Sci. **235**, 106–116 (2013)
44. Wang, J., Peng, H.: Adaptive fuzzy spiking neural P systems for fuzzy inference and learning. Int. J. Comput. Math. **90**, 857–868 (2013)
45. Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.J.: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. IEEE Trans. Power Syst. **30**, 1182–1194 (2015)
46. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An optimization spiking neural P system for approximately solving combinatorial optimization problems. Int. J. Neur. Syst. **24**, 1440006 (2014)
47. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems with exhaustive use of rules. Int. J. Unconv. Comput. **3**, 135–154 (2007)
48. Ciobanu, G., Pan, L., Păun, G., Pérez-Jiménez, M.J.: P systems with minimal parallelism. Theor. Comput. Sci. **378**, 117–130 (2007)
49. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Language: Volume 3 Beyond Words. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59126-6
50. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010)
51. Minsky, M.: Computation - Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)