Spiking Neural P Systems: Stronger Normal Forms

Marc García-Arnau[1], David Peréz[1], Alfonso Rodríguez-Patón[1], Petr Sosík[1,2]

[1] Universidad Politécnica de Madrid - UPM, Facultad de Informática
Campus de Montegancedo s/n, Boadilla del Monte
28660 Madrid, Spain mgarnau@dia.fi.upm.es

[2] Institute of Computer Science, Silesian University
74601 Opava, Czech Republic petr.sosik@fpf.slu.cz

Summary. Spiking neural P systems are computing devices recently introduced as a bridge between spiking neural nets and membrane computing. Thanks to the rapid research in this field there exists already a series of both theoretical and application studies. In this paper we focus on normal forms of these systems while preserving their computational power. We study combinations of existing normal forms, showing that certain groups of them can be combined without loss of computational power, thus answering partially open problems stated in [8, 9]. We also extend some of the already known normal forms for spiking neural P systems considering determinism and strong acceptance condition. Normal forms can speed-up development and simplify future proofs in this area.

1 Introduction

Spiking neural P systems (SN P systems) are a rather new bio-inspired computational model that incorporates to membrane computing [6] some ideas from spiking neurons [3], [4].

Since they were first presented in [2], the number of publications dealing with this model is constantly growing. An interesting review on the current research topics in SN P systems can be found in [9].

Informally, an SN P system consists of a set of neurons placed in the nodes of a graph that are linked by synapses. These neurons send signals (*spikes*) along the arcs of the graph. To do so, the neurons contain firing or spiking rules which are of the form $E/a^r \to a; t$ with E being a regular expression, r being the number of spikes consumed by the rule and t being the delay from firing the rule and emitting the spike. A firing rule can be only used if the number n of spikes collected by the neuron is such that $a^n \in L(E)$, that is, a^n is covered by the regular expression E, and $n \geq r$. The neurons also have an interesting feature imitating the refractory period of real neural cells. Thus, a neuron in an SN P system is closed/blocked for exactly t time steps after firing. During this period it cannot fire again. The second type of rules have the form $a^s \to \lambda$ and are called forgetting rules. They are used to simply forget (remove) s spikes from a cell. SN P systems start from an initial configuration of spikes and evolve in a synchronized manner (a global clock is assumed for the whole system). One of the neurons is designated as the output cell and the spikes it sends to the environment constitute the output of the system.

The first work in SN P systems [2] presented them as devices generating or accepting sets of natural numbers. Their universality was proven when no bound is imposed on the number of spikes. Otherwise, only a characterization of semilinear sets is obtained. Later, a new paper [8] dealing with normal forms of the model attacked its universality trying to improve the previous proofs. In that article, universality results were obtained even if some of the main features of the model were weakened. For instance, universality was proven even without the use of delays. Furthermore, the outdegree of neurons was reduced to the minimal bound of two. Next, SN P systems were found to be also universal when forgetting rules were removed and, finally, computational completeness was still achieved when using the simplest possible regular expressions λ and a^* over the alphabet $\{a\}$ in firing rules. Another work [10] has also shown that not only the outdegree but also the indegree of neurons can be bound to two without loosing universality.

In the present paper, we deal with some of the open problems stated in [8]. Actually, we try to keep the universality of the model when eliminating two of its key features simultaneously. Interesting results have been obtained. Surprisingly, SN P systems are still universal when we use neither delays nor forgetting rules. Moreover, one can also eliminate delays while simplifying regular expressions and the model keeps its computational completeness. Finally, we have proven the universality of the model in two more cases: 1) using simple regular expressions with strong halting condition and 2) using simple regular expressions with SN P systems working in the accepting mode.

In all these cases, the reader will observe that the simultaneous elimination of two features of the model has a price in terms of other complexity parameters, such as the maximal number of firing rules in a neuron, the complexity of regular expressions, the maximum number of spikes consumed in a firing rule or the maximal number of spikes removed in a forgetting rule.

The remainder of the paper is organized as follows. Section 2 presents some important definitions. In section 3 we present the universality result of the model using neither delays nor forgetting rules. Section 4 describes the power of SN P systems with simple regular expressions that do not use delays. Some more aspects concerning regular expressions are revisited in section 5. The paper concludes with some final remarks in section 6.

2 Definitions

In this section, we recall some useful definitions. We consider the reader to be familiar with elements of membrane computing. One can find in [11] the most updated information on this area. The reader is considered to be familiar with elements of language and automata theory, as well.

Nevertheless, we recall some basic notation. Let V denote an alphabet, while V^* denotes the set of all finite strings of symbols from V. The set of nonempty strings over V is denoted by V^+ and λ denotes the empty string. The length of a string $x \in V^*$ is denoted by $|x|$. In the domain of SN P systems, the alphabet V contains only one symbol, i.e., the alphabet is a singleton $V = \{a\}$. Then a^* and a^+ are normally used instead of $\{a\}^*$ and $\{a\}^+$.

A *spiking neural membrane system* (abbreviated as SN P system), of a degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

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) $E/a^r \rightarrow a; t$, where E is a regular expression over a, $r \geq 1$, and $t \geq 0$;
      (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. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (i.e., σ_{i_0} is the output neuron).

The rules of type (1) are *firing rules* (also called *spiking rules*). The notation $E/a^r \rightarrow a, t$ means that when the number of spikes present in a neuron is covered by the regular expression E, the neuron gets fired, r spikes are consumed and, after t time steps, one spike is emitted by the neuron to all its neighbors (the system is synchronized, a global clock is assumed for all its cells).

In a SN P system we have maximal parallelism at the level of the system as, in one step, all neurons that can use a rule have to use it. However, at the neuronal level, we work in a sequential mode with at most one rule used in each step by a neuron. SN P systems also incorporate an interesting bio-inspired feature called the refractory period. In the interval between using a spiking rule (getting fired) and releasing the spike, the neuron is assumed to be closed (it omits any other spike received during this interval and, of course, it cannot fire). Then, if $t = 0$

there is no restriction and the neuron can receive spikes in the same step it uses the rule. Similarly, a neuron can receive spikes in moment t when $t \geq 1$. When a neuron spikes, its spike is replicated to all its neighboring neurons that are not closed in that moment. These spikes are available in the receiving neuron already in the next step, so the transmission of a spike takes no time.

The rules of type (2) are called *forgetting rules.* They are written as $a^s \rightarrow \lambda$ and they can be applied only when the neuron contains exactly s spikes. After applying one of these rules, s spikes are simply removed from the cell.

Firing rules can be used in a non-deterministic way, that is, a neuron may contain two firing rules $E_1/a^{r_1} \rightarrow a; t_1$ and $E_2/a^{r_2} \rightarrow a; t_2$ such that $L(E_1) \cap L(E_2) \neq \emptyset$. However, this non-determinism is not allowed between firing and forgetting rules, that is, in a single step a neuron has either to fire or forget, without being possible to freely choose between these two actions. This is called the *minimal determinism-like restriction* or the *coherence* condition. Hence, we just allow branching in the case of spiking rules.

We define a *computation* of an SN P system as a sequence of steps during which rules are applied in the above described parallel manner. A computation starts in the initial configuration when each neuron σ_i contains n_i spikes, $1 \leq i \leq m$. A *halting computation* is that in which the system reaches a configuration where no more rules can be applied. A computation is called *strong halting* if, in addition, no spike is present in the system when it halts.

The usual way of interpreting *outputs* of SN P systems is considering *intervals* in which the output neuron i_0 spikes (not when it fires). For simplicity, one considers as successful only computations with the output neuron spiking at least twice. The set of numbers computed by an SN P system in this way is denoted by $N_2(\Pi)$. If we take into consideration only the computations having exactly 2 spikes (strong case) then this set is written as $N_{\underline{2}}(\Pi)$. Similarly, if only halting (strong halting) computations are taken into the account, we denote the resulting sets by $N_2^h(\Pi)$ ($N_2^{\underline{h}}(\Pi)$, respectively). The reader will find in [7] and [8] several other relevant definitions.

We denote by $Spik_2^\beta P_m(rule_k, cons_p, forg_q, dley_r, outd_s)$ the family of all sets $N_2^\beta(\Pi)$ (with $\beta = \{h, \underline{h}\}$), for all systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, each of the forgetting rules removing no more than q spikes, with all spiking rules having a delay small or equal to r and with all neurons having at most s outgoing synapses. We also may write $rule_k^*$ if the firing rules are of the form $E/a^r \rightarrow a; t$ with the regular expression of one of the forms $E = \lambda$ or $E = a^*$ (in the former case the rule is written as $a^r \rightarrow a; t$ to simplify).

Finally, we define a *register machine*, which is the computational model used to prove the universality of SN P systems, as it is known (see [5]) that register machines (even with a small number of registers, although this detail is not relevant here) characterize *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 the instruction HALT), and I is the set of instructions. Each label from H labels only one instruction from I (but the same instruction may be assigned to more labels). The instructions are of the following forms:

- $l_1 : (\mathtt{ADD}(r), l_2, l_3)$ (add 1 to register r and then go to one of the instructions with labels l_2, l_3),
- $l_1 : (\mathtt{SUB}(r), l_2, l_3)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \mathtt{HALT}$ (the halt instruction).

A register machine M computes a number n in the following way: we start with all registers empty (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. Therefore $N(M)$ denotes the set of all numbers computed by the register machine M.

Without loss of generality, we may assume in the next sections 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.

A register machine can also work in the *accepting* mode: a number n is introduced in the first register (all other registers are empty) and we start computing with the instruction with label l_0; if the computation eventually halts, then the number n is accepted. Register machines are universal also in the accepting mode; moreover, this is true even for deterministic machines, having ADD rules of the form $l_1 : (\mathtt{ADD}(r), l_2, l_3)$ with $l_2 = l_3$ (in such a case, the instruction is written in the form $l_1 : (\mathtt{ADD}(r), l_2)$). Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

3 Removing Delays and Forgetting Rules Simultaneously

In this section we extend the original result of Theorem 3.1 of [2], paying special attention to delays and forgetting rules. As it was shown in [8], computational completeness is achieved when eliminating each one of these two parameters separately. Here, we extend these results demonstrating that SN P Systems are also universal when eliminating both delays and forgetting rules at the same time. Simultaneously, we have also bounded the outdegree of neurons to two. However, for the sake of clarity, we don't show it graphically in our demonstration.

Nevertheless, this elimination has a price in terms of other parameters. Namely, the maximal number of rules used in a neuron rises to three and we lose the strong halting condition. As all rules we use have delay 0, we write them in the simpler form $E/a^r \to a$, that is, omitting the delay.

Theorem 1. *$Spik_{\underline{2}}^{\beta}P_*(rule_3, cons_3, forg_0, dley_0, outd_2) = NRE$, where $\beta = h$ or β is omitted.*

Proof. The inclusion $Spik_{\underline{2}}^{\beta}P_*(rule_*, cons_*, forg_*, dley_*, outd_*) \subseteq NRE$ is straightforward and therefore we omit it (Turing-Church thesis). To complete the proof we must show $NRE \subseteq Spik_{\underline{2}}^{\beta}P_*(rule_3, cons_3, forg_0, dley_0, outd_2)$. As in other demonstrations, we will construct an SN P system (Π) spiking only twice, at an interval of time which corresponds to a number computed by a register machine M. Our system consists of modules simulating the ADD and SUB instructions and the output module FIN which takes care of the final spiking of the system Π.

Every register r of M will be associated to a neuron of Π. However, in contrast with some other previous demonstrations, a register containing the value n will hold $2n + 2$ spikes. Any register representing the value 0 will therefore contain a couple of spikes. This slight modification in the way of representing numbers will allow us to detect correctly whether a register contains the value 0 in the SUB module without making use of delays or forgetting rules.

Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ – module ADD (Figure 1).

This instruction adds one to the register r and branches non-deterministically to label l_j or l_k. This module is initiated when a spike enters neuron l_i (we can assume that the initial instruction of M, labeled with l_0, is always an ADD instruction). The neuron l_i sends then one spike to neurons c_1 and c_2. In the next step, one spike coming from each of these neurons reaches the neuron r, adding one to the content of the register. At the same time, the spike emitted by c_1 arrives to c_3 (which will in turn be send to c_6 in the following step) and c_4, while the spike of c_2 reaches c_4 and c_5. Neuron c_4 will allow us to branch non-deterministically to either l_j or l_k. If c_4 uses the rule $a^2 \to a$, then two spikes will be blocked in c_8 (those coming from c_4 and c_5), while just one will arrive to neuron c_7 waiting for another one to come. In the next step, the spike from c_6 reaches also c_7 and it gets fired, activating neuron l_j one step later.

On the other hand, if c_4 uses the rule $a^2/a \to a$ it consumes only one of its two spikes. This means that, in the following step, c_7 receives one spike and c_8 receives two (from c_4 and c_5). One step later, c_4 uses its rule $a \to a$ and sends another spike to c_7 (which also receives the one from c_6 and therefore cannot fire) and to c_8 that now contains three spikes and fires, activating l_k in the following step.

The reader will appreciate that, after each ADD instruction, neurons c_7 and c_8 will hold 3 or 2 spikes, respectively, depending on the rule selected in the non-deterministic neuron c_4. Thanks to the regular expressions used in the rules of c_7 and c_8, this does not disturb further computations using this instruction.

In this construction, neurons c_1 and c_2 have an outdegree of three. However, it is trivial to see that it could be reduced to two by placing more neurons between them and neurons c_3, c_4 and c_5, as it is explained in Section 5 of [8].

Simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ – module SUB (Figure 2).

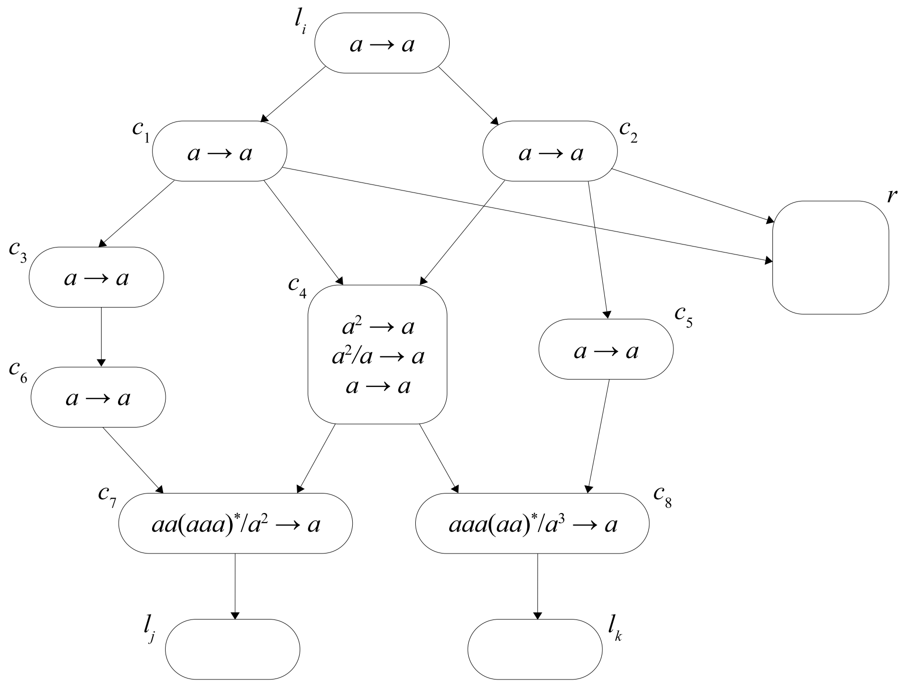

Fig. 1. Module ADD (simulating $l_i : (\mathtt{ADD}(r), l_j, l_k)$)

The module is initiated when a spike is sent to neuron l_i. This neuron fires and its spike reaches neurons d_1, d_2 and r. The three rules of neuron r allow us to differentiate whether the register is empty or not. As we have previously explained, storing the value n means to contain $2n + 2$ spikes. Thus, when $r > 0$, (i.e., it contains at least 4 spikes) the spike coming from l_i makes the neuron fire (rule $aaa(aa)^+/a^3 \to a$) sending a spike to d_4 and d_5. At the same time, another spike coming from d_2 reaches d_5, not allowing it to fire. In parallel, a spike is sent from d_1 to d_3 and, in the following step, it arrives to d_4. This neuron fires because it already contains two spikes, allowing us to finally reach l_j.

On the other hand, when r stores number zero (it contains 2 spikes), the spike received from l_i fires the rule $a^3/a^2 \to a$. Then neuron r spikes, consuming two of the three spikes it contains. This spike is sent to neurons d_4 and d_5. Another spike reaches d_5 simultaneously (from d_2), while d_3 receives the spike coming from d_1. In the following step, r fires again (using the rule $a \to a$), consuming its last spike and sending a new spike to d_4 and d_5 (the value 0 of r is now degraded and needs to be reconstituted). This spike reaches neuron d_4 at the same time that the one coming from d_3. Then d_4 cannot fire as it contains now three spikes. Meanwhile, d_5 receives the new spike from r (d_5 now contains three spikes). It gets fired and spikes, allowing neurons d_6 and d_8 to fire in the following step. Each of these two

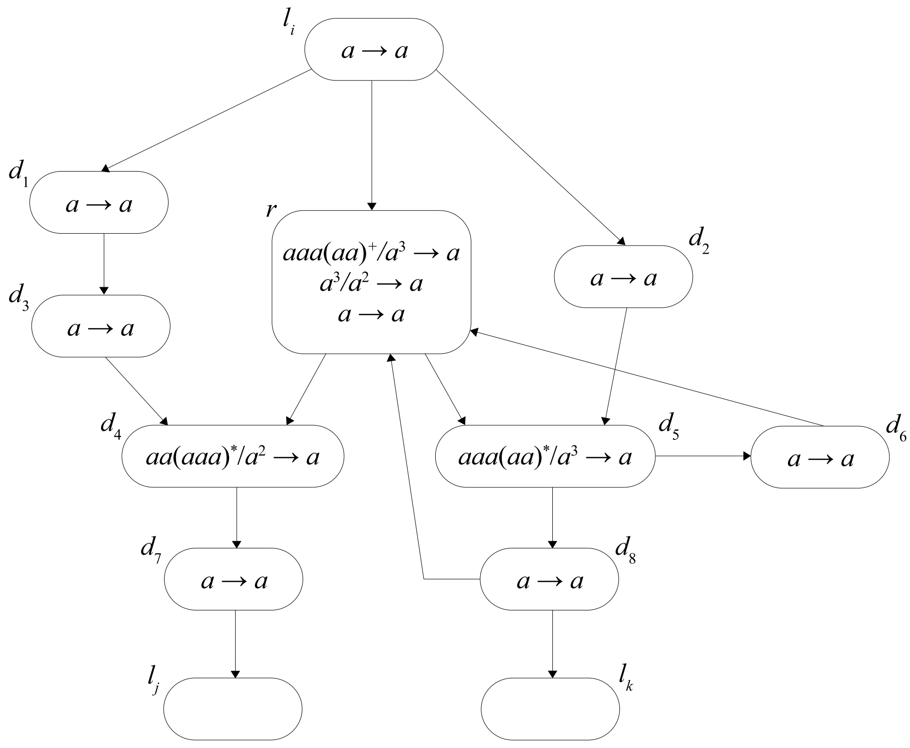

Fig. 2. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

neurons emits then one spike to r which reconstitute the value 0 in the register before reaching l_k.

The reader can check that the remaining spikes in neurons d_4 and d_5 do not disturb further computations. In this case, the outdegree can also be easily reduced to two using a couple of intermediate neurons between l_i and d_1, r and d_2.

Ending a computation – module FIN (Figure 3).

When the computation in M halts, a spike reaches the neuron l_h of Π. In that moment, register 1 of M stores value n and neuron 1 of Π contains $2n + 2$ spikes. The spike emitted by l_h reaches neuron 1 (thus containing an odd number of spikes). This leads neuron 1 to fire continuously, consuming two spikes at each step. One step after receiving the spike from l_h, neuron 1 fires and one spike reaches neuron e_1 and neuron *out*. Next, neuron *out* fires and spikes for the first time. From that step on, neuron *out* simultaneously receives a couple of spikes from 1 and e_1 that do not let it fire again until one step after neuron 1 fires for the last time. When neuron 1 stops spiking, neuron *out* still receives one spike from e_1 making it fire and emitting its second and last spike (exactly n steps after the first one).

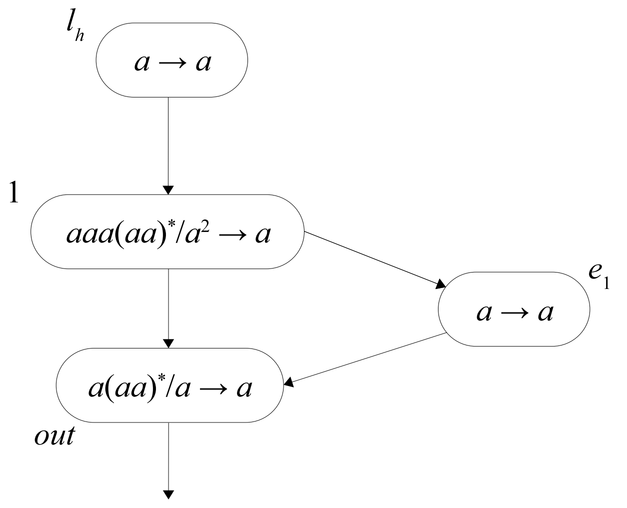

Fig. 3. The FIN module

Once the computation has ended, neuron 1 holds three spikes and neuron *out* contains $2(n-1)$ spikes. As this construction needs to leave some spikes in the system after halting, it cannot be extended to the case of the strong halting. □

4 Removing Delays and Simplifying Regular Expressions

Theorem 7.1 of [8] stated that the regular expressions used in firing rules could be simplified to the point of just using the simplest expressions over the alphabet $\{a\} : \lambda$ and a^*. In this section we consider the same problem, but removing delays simultaneously. Surprisingly, the proof construction shows that SN P systems are still universal even in that case. Moreover, we keep universality using, in each neuron, one rule of the form $a^*/a \to a$ or $(a^r \to a)$, and at most two rules $a^s \to \lambda$, with $r, s \leq 3$. (with the only exception of the non-deterministic neuron c_3 in the ADD module). Finally, we have also kept the limitation of outdegree ≤ 2 for each neuron. Comparing this result with that of Theorem 7.1 of [8], one can notice that removing delays has some computational cost in terms of other parameters, as the maximum degree of forgetting rules, the number of rules per neuron and the maximum number of spikes consumed in a rule.

Theorem 2. *$Spik_{\underline{2}}^{\beta}P_*(rule_3^*, cons_3, forg_3, dley_0, outd_2) = NRE$, where either $\beta = h$ or β is omitted.*

Proof. This proof is based on that of Theorem 7.1 from [8], trying of imitate, as long as possible, the structure and functioning of modules ADD, SUB and FIN as well as of the dynamical register. In the proof of Theorem 7.1 [8], there exist two kinds of neurons using delays. The first one uses the delay just to slow down the emission of a spike, while the second one makes also use of the refractory period of the neuron. Remember that a neuron omits all spikes received during its refractory period. This property of neurons is then used to implement some desynchronizing circuits allowing, for instance, to decrement the dynamical register without using regular expressions that check the parity of spikes.

While eliminating delays in the first case is trivial (replacing the neuron by a chain of basic neurons with delay zero), it becomes a challenge to simulate the behavior of a neuron that makes use of its refractory period. If a neuron has delay 1 and it receives a spike in t, it fires in $t+1$, while remaining closed, and it finally spikes in $t+2$ (so it remains closed for one step). In turn, if a neuron with delay 2 receives a spike in t, it fires in $t+1$, remaining closed until $t+2$, and it finally emits a spike in $t+3$ (so it is closed during two steps).

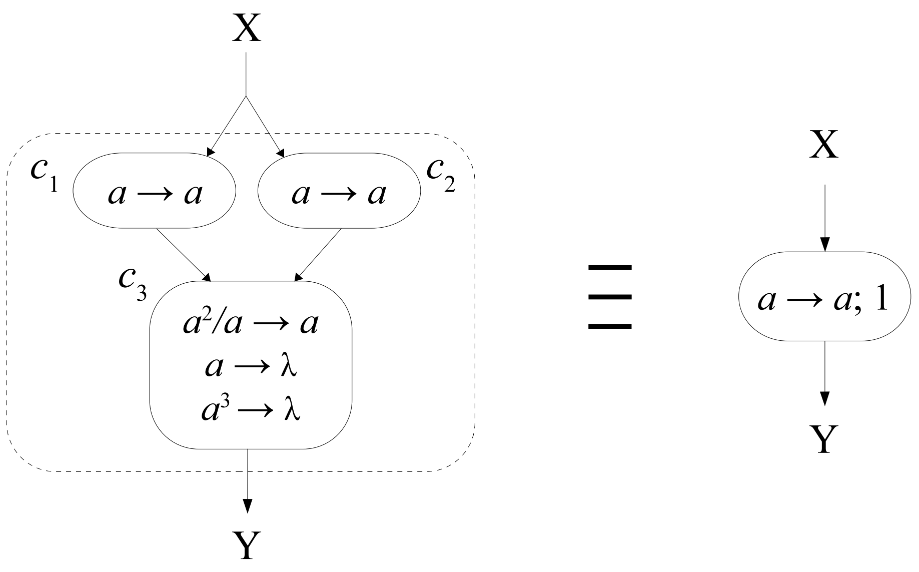

Fig. 4. Subsystem Π_{d_1} simulating a neuron with delay 1

Figure 4 shows a subsystem Π_{d_1} simulating the exact behavior of neurons with delay 1 which also use their refractory period. Let us consider X emits two spikes consecutively to Π_{d_1} in t and $t+1$. The spike received by c_1 and c_2 in t is emitted to c_3 in $t+1$ (meanwhile the second spike emitted by X reaches c_1 and c_2). Neuron c_3 fires, consuming just one of its two spikes, and spikes in $t+2$. In that moment c_1 and c_2 also spike to c_3 which now contains three spikes, which are forgotten in $t+3$ (using the rule $a^3 \rightarrow \lambda$). The reader can check that this system works also appropriately in the trivial case of X emitting just one spike in t (c_3 then uses the rule $a \to \lambda$).

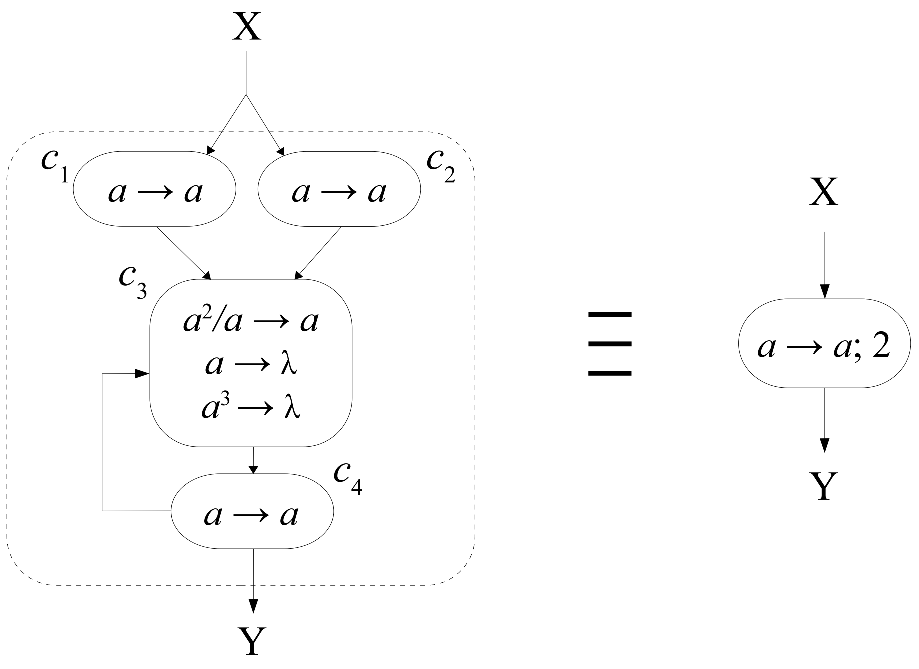

Fig. 5. Subsystem Π_{d_2} simulating a neuron with delay 2

Some more considerations have to be taken into account when simulating a neuron with delay 2 that has to remain closed during two steps. Figure 5 shows a subsystem that spikes at step $t+3$ (when it receives a spike in t) and omits any spike arriving at steps $t+1$ and/or $t+2$. Its function is analogous to that of Π_{d_1}.

We now present the dynamic register and the rest of modules where all the neurons having delays have been replaced, depending on the case, by either a chain of basic neurons or by one of the subsystems of type Π_{d_1} and Π_{d_2}. In the case of the dynamic register, neurons x, s and y do not use in any case their refractory period, so they are substituted by a trivial circuit of chained neurons with delay zero. On the other hand, neurons t and w have to be replaced by the subsystem of type Π_{d_1}. Finally, neuron r (which has delay two) is replaced by a subsystem of type Π_{d_2} as it needs to spike at every three steps whenever it contains any spike inside.

In the proof of Theorem 7.1 of [8] it is said that the dynamic register stores a number equal to the number of spikes that are continuously circulating in the close circuit $r - s - t - u$ (counting the pair of spikes simultaneously received and later emitted by s and t as one spike). In our case, there is a slight modification in the way of representing the number stored in the dynamic register. This is due to the structural effects of replacing some neurons by subsystems Π_{d_1} and Π_{d_2}. As the reader can see, these subsystems have two input neurons c_1 and c_2 (hence the input synapses have to be doubled). Then, the need to maintain the outdegree ≤ 2 forces us to replicate some cells (with their respective synapses) in order to keep the same functionality of the original dynamic register. Figure 6 shows the aspect of the dynamic register after introducing all these changes.

The reader can appreciate that one spike circulates from subsystem U to subsystem R (actually a group of 6 parallel spikes). It is easy to see that both r_1 and r_2 need to receive three spikes in order to have R behaving as expected (spiking every three steps as long as it holds some spike). Thus, to perform the ADD operation three spikes have also to be sent to both neurons r_1 and r_2. Subsystem R is connected to subsystems S and T, as well. Hence, when R spikes, one spike is sent simultaneously to neurons s_1, s_2, t_1 and t_2. Finally, a synapse also connects subsystems S and T with subsystem U. In this case, three spikes have to be sent from each S and T to feed the three equal neurons in U.

Hence, the computation in our dynamic register is cyclic every six steps when it stores the value $n = 1$ (all 6 spikes present at step t in R are consumed to make it spike once to S and T at $t+3$). In turn, S and T send three spikes (one to each neuron in U) at step $t+5$. Finally, U emits again six spikes to R at step $t+6$ and the cycle is complete. Moreover, similarly as in the former dynamic register, the six-step computation cycle actually consists of two identical halves of three steps, when the value stored is $n > 1$. Thus, the functioning of our dynamic register is identical to that of [8], if we consider as a single spike the group of spikes emitted simultaneously from U to R, if we count as another spike every pack of six spikes stored in R and, finally, if we also consider as one spike the two ones simultaneously received and later emitted by S and T.

Simulating an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ – module ADD (Figure 7).

To avoid the use of delays, we replace the former non-deterministic neuron c_4 by a new one c_3 containing three rules. At step one, neurons $l_i, l_i^{'}, l_i^{''}$ and $l_i^{'''}$ send one spike to c_1 and c_2 and a group of 6 spikes to the subsystem R of the dynamic register (incrementing by one the stored value). In the next step, two spikes reach neurons c_3 and c_4. Then, if rule $a^2/a \to a$ of c_3 is chosen, one spike is sent to c_6 and c_7 while another one still remains in c_3. In the following step, c_3 uses its rule $a \to a$ and two more spikes arrive to c_6 and c_7 (one from c_3 and another from c_5). This makes c_7 fire (leading the computation to l_k) while c_6 forgets its three spikes. On the other hand, if c_3 first chooses rule $a^2 \to a$, then it just emits one spike to c_6 and c_7 which will receive another one from c_5 in the next step. This situation makes c_6 fire (leading now the computation to l_j) while c_7 forgets its two spikes. Finally, it is easy to see that neurons c_8 and c_9 can be replaced by a chain of six basic zero-delay neurons.

Simulating a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ – module SUB (Figure 8).

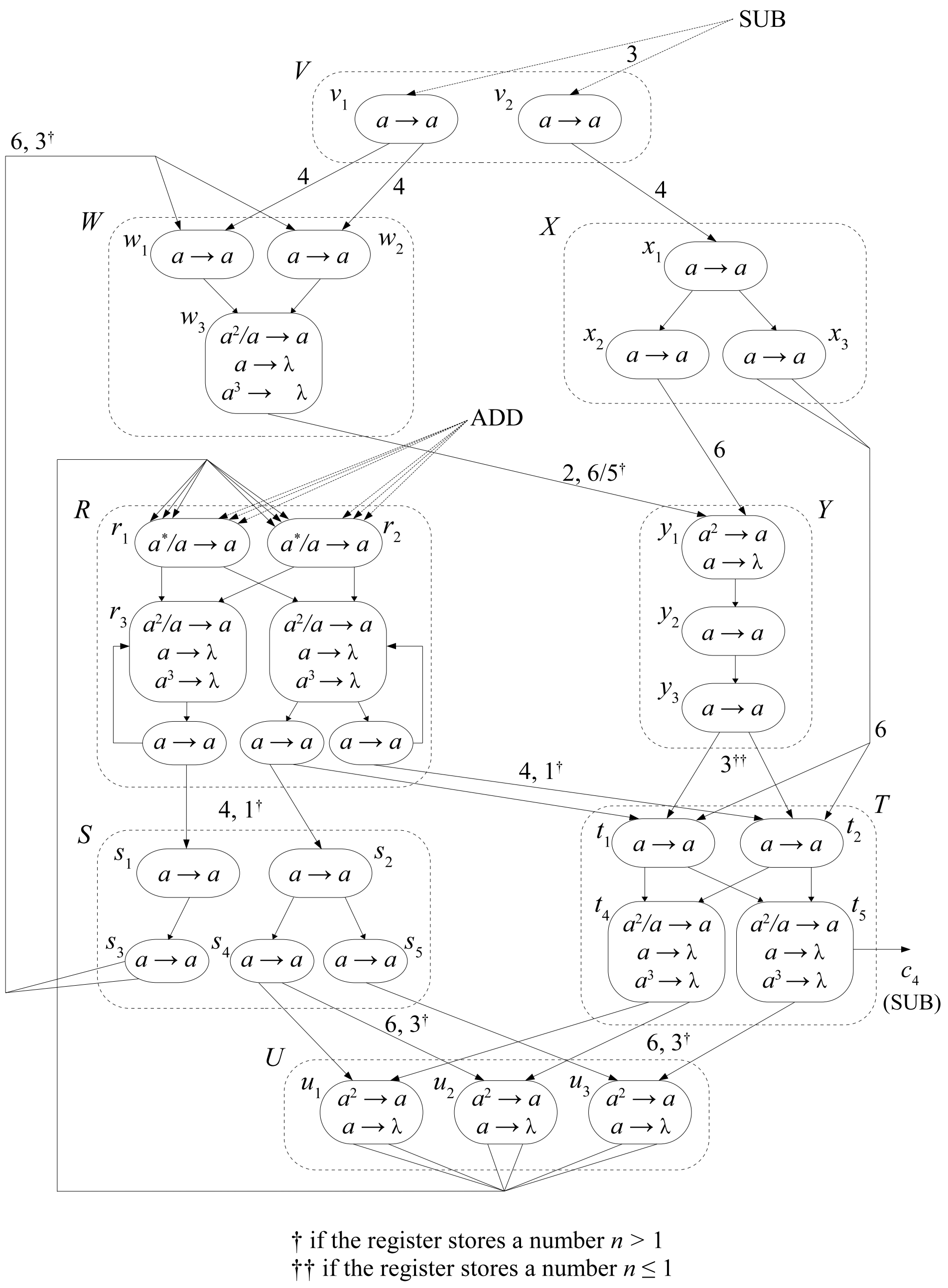

Fig. 6. A register with dynamical circulation of spikes without using delays

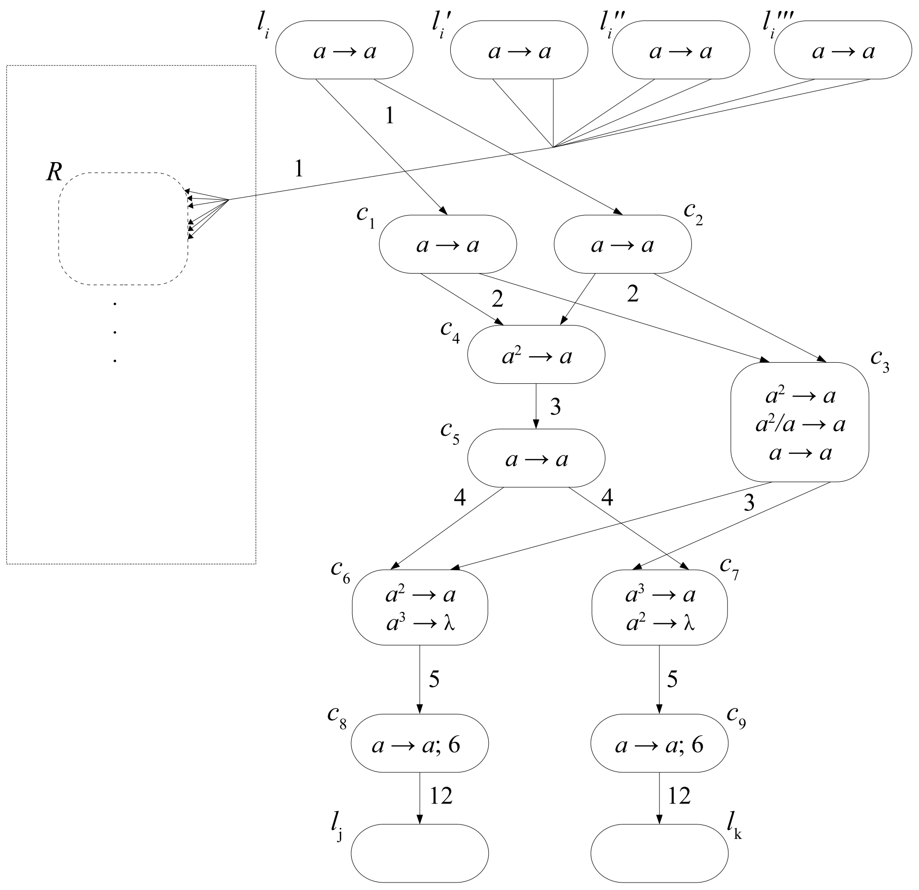

Fig. 7. Module ADD (simulating $l_i : (\mathtt{ADD}(r), l_j, l_k)$)

This module maintains the functioning of an analogous module of Theorem 7.1 with some structural changes. As neuron c_5 is the only one using the refractory period, it is replaced by a subsystem of type Π_{d_1}. The rest of neurons that have delays are replaced by a chain of basic neurons with delay 0 (except, for the sake of clarity, in the case of c_6). Some neurons are also replicated in order to maintain outdegree ≤ 2. This module is initiated when a spike is sent to neuron $l_i^{'}$. Then, two spikes are sent to subsystem V at step three and the de-synchronizing of the dynamic register starts, decrementing its value by 1. This forces T to sent a spike to c_4 at step 6. After that, $c_4^{''}$ spikes at step 8 and the computation continues by instruction l_j. If the register stored zero, then neuron $c_4^{''}$ do not spike at step 8 ($c_4^{'}$ forgets the spike emitted by c_1 at 6). Then, two spikes reach c_7 at step 11 and the computation continues by l_k. It is important to note that if there exists more than one instruction SUB decrementing the same register, then we would need more connections from T to the neurons c_4 corresponding to these instructions.

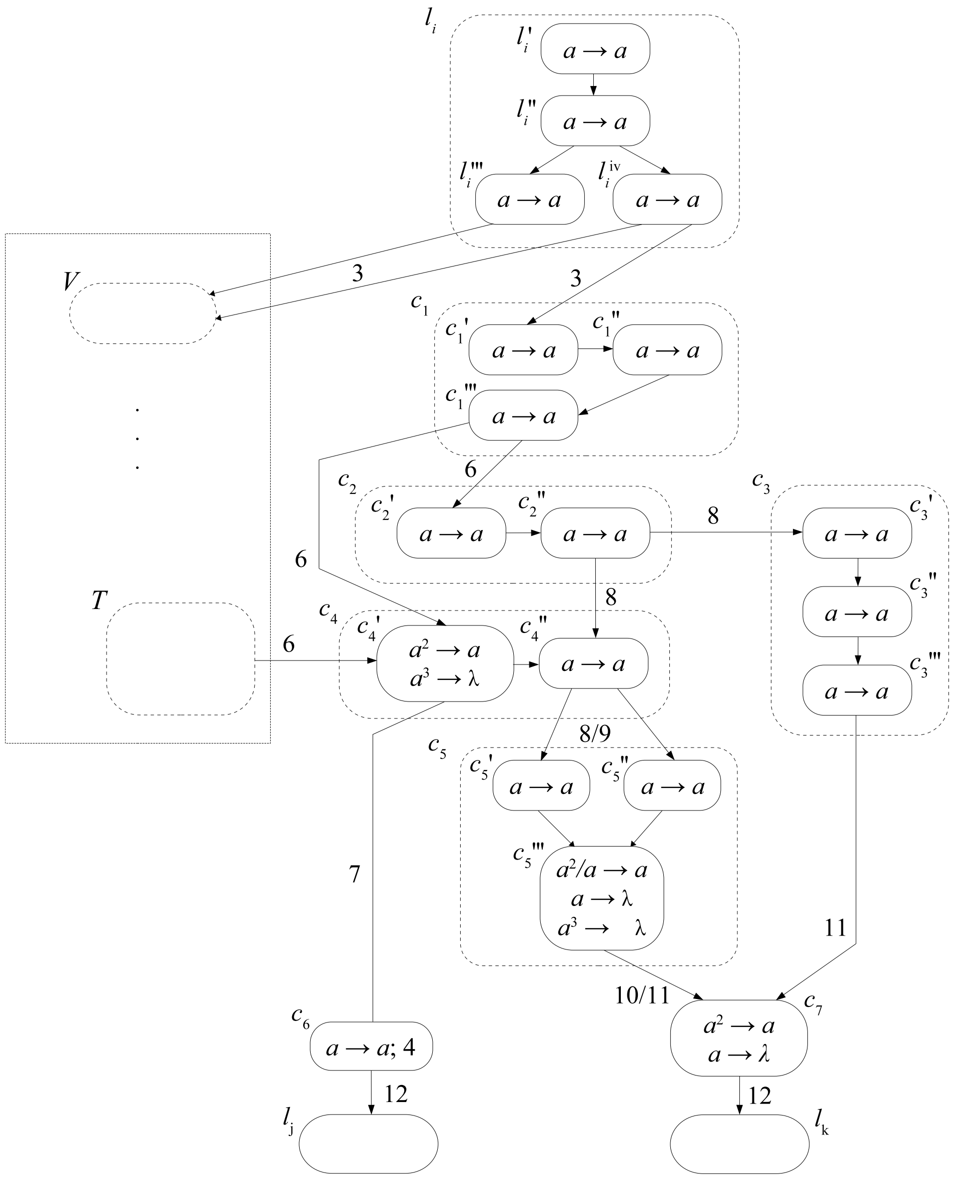

Fig. 8. Module SUB (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$)

Ending a Computation – module FIN (Figure 9).

The module FIN has also the same behavior as that of Theorem 7.1 in [8]. However, as in the case of module SUB, some structural changes have been made to eliminate the delays of neurons l_h, c_9, c_{10}, d_9 and d_{10}. On the one hand, both neurons c_9 and d_9 are substituted by two basic cells with delay zero. On the other hand, as neurons c_{10} and d_{10} make use of their refractory period, we replace each of them by one of our subsystems of type Π_{d_1}. Finally, neurons c_2, c_7, d_7, c_8 and d_8 are duplicated to keep outdegree ≤ 2. Again, for the sake of clarity, we do not replace c_3 with its corresponding four basic neurons. $\square$

5 Simplified Regular Expressions Revisited

Unlike some other results in [8], Theorem 7.1 mentioned above considered neither the case of strong halting, nor the case of accepting SN P systems. This sections extends this result and shows that it remains valid even if these additional restrictions are imposed. First we deal with the strong halting case.

Theorem 3. *$Spik_{\underline{2}}^{h} P_*(rule_2^*, cons_2, forg_2, dley_2, outd_2) = NRE$.*

Proof. Considering the strong halting case, the construction in the proof of Theorem 7.1 in [8] has to be changed slightly. Recall the assumption that all the registers except register 1 are empty at the end of computation. Under this assumption, one can verify by inspection of the above mentioned proof that the only neurons containing spikes at the moment of halting are in the module FIN. To remove these spikes, we have to release the additional restriction we stated in [8]: the rules of the form $a^r \to a; t$ and $a^s \to \lambda$ in the same neuron satisfy $s < t$. Removing this restriction allows to simplify dramatically the construction, while keeping all other properties of the normal form. The new module FIN which satisfies the strong halting condition can be found in Figure 10.

Function of the module FIN is described in Table 1. Assume that the output register 1 holds a value $n \geq 1$. Accordingly, the cycle consisting of neurons 1 and c_1 contains n spikes ($n-1$ in neuron 1 and one spike in neuron c_1). Both neurons fire at every step (except the case $n = 1$ which will be dealt with later.)

At step 1 neuron l_h receives spike and fires. At step 2 both neurons c_2 and c_3 receive spikes and start to fire at every step. Neuron c_4 receives at every step two spikes which are removed. From now on the number of spikes the cycle $1 - c_1$ decreases by one at every step. The output neuron *out* fires first time at step five. After $n + 2$ steps all the spikes in the cycle $1 - c_1$ are removed. At step $n + 3$ neuron c_2 receives no spike and does not fire. Consequently, at step $n + 4$ neuron c_4 receives only one spike and fires. Finally, at step $n + 5$, exactly n steps after its first firing, neuron *out* fires second time.

The case $n = 1$ needs a special attention. In this case neuron 1 fires at every even step and c_1 fires at every odd step. For a correct function of the module FIN, neuron l_h must receive spike at an odd step. However, this is already taken care

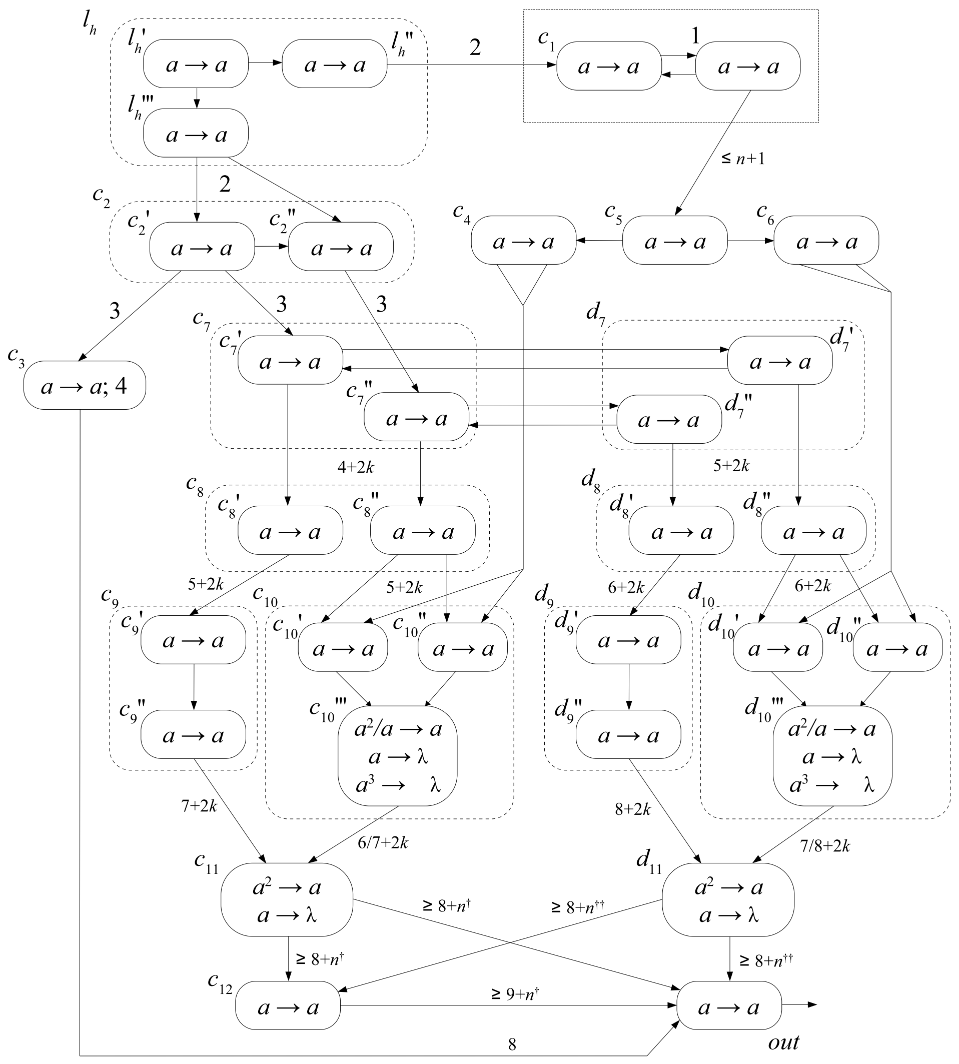

Fig. 9. Module FIN (ending the computation)

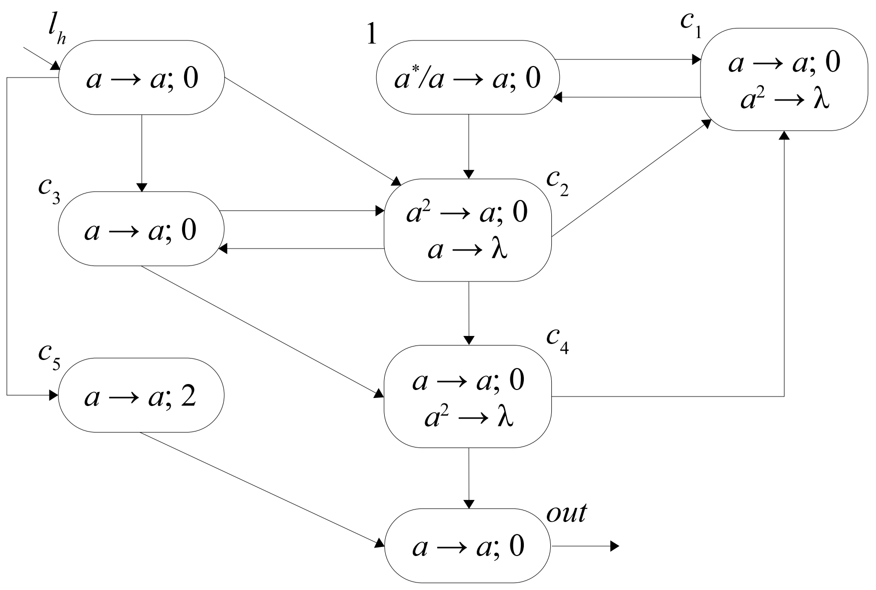

Fig. 10. Module FIN with simplified regular expressions and strong halting state

of in the proof of Theorem 7.1 in [8], as each instruction of the register machine is simulated in exactly 6 steps of the SN P system. Hence neuron l_h would receive spike at a step $6k+1$, $k \geq 0$, and the module FIN would function correctly again.

Finally, note that the module FIN of Figure 10 contains neurons with outdegree three. This version of the module was presented for the sake of simplicity, but it can be easily enhanced so that outdegree is reduced to two. The resulting diagram is shown in Figure 11. □

Another extension of the previous result in [8] mentioned above is the case of accepting SN P systems. In the accepting mode [2], the SN P system obtains an input in the form of an interval between two consecutive spikes sent from outside to the input neuron i_0. Therefore, we need a special module INPUT which translates this input value into the number of spikes present in a neuron labeled 1. Furthermore, the SN P system must behave deterministically. Both conditions can be satisfied and the resulting statement is given bellow.

Theorem 4. *$DSpik^{\beta}_{\underline{2}acc}P_*(rule^*_2, cons_2, forg_\alpha, dley_2, outd_2) = NRE$ where (i) $\beta = h$, $\alpha = 1$, or (ii) $\beta = \underline{h}$, $\alpha = 2$, or (iii) β is omitted and $\alpha = 1$.*

Proof. Considering the proof Theorem 7.1 in [8], we can observe that the module SUB is already deterministic. Then it remains only to "determinize" the module ADD and add a module INPUT. For the former goal, it is enough to remove the

Step Neuron	1	2	3	4	5	...	$n+1$	$n+2$	$n+3$	$n+4$	$n+5$
l_h	$a \to a$!	—	—	—	—	...	—	—	—	—	—
1	$a \to a$!	$a \to a$!	$a \to a$!	$a \to a$!	$a \to a$!	...	$a \to a$!	—	—	$a \to a$!	—
(spikes)	$n-1$	$n-1$	$n-1$	$n-2$	$n-3$	...	1	0	0	1	0
c_1	$a \to a$!	$a \to a$!	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	...	$a^2 \to \lambda$	$a^2 \to \lambda$	$a \to a$!	—	$a^2 \to \lambda$
c_2	—	$a^2 \to a$!	$a^2 \to a$!	$a^2 \to a$!	$a^2 \to a$!	...	$a^2 \to a$!	$a^2 \to a$!	$a \to \lambda$	$a \to \lambda$	$a \to \lambda$
c_3	—	$a \to a$!	$a \to a$!	$a \to a$!	$a \to a$!	...	$a \to a$!	$a \to a$!	$a \to a$!	—	—
c_4	—	—	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	...	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	$a \to a$!	—
c_5	—	$a \to a; 2$	—	!	—	...	—	—	—	—	—
out	—	—	—	—	$a \to a$!	...	—	—	—	—	$a \to a$!

Table 1. Function of the module FIN with strong halting state. Firing is denoted by !

rule $a \to a; 1$ from the neuron c_{i4} in the above mentioned module ADD, and the whole module becomes deterministic.

For the latter goal, one must construct a module INPUT which would fill-in register 1 with the number of spikes corresponding to the delay between two input spikes. However, as the module ADD works in a synchronized cycle of the length three, we have to send spikes to register 1 each three computational steps (or its multiple). Otherwise the spikes might be lost (consumed) within the module ADD. The module INPUT solving this task is presented in Figure 12.

One can observe that three steps after neuron i_0 receives the first input spike, neurons $c_3 - c_6$ start to fire at each step. Similarly, three steps after neuron i_0 receives the second input spike, neurons $c_3 - c_6$ stop firing and remove all their spikes. Therefore, neuron c_7 will receive exactly $3n$ spikes, where n is the period between the first and the second input spike. Neuron c_7 emits one spike at each step but neuron c_8 lets pass only each third spike. Therefore, neuron 1 corresponding to the input register receives spikes in steps 8, 11, 14..., and the number of spikes is exactly n.

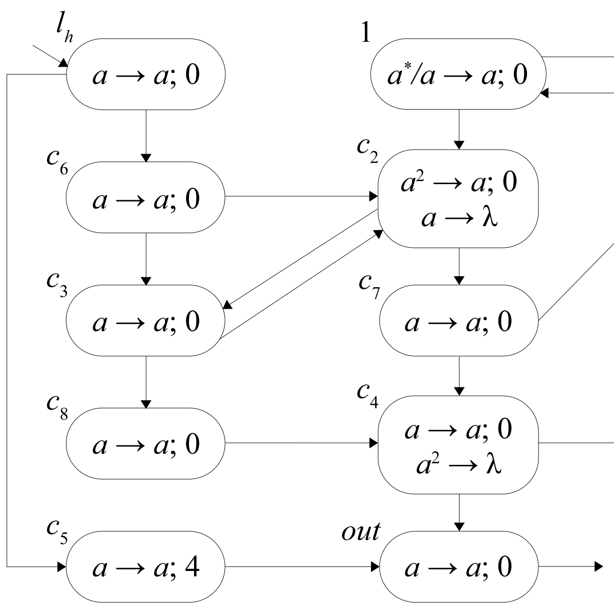

Fig. 11. Module FIN with simplified regular expressions and strong halting state, a version with outdegree two.

Finally, observe that there are no spikes left in neurons after finishing the computation. As this is also the case in the modules ADD and SUB described in the proof of Theorem 7.1 in [8] (provided that all the registers are empty), the system halts always in the strong halting state. If we do not require strong halting, the rules $a^2 \to \lambda$ in neurons $c_3 - c_6$ can be omitted. Therefore, the parameter *forg* is reduced to 1 in this case.

Therefore, the described SN P system correctly simulates a register machine in the accepting mode and the inclusion $NRE \subseteq DSpik^{\beta}_{\underline{2}\,acc}P_*(rule^*_2, cons_2, forg_\alpha, dley_2, outd_2)$. The converse inclusion follows by the Church-Turing thesis. □

6 Final Remarks

In this paper we have proven the universality of SN P systems even in the situations when we have eliminated more than one of its features simultaneously. Thus, this model has been found to be computationally complete 1) when using neither delays nor forgetting rules, 2) when simplifying regular expressions and eliminating delays, 3) when using simple regular expressions and the strong halting condition and 4) when using simple regular expressions in the accepting mode.

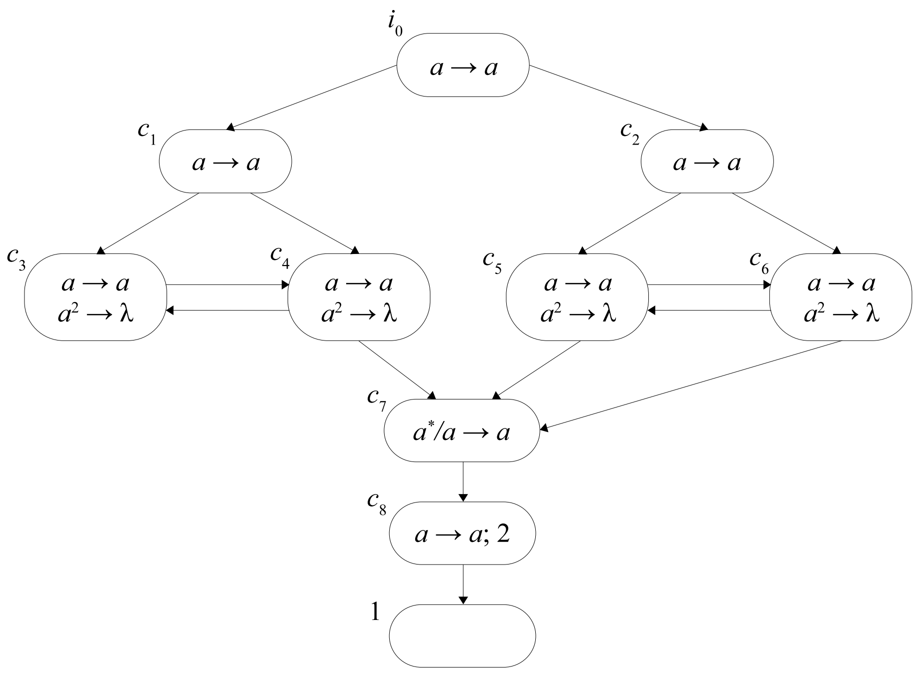

Fig. 12. The Module INPUT with simplified regular expressions.

We conjecture that in the cases 3) and 4), also delays could be removed without loss of computational universality. The case of simultaneously removing forgetting rules and simplifying regular expressions remains open but we conjecture that the universality would not be preserved in this case.

In all these results one can observe a trade-off between some other computational parameters, such as the number of neurons, the maximal number of firing rules per neuron, the complexity of regular expressions, the maximum number of spikes consumed in a firing rule or the maximal number of spikes removed in a forgetting rule. In all the above mentioned cases, however, the outdegree of neurons has been bounded by two.

It now remains an open problem whether these results concerning normal forms of SN P systems can be further improved. Would it be possible, for instance, to eliminate some more features of the model (three of them simultaneously) while keeping universality? If not, which would be the computational power of such a restricted model? Another open question would be, naturally, whether we can still achieve lower bounds for some of the computational parameters in our current proofs, as the number of rules in neurons, number of spikes consumed in one rule etc.

References

1. M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing, Febr. 2006*. Fenix Editora, Sevilla, 2006.
2. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
3. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
4. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
5. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
6. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
7. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002
8. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. In [1], Vol. II, 105-136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
9. Gh. Păun, Twenty Six Research Topics About Spiking Neural P Systems. In the present volume
10. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
11. The P Systems Web Page: `http://psystems.disco.unimib.it`.