Normal Forms for Spiking Neural P Systems

Oscar H. Ibarra[1], Andrei Păun[2,3], Gheorghe Păun[4],
Alfonso Rodríguez-Patón[3], Petr Sosik[3,5], Sara Woodworth[1]

[1] Department of Computer Science University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu, swood@cs.ucsb.edu
[2] Department of Computer Science, Louisiana Tech University, Ruston
PO Box 10348, Louisiana, LA-71272 USA
apaun@latech.edu
[3] Universidad Politécnica de Madrid - UPM, Faculdad de Informática
Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain
arpaton@fi.upm.es
[4] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es
[5] Institute of Computer Science, Silesian University
74601 Opava, Czech Republic
petr.sosik@fpf.slu.cz

Summary. The spiking neural P systems are a class of computing devices recently introduced as a bridge between spiking neural nets and membrane computing. In this paper we prove a series of normal forms for spiking neural P systems, concerning the regular expressions used in the firing rules, the delay between firing and spiking, the forgetting rules used, and the outdegree of the graph of synapses. In all cases, surprising simplifications are found, without losing the computational universality – sometimes at the price of (slightly) increasing other parameters which describe the complexity of these systems.

1 Introduction

The spiking neural P systems (in short, SN P systems) were recently introduced in [2], and then investigated in [7] and [8], thus incorporating in membrane computing [6] ideas from spiking neurons, see, e.g., [1], [3], [4].

In short, an SN P system consists of a set of neurons placed in the nodes of a graph, representing synapses. The neurons send signals (spikes) along synapses (edges of the graph). This is done by means of firing rules, which are of the form $E/a^c \to a; d$, where E is a regular expression, c is the number of spikes consumed

by the rule, and d is the delay from firing the rule and emitting the spike. The rule can be used only if the number of spikes collected by the neuron is "covered" by expression E, in the sense that the current number of spikes in the neuron, n, is such that $a^n \in L(E)$, where $L(E)$ is the language described by expression E. In the interval between firing a rule and emitting the spike, the neuron is closed/blocked, it does not receive other spikes and cannot fire again. There also are rules for forgetting spikes, of the form $a^s \to \lambda$ (s spikes are just removed from the neuron). Starting from an initial distribution of spikes in the neurons and using the rules in a synchronized manner (a global clock is assumed), the system evolves. A sequence of transitions among configurations of an SN P system, starting in the initial configuration, is called a computation. One of the neurons is designated as the output neuron and its spikes can also exit the system. The sequence of steps when the output neuron sends spikes to the environment is called the spike train of the computation.

An SN P system can be used as a computing devices in two main ways: as a number generator and as a generator and transducer of infinite sequences of bits. In the first case, considered in [2] and [7], one associates a set of numbers with a spike train in various ways: considering the distance between the first k spikes of a spike train, or the distances between all consecutive spikes, in both cases taking into account all intervals or only considering intervals that alternate (ignoring every second one), accepting only halting or only infinite computations. In this last case, one can naturally associate an infinite binary sequence with a spike train by writing 0 for a step when no spike exits the system and 1 for a step when a spike is emitted by the output neuron.

In the first interpretation of SN P systems, as devices computing sets of natural numbers, it was proven in [2], [7] that Turing completeness is achieved if no bound is imposed on the number of spikes present in the neurons, and a characterization of semilinear sets of numbers is obtained if a bound is imposed on the number of spikes present in neurons during a computation.

In the proofs of these results, all features of the SN P systems as briefly introduced above were used: regular expressions describing languages different from a^*, delays d different from 0, forgetting rules, while the synapse graphs of the systems involved in the proofs have outdegree four. The question of improving these proofs from these points of view was formulated as a research topic in the papers [2], [7]. We contribute here to this topic with several results, some of them surprising: we prove universality: (i) with regular expressions of the form $E = a^+$ (hence telling nothing else about the number of spikes from the neuron other than the fact that some spikes do exist) or of the form a^i for some $i \geq 1$, (ii) rules without delay, i.e., of the form $E/a^c \to a; 0$, and (iii) without using forgetting rules (this last result solves an open problem from [2], asking whether the universality is preserved even when forgetting rules are not used – the answer proves to be affirmative). We do not know whether these normal forms can be combined: in the proofs of (i) and (iii) we use delays, in the proofs of (ii) and (iii) we use non-trivial regular expressions, while in the proofs of (i) and (ii) we use forgetting rules. What can be

combined with all these three normal forms is the next condition: each neuron has only two outgoing synapses (hence the synapse graph has the outdegree two). This is of a clear interest in our framework, because the spikes in an SN P system can be increased only by means of multiple outgoing synapses – the result mentioned above shows that the minimal outdegree suffices.

In the next section we introduce a few technical prerequisites, then (Section 3) we recall from [2], [7] the definition of spiking neural P systems and fix the notation we use. In Section 4 we give the result about the possibility of working with no delay, the next section (Section 6) describes the power of systems that are not allowed to use forgetting rules, then (Section 5) we also bound the outdegree of SN P systems without losing the universality. Section 7 gives the normal form about the regular expressions from the firing rules. The paper ends with some open problems and research topics discussed in Section 8.

2 Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as with basic membrane computing, e.g., from [9] and [6], respectively (we also refer to [10] for the most updated information about membrane computing), so that we introduce here only some notations and the notion of register machines, used later in proofs.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The length of a string $x \in V^*$ is denoted by $|x|$.

The family of Turing computable sets of natural numbers is denoted by NRE and the family of semilinear sets of natural numbers is denoted by $NREG$ (they are the families of length sets of recursively enumerable languages and of regular languages, respectively, hence the notations).

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:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$ (add 1 to register r and then go to one of the instructions with labels l_2, l_3),
- $l_1 : (\texttt{SUB}(r), l_2, l_3)$ (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),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes 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 (see, e.g., [5]) that register machines (even with a small number of registers, but this detail is not relevant here) compute all sets of numbers which are Turing computable, hence they characterize NRE.

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

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 $l_1 : (\texttt{ADD}(r), l_2, l_3)$ with $l_2 = l_3$: after adding 1 to register r we pass precisely to one instruction, without any choice (in such a case, the instruction is written in the form $l_1 : (\texttt{ADD}(r), l_2)$).

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

We close this section by establishing the following **convention**: when evaluating or comparing the power of two number generating/accepting devices, we ignore the number zero; this corresponds to a frequently made convention in grammars and automata theory, where the empty string λ is ignored when comparing two language generating/accepting devices.

3 Spiking Neural P Systems

The neural motivation for introducing spiking neural P systems can be found in [2], here we pass directly to recalling the definition.

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

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

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

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

Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the *total* number of spikes contained in the neuron. Thus, e.g., if a neuron σ_i contains 5 spikes, and R_i contains the rules $(aa)^*/a \to a; 0$, $a^3 \to a; 0$, $a^2 \to \lambda$, then none of these rules can be used: a^5 is not in $L((aa)^*)$ and not equal to a^3 or a^2. However, if the rule $a^5/a^2 \to a; 0$ is in R_i, then it can be used: two spikes are consumed (thus three remain in σ_i), and one spike is produced and sent immediately ($d = 0$) to all neurons linked by a synapse to σ_i, and the process continues.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron. During a computation, the "state"

of the system is described by both by the number of spikes present in each neuron, and by the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again.

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

In the spirit of spiking neurons, as the result of a computation, in [2] one takes the number of steps between two spikes sent out by the output neuron, and, for simplicity, one considers as successful only computations whose spike trains contain exactly two spikes. This has been generalized in [7], where several ways of defining a set of numbers associated with a spike train were systematically examined. We recall from [7] several relevant definitions.

Let $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ be an SN P system and let γ be a computation in Π, $\gamma = C_0 \Longrightarrow C_1 \Longrightarrow C_2 \Longrightarrow \ldots$ (C_0 is the initial configuration, and $C_{i-1} \Longrightarrow C_i$ is the ith step of γ). In some steps a spike exits the (output neuron of the) system, in other steps it does not. The *spike train* of computation γ is the sequence of steps i when the output neuron σ_{i_0} emits a spike. We denote by $st(\gamma)$ the sequence of emitting steps, and we write it in the form $st(\gamma) = \langle t_1, t_2, \ldots \rangle$, with $1 \leq t_1 < t_2 < \ldots$. The sequence can be finite (this happens if the computation halts, or if it sends out only a finite number of spikes) or infinite (then, of course, the computation does not halt).

One can associate a set of numbers with Π in several ways. We follow here the idea of [2] (with the extension from [7]), and we consider the intervals between consecutive spikes as numbers computed by a computation, with several alternatives (by $COM(\Pi)$ we denote the set of all computations in Π):

- Taking into account only the first two spikes:

$$N_2(\Pi) = \{t_2 - t_1 \mid st(\gamma) = \langle t_1, t_2, \ldots \rangle, \gamma \in COM(\Pi)\}.$$

- Generalizing to the first $k \geq 2$ spikes:

$$N_k(\Pi) = \{n \mid n = t_i - t_{i-1}, \text{ for } 2 \leq i \leq k, st(\gamma) = \langle t_1, t_2, \ldots \rangle, \\ \gamma \in COM(\Pi), \text{ and } st(\gamma) \text{ has at least } k \text{ spikes}\}.$$

- Taking into account all spikes of computations with infinite spike trains:

$$N_\omega(\Pi) = \{n \mid n = t_i - t_{i-1}, \text{ for } i \geq 2, \gamma \in COM(\Pi), \\ st(\gamma) = \langle t_1, t_2, \ldots \rangle \text{ infinite}\}.$$

- Taking into account all intervals of all computations:

$$N_{all}(\Pi) = \bigcup_{k \geq 2} N_k(\Pi) \cup N_\omega(\Pi).$$

For $N_k(\Pi)$ we can consider two cases, the *weak* one, where, as above, we take into consideration all computations having *at least* k spikes, or the *strong* case, where we take into consideration only the computations having *exactly* k spikes. In the strong case we underline the subscript k, thus writing $N_{\underline{k}}(\Pi)$ for denoting the respective set of numbers computed by Π.

Two subsets of (some of) these sets are also of interest (the strong halting case is newly introduced):

- Taking only *halting* computations; this makes sense only for $N_k(\Pi), k \geq 2$, and for $N_{all}(\Pi)$ – the respective subsets are denoted by $N_k^h(\Pi)$ and $N_{all}^h(\Pi)$, respectively.
- Considering *strong halting* computations: halting computations as described above, with the extra condition that when the system halts, no spike is present in the whole system. The respective sets of numbers will be denoted by $N_{\underline{k}}^h(\Pi)$ and $N_{\underline{all}}^h(\Pi)$.
- Considering *alternately* the intervals:

$$N^a(\gamma) = \{n \mid n = t_{2k} - t_{2k-1}, \text{ for } k \geq 1, \gamma \in COM(\Pi), \text{ and } st(\gamma) = \langle t_1, t_2, \ldots \rangle\}.$$

  This means that every second interval is "ignored", we take the first one, we skip the second interval, we take the third, we skip the fourth interval, and so on. This strategy can be used for all types of sets, hence we get $N_k^a(\Pi), N_\omega^a(\Pi), N_{all}^a(\Pi)$, as subsets of $N_k(\Pi), N_\omega(\Pi), N_{all}(\Pi)$, respectively.

Finally, we can combine the halting restriction with the alternate selection of intervals, obtaining the sets $N_\alpha^{ha}(\Pi)$ and $N_{\underline{\alpha}}^{ha}(\Pi)$, for all $\alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\}$, as well as $N_{\underline{k}}^{ha}(\Pi)$ and $N_{\underline{\underline{k}}}^{ha}(\Pi)$, for $k \geq 2$.

We do not illustrate here these definitions with examples of SN P systems, but several explicit constructions will be found in the subsequent sections – the reader can find many examples in [2], [7], and [8].

As in these papers, we denote by $Spik_\alpha^\beta P_m(rule_k, cons_p, forg_q)$ the family of sets $N_\alpha^\beta(\Pi)$, 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, and each forgetting rule removing at most q spikes; then, $\alpha \in \{all, \omega\} \cup \{k, \underline{k} \mid k \geq 2\}$, and β is either omitted or it belongs to the set $\{h, a, ha, \underline{h}, \underline{ha}\}$. As usual, a parameter m, k, p, q is replaced with $*$ if it is not bounded.

In the above notation, we add to the list of features mentioned between parentheses the following two: $dley_r$, meaning that we use SN P systems whose rules $E/a^c \to a; d$ have $d \leq r$ (the delay is at most r), and $outd_s$, meaning that the outdegree of the synapse graph has the outdegree at most s. We also write $rule_k^*$ if the firing rules are of the form $a^+/a^c \to a; d$ or of the form $a^c \to a; d$.

With these notations, the universality result from [2] can be written as:

Theorem 1. *$Spik_{\underline{2}}^{\beta}P_*(rule_2, cons_3, forg_3, dley_1, outd_4) = NRE$, where either $\beta = h$ or β is omitted.*

Similar results were proven in [7] for all families $Spik_{\alpha}^{\beta}P_*(rule_k, cons_p, forg_q, dley_1, outd_4)$, with various parameters k, p, q, but always with the delay 0 or 1, and the outdegree four. In the next sections we improve the result in Theorem 1 from the point of view of $forg$, $dley$, and $outd$, then, in Section 7 we also simplify the regular expressions used in the universality proof.

4 Removing the Delay

We imitate here the proof of Theorem 1 from [2], with an additional care paid to the delay from firing to spiking. Because all rules we use have the delay 0, we write them in the simpler form $E/a^c \to a$, hence omitting the indication of the delay. The price of the elimination of the delay will be the slight increase in the number of neurons and of other parameters (the number of rules from each neuron, of spikes consumed for firing, and of spikes forgotten by each rule). Then, we also bound the outdegree of the system, to two. Although this can be done at the same time with the removing of the delay, we do not pay attention here to the outdegree, because we want to make explicit the (simple) technique used in that case, adding in this way new items to the "tool-kit" used in previous papers, especially in [8], for handling SN P systems and their spike trains.

In the proof below we present the SN P system used (actually, its modules) in a way already proposed in [2]: neuron-membranes placed in the nodes of a graph, with the edges representing the synapses, and with an arrow pointing from the output neuron to the environment; inside neurons, we give the rules and the initial number of spikes.

Theorem 2. *$Spik_{\underline{2}}^{\beta}P_*(rule_3, cons_4, forg_q, dley_0, outd_*) = NRE$, where $\beta \in \{h, \underline{h}\}$ or β is omitted, and $q = 5$ for $\beta = \underline{h}$, otherwise $q = 4$.*

Proof. In view of the Turing-Church thesis (the inclusion in NRE can also be proved directly), we only have to prove the inclusion $NRE \subseteq Spik_{\underline{2}}^{\beta}P_*(rule_3, cons_4, forg_q, dley_0, outg_*)$.

Let $M = (m, H, l_0, l_h, I)$ be a register machine generating a set $N(M)$, having the properties specified in Section 2: the result of a computation is the number from register 1, this register is never decremented during the computation, and the machine halts with all registers $2, 3, \ldots, m$ empty.

We construct a spiking neural P system Π as in [2], simulating the register machine M and spiking only twice, at an interval of time which corresponds to a number computed by M. The system Π will be presented graphically, through modules which simulate the ADD and SUB instructions of M; there also is a FIN module, which takes care of the spiking of the system Π.

These three (types of) modules are given in Figures 1, 2, 3, respectively. The neurons appearing in these figures have labels $l_1, l_2, l_3, \ldots$, as in the instructions

from I, labels $1, 2, \ldots, m$ associated with the m registers of M, as well as a series of labels which we do not specify here, but we only mention that they are supposed to be distinct to each other, so that no "illegal" interference of modules is possible; the output neuron is labeled with *out* in the module FIN.

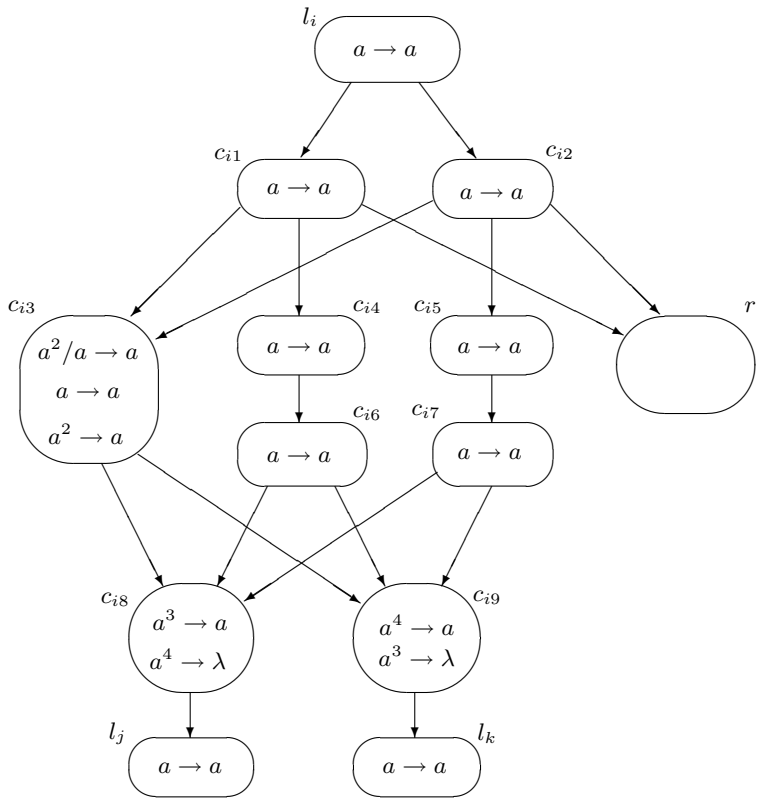


Fig. 1. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

In the initial configuration, there is only one spike in the system, in the neuron with label l_0, the initial label of M. During the computation, the contents of register $r, 1 \leq r \leq m$, will be encoded by the number of spikes from neuron r in the following way: if register r holds the number n, then neuron r will contain $2n$ spikes.

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

The initial instruction of M, the one with label l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction

l_i : (ADD$(r), l_j, l_k$), with one spike present in neuron l_i (like in the initial configuration) and no spike in any other neuron, except those neurons associated with the registers. Neuron l_i fires and sends its spike to neurons c_{i1} and c_{i2}. These neurons fire, and from them both the neuron r (the one associated with the register involved in the instruction we simulate) and the "non-deterministic" neuron c_{i3} receive two spikes, while the synchronizing neurons c_{i4}, c_{i5} receive one spike each. Thus, neuron r has increased its contents as needed. In Figure 1, this neuron contains no rules, but, as we will see immediately, the neurons associated with registers have two rules each, used when simulating the SUB instructions, but both these rules need an odd number of spikes to be applied (this is true also for the module FIN, which only deals with the neuron associated with register 1). Therefore, during the simulation of an ADD instruction, neuron r just increases by 2 its contents, and never fires.

We have now to pass non-deterministically to one of the instructions with labels l_j and l_k, and this is done with the help of neuron c_{i3}, which contains two rules which can be applied to the two spikes it contains. If the rule $a^2 \to a$ is used, then both its spikes (that the neuron has at the moment) are consumed, and then the neurons c_{i8}, c_{i9} will have one spike each; this spike has to wait unchanged until the spikes from intermediate neurons c_{i6}, c_{i7} arrive. These neurons send two spikes to neurons c_{i8}, c_{i9}, hence we have here 3 spikes. With three spikes inside, only neuron c_{i8} fires, while c_{i9} forgets the spikes. In this way, neuron l_j receives a spike, and it is "activated".

If instead of rule $a^2 \to a$, neuron c_{i3} uses the rule $a^2/a \to a$, then only one spike is consumed, one spike reaches immediately each neuron c_{i8}, c_{i9} and a second one arrives in c_{i8}, c_{i9} one step later (when the rule $a \to a$ of neuron c_{i3} is used), hence, together with the spikes of neurons c_{i6}, c_{i7} (which arrive at the same time as the last spike from c_{i3}) we have now four spikes in each neuron c_{i8}, c_{i9}. This makes possible the firing of neuron c_{i9} only, which implies the "activation" of neuron l_k, which receives a spike.

The simulation of the ADD instruction is correct: we have increased the number of spikes in neuron r by two and we have passed to one of the neurons l_j, l_k non-deterministically.

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

Let us examine now Figure 2. We start with a spike in neuron l_i and no spike in other neurons, except neuron r, which holds an even number of spikes (half of this number is the value of the corresponding register r). The spike of neuron l_i goes immediately to three neurons, c_{i1}, c_{i2}, and r. Neurons c_{i1}, c_{i2} will send in the next step a spike to neurons c_{i3} and c_{i4}, while neuron r will send a spike to neuron c_{i3} only if it contains more than one spike.

Indeed, neuron r contains now an odd number of spikes. If the only spike it holds is the one sent by l_i, then this spike will be forgotten and no spike is produced. This means that the register r was empty. The neurons c_{i3}, c_{i4} receive one spike each (from c_{i1} and c_{i2}, respectively). While neuron c_{i4} fires, and sends

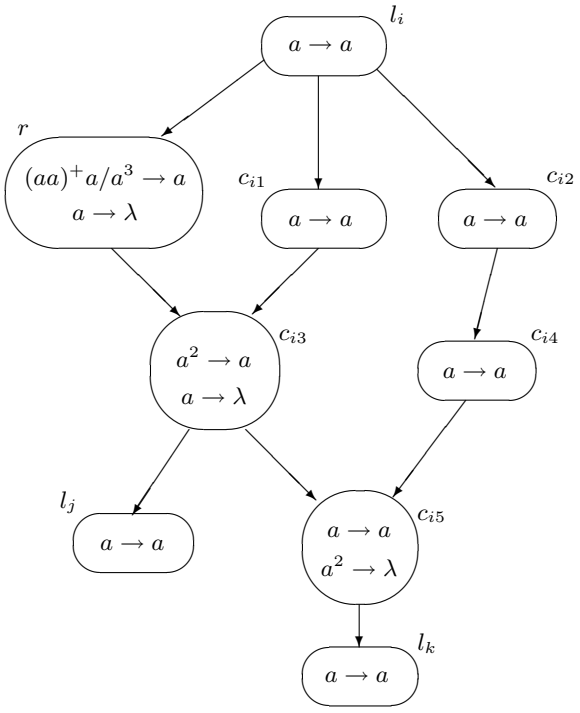


Fig. 2. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

a spike to neuron c_{i5}, neuron c_{i3} forgets the spike. Now, neuron c_{i5} fires and its spike is sent to neuron l_k, which is thus "activated".

If neuron r contains at least three spikes, hence the register r is not empty, then we have to use the rule $(aa)^+a/a^3 \to a$, which decreases the number of spikes from neuron r to an even value while removing three spikes (which corresponds to decrementing the register r and removing the spike received from l_i). The spike of neuron r arrives in neuron c_{i3} at the same time with the spike of neuron c_{i1} (while, at the same time, the spike of neuron c_{i2} arrives in neuron c_{i4}). With two spikes inside, neuron c_{i3} fires. Its spike reaches both neurons l_j – which is thus "activated", and neuron c_{i5}. Neuron c_{i5} contains now two spikes, because it has also received one from c_{i4}; it forgets them by using the rule $a^2 \to \lambda$, hence no spike is emitted here (l_k remains empty).

The simulation of the SUB instruction is correct, we started from l_i and we ended in l_j if the register was non-empty and decreased by one, and in l_k if the register was empty.

Note that there is no interference between the neurons used in the ADD and the SUB instructions, other than correctly firing the neurons l_j, l_k which may

label instructions of the other kind (the ADD instructions do not use any rule for handling the spikes of neurons $1, 2, \ldots, m$ associated with the registers of M). However, there is an interference between SUB modules, because each neuron r associated with a register which is subject of a SUB instruction sends a spike to all neurons with label c_{i3} in a module SUB as that from Figure 2; however, all these neurons will immediately forget this spike with one exception, of the neuron c_{i3} from the module of the SUB instruction whose simulation proceeds correctly, and which also receives one spike from the corresponding neuron c_{i1}. It is also worth noting here that register 1 is never decremented, hence for it there is no SUB module as above.

Ending a computation – module FIN (Figure 3).

Assume now that the computation in M halts, which means that the halting instruction is reached. For Π this means that the neuron l_h receives a spike. At that moment, neuron 1 contains $2n$ spikes, for n being the contents of register 1 of M (and all other neurons $2, 3, \ldots, m$ are empty). The spike of neuron l_h is sent to four neurons, 1, c_{i1}, c_{i3}, and c_{i4} from Figure 3. In this way, neuron 1 accumulates an odd number of spikes, and it can fire. It is important to remember that this neuron was never involved in a SUB instruction, hence it does not contain any rules as described in Figure 2. Thus, the only rules available in neuron 1 are the ones defined at this step.

Let 1 be the moment when neuron l_h fires. The spike sent to neuron d_1 passes to neuron d_2 and then to the output neuron, which thus spikes at step 4.

Let us now follow the other spikes emitted by neuron l_h. Neurons d_3, d_4 form a pair of self-sustaining neurons, spiking to each other in each step; at the same time, each of them sends one spike to neuron d_5. If this neuron also receives a spike from neuron 1, then it has to forget the three spikes. In turn, at each step when spiking, neuron 1 consumes two spikes, which corresponds to decreasing by one the value of register 1.

These operations continue until exhausting the spikes from neuron 1; in the last step when neuron 1 fires, we have to use the rule $a^3 \to a$, and this means that neuron 1 can fire n times, for n being the number stored by register 1 of M in the end of the computation. Therefore, the last time when neuron 1 fires is in step $n + 1$ (one step was necessary initially, for firing neuron l_h).

In step $n + 2$, the first step when neuron 1 does not fire, neurons d_3, d_4 fire again, their spikes arrive in neuron d_5, and, with only two spikes inside, this neuron fires (step $n + 3$). Its spike goes at the same time to neuron d_6 and to the output neuron, and both these neurons fire. This is step $n+4$, hence the distance between the two spikes of neuron *out* is n, the contents of register 1 of M.

In step $n+3$, neurons d_3, d_4 fire again, hence two spikes reach neuron d_5, which spikes again (step $n+4$). Its spike reaches the output neuron at the same time with the spike of neuron d_6, hence, with two spikes inside, neuron *out* can never spike again. The spike of neuron d_6 also reaches neuron d_4, which holds now two spikes (it has one from the partner neuron d_3), hence also this neuron will never spike again. Neuron d_3 spikes once more, and this is the last step of the computation:

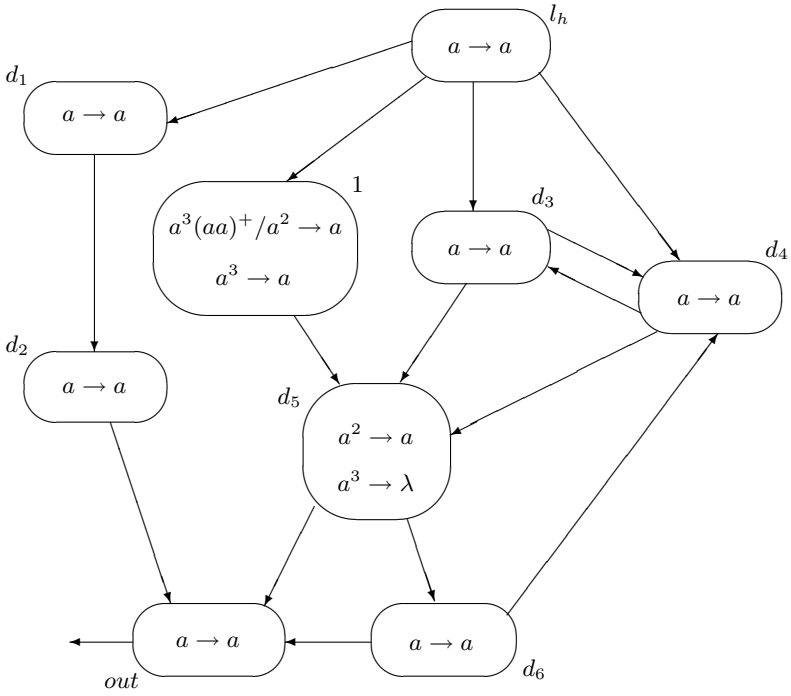


Fig. 3. Module FIN (ending the computation)

neuron d_5 cannot fire with only one spike inside, while neuron d_4 contains already three spikes.

It is now clear that the previous system halts and this is done using forgetting rules of size 4 (in module ADD).

We can modify the previous construction to reach also a strong halting state. First, let us note that the ADD and SUB modules do not leave any spikes in their neurons. Because the register machine is assumed to stop with all registers empty, except register 1, when reaching the FIN module, we will have only spikes stored in the output register 1 and one spike in the neuron l_h. Starting from this configuration, the system halts at step $n+7$ (counting as step 1 the moment when neuron l_h fires). At that time, *out* contains five spikes (three from d_6 and two from d_5), d_4 has also five spikes (three from d_6 and two from d_3), and d_5 has one spike (from d_3). We add the rule $a^5 \to \lambda$ to neurons *out* and d_4, and the rule $a \to \lambda$ to neuron d_5. Thus, at the step $n+7$ there will be no spike in the whole system. Consequently, $N_2(\Pi) = N_2^h(\Pi) = N_{\underline{2}}^{\underline{h}}(\Pi) = N(M)$ and this completes the proof (the outdegree of the system is not bounded: a neuron r corresponding to a register has a synapse (r, c_{i3}) for each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$).

Besides the fact that in the ADD module we have used firing rules which consume four spikes, and that we also have forgetting rules which remove four spikes, the above construction is also more complex than the one from [2] in what concerns the number of neurons: the ADD module in [2] contained 8 neurons, here we use 13, while in the SUB case we use 9 neurons instead of 6; the FIN module uses the same number of neurons in [2] and in the proof above, 9, but the construction from Figure 3 has a more transparent functioning than that from [2].

In what concerns the number of neurons which behave non-deterministically, it is also possible to have lost here the nice result from [2], that one neuron with a non-deterministic behavior in the whole system is sufficient: for the moment, we do not see a way to use only one such neuron (like c_3 in Figure 1) for all ADD modules.

However, the previous construction holds for the case when we want to define as the result of a computation the number of spikes collected by the output neuron in the end of halting computations (when dealing with such a number, we need to consider halting computations, otherwise we do not know when the computation of a number is completed). The changes in the module FIN are the same as in [2].

Moreover, the previous normal form also holds in the case of accepting SN P systems.

Like in [2], we consider the following way of introducing into a system the number to be accepted, again in the spirit of spiking neurons, with the time as the main data support: the special neuron i_0 is used now as an input neuron, which can receive spikes from the environment of the system (in the graphical representation an incoming arrow will indicate the input neuron); we assume that exactly two spikes are entering the system; the number n of steps elapsed between the two spikes is the one analyzed; if, after receiving the two spikes, the system halts (not necessarily in the moment of receiving the second spike), then the number n is accepted.

In the accepting mode we can impose the restriction that in each neuron, in each time unit at most one rule can be applied, hence that the system behaves deterministically. A counterpart of Theorem 2 is then true (the notation of the respective families are obvious):

Theorem 3. *$DSpik^{\beta}_{\underline{2}acc}P_*(rule_2, cons_3, forg_2, dley_0, outd_*) = NRE$, where $\beta \in \{h, \underline{h}\}$ or β is omitted.*

Proof. The theorem is a consequence of the proof of Theorem 2. This time we start from a deterministic register machine M and we construct the SN P system Π as in the proof of Theorem 2, with no module FIN (the neuron l_h has no rule inside), with the same module SUB, with a simpler module ADD (see Figure 5), as well as with a further module, called INPUT, which takes care of initializing the work of Π. This time Π has initially no object inside, and the same is true with the new module.

The module INPUT is given in Figure 4 and it is a simplification of the similar module from [2], due to the fact that now we have only one spike in neurons

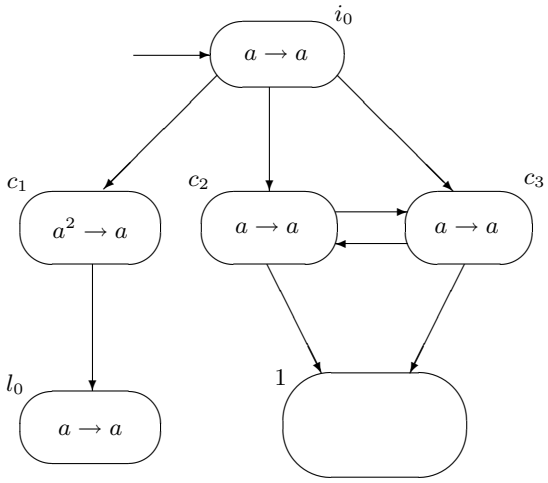


Fig. 4. Module INPUT (initializing the computation)

with labels $l \in H$. The functioning of this module is obvious: the first input spike triggers the self-sustaining neurons c_2, c_3, which will send pairs of spikes to neuron 1 until having the second spike entering the system; at that time, neurons c_2, c_3 stop, because they cannot handle two spikes at the same time. In turn, neuron l_0 gets a spike only after introducing in the system two spikes (the first one just waits in neuron c_1).

Now, we start using modules ADD and SUB associated with the register machine M, with modules ADD constructed for instructions of the form $l_i : (\texttt{ADD}(r), l_j)$. This means that the module ADD is now much simpler than in Figure 1, namely, it looks like in Figure 5.

The functioning of this modified ADD module is obvious, hence we omit the details.

The modules SUB remain unchanged, while the module FIN is simply removed, with the neuron l_h remaining in the system, with no rule inside. Thus, the computation will stop if and only if the computation in M stops.

This time, there are SUB instructions acting on register 1, but, because we no longer have the module FIN, neuron 1 contains only the rules defined in its corresponding modules SUB, hence no "illegal" operation is performed.

For obtaining the strong halting, we need to remove all the spikes from the system in a halting configuration. To this aim we add the rule $a^2 \to \lambda$ to the neurons c_2 and c_3 from module INPUT, and $a \to \lambda$ to the neuron l_h. In this way, if the deterministic register machine halts with all its registers empty, then also our system will halt with no spike inside.

The observation that the only forgetting rule $a^s \to \lambda$ with $s = 3$ was present in module FIN, which is no longer used, completes the proof.

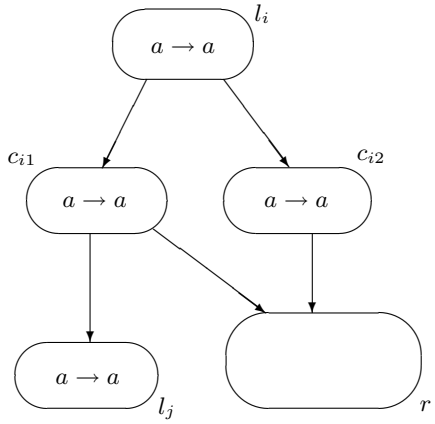


Fig. 5. Module ADD in the deterministic case

We end this section with the remark that all proofs from Section 6 of [7], where one extends the proof of Theorem 1 from the case of spike trains with only two spikes to all cases considered in Section 3 above, use only firing rules with delay 0, hence all these extensions are valid also starting from the proof of Theorem 2. Therefore, a result as that in Theorem 2, stating that the delay can be 0, holds true for all sets $N_\alpha^\beta(\Pi)$ – with various values for other parameters, depending on the constructions from [7].

5 Diminishing the Outdegree

As we have mentioned in the Introduction, the number of spikes from an SN P system can be increased only by replicating them by means of neurons with multiple outgoing synapses. Therefore, systems with the maximal outdegree one can only have inside at most the number of spikes from the initial configuration, hence a bounded number. Such SN P systems can only compute semilinear sets of numbers (see [2] and [7]), hence the outdegree cannot be decreased to one without losing the universality. As we have mentioned in the end of the proof of Theorem 2, the outdegree of that system Π can be rather large. Can we decrease it to two?

The answer is affirmative, and it can be obtained in a rather easy way, by using the idea suggested in Figure 6: by introducing intermediate neurons f_1, f_2, which take the spike they receive and split it in several spikes, and repeating this operation as many times as necessary, we can replace all neurons with more than two synapses going to other neurons with neurons from which only two synapses go out (or only one when the outdegree of d_0 is 3; of course, if the outdegree of d_0 is not an even number, then one of neurons f_1, f_2 have one synapse less than the other). If we start from a neuron with the outdegree 3, this procedure introduces

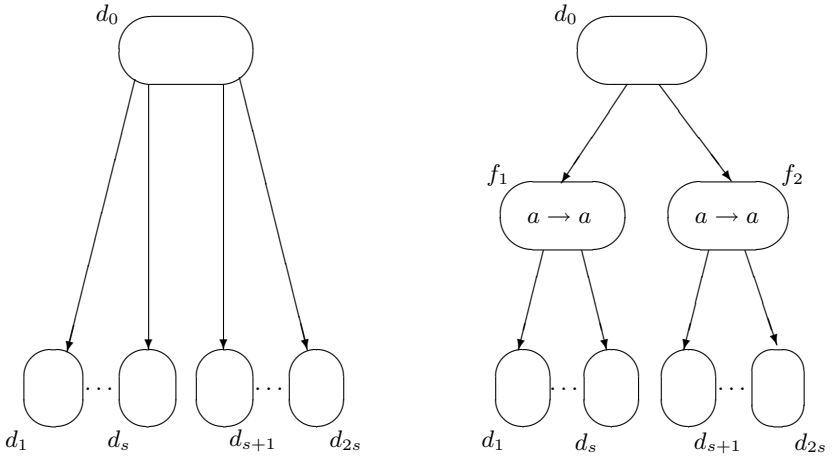


Fig. 6. Halving the outdegree

a delay in the computation proportional with $\log_2 k$, which can make problems in the case of synchronized subcomputations, or when we have a back synapse (e.g., from one of neurons $d_i, 1 \leq i \leq 2s$, to neuron d_0 in Figure 6). Fortunately, this is not the case in the modules ADD, SUB, FIN from the proof of Theorem 2 (intermediate neurons can be introduced just passsing the spike to the next neuron always when synchronizing delays should be provided), hence we can state the following strengthening of it:

Theorem 4. *$Spik_{\underline{2}}^{\beta}P_*(rule_3, cons_4, forg_4, dley_0, outd_2) = NRE$ where either $\beta = h$ or β is omitted.*

The extension of this result to the proofs from [7] is no longer immediate, because two of the proofs in [7] (Theorem 6.3 and Lemma 6.1) contain neurons with the outdegree greater than 2 and involved in processes which would be desynchronized by adding intermediate neurons like in Figure 6.

The dual problem, of bounding also the indegree of the synapse graph remains as a research topic. In the proof of Theorem 1 from [2], as well as in the proof of Theorem 2 above, there appear neurons with an arbitrarily large indegree, depending on the instructions of the register machine. This is the case for the neurons representing the registers, where there are incoming synapses in all modules which operate on those registers.

6 Removing the Forgetting Rules

We consider in this section spiking neural P systems that do not make use of forgetting rules. Surprisingly enough, universality still holds for these restricted systems.

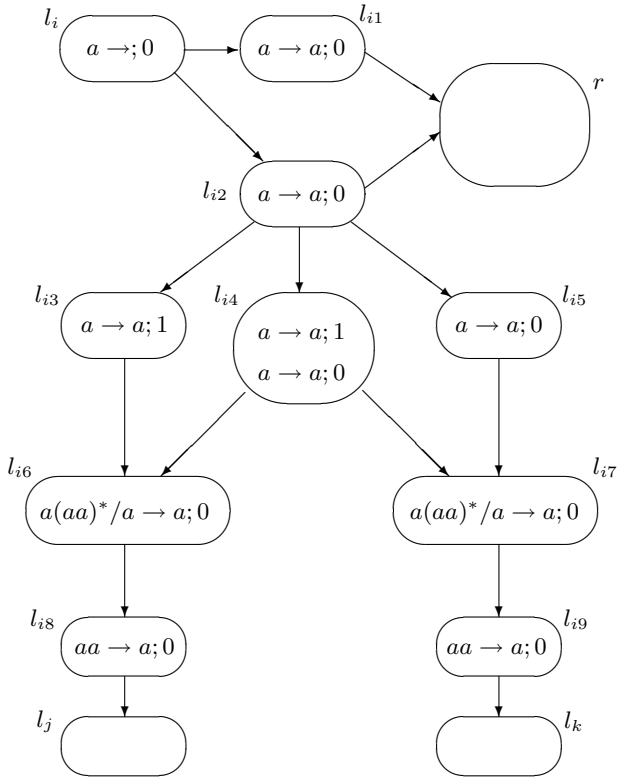


Fig. 7. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

Theorem 5. *$Spik_2^\beta P_*(rule_2, cons_3, forg_0, dley_1, outd_2) = NRE$, where $\beta = h$ or β is omitted.*

Proof. The inclusion $Spik_2^\beta P_*(rule_*, cons_*, forg_*, dley_*, outd_*) \subseteq NRE$ is straightforward and therefore omitted. To complete the proof we must show $NRE \subseteq Spik_2^\beta P_*(rule_2, cons_3, forg_0, dley_1, outd_2)$. We will do this by constructing a spiking neural P system Π with the requested parameters which simulates a register machine $M = (m, H, l_0, l_h, I)$ with the properties specified in Section 2. Like in the proof of Theorem 2, we construct modules ADD and SUB to simulate

the instructions of M, as well as an output module FIN. Each register r of M will have a neuron r in Π, and if the register contains the number n, then the neuron will contain $2n$ spikes.

The ADD module (Figure 7) used to simulate an addition instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is initiated when a spike enters the neuron with the label l_i. This causes neuron l_i to spike sending a spike to both neuron l_{i1} and l_{i2}. In the next step, both of these two neurons spike, sending two spikes to neuron r, and this represents the increment of register r by one. In this step, neuron l_{i2} also sends a spike to neurons l_{i3}, l_{i4}, and l_{i5}. The spikes in these three neurons are used to non-deterministically initiate one of the instructions l_j or l_k. After the computation of the entire module, two spikes are left in either neuron l_{i6} or l_{i7}, but these spikes do not disrupt future computations if the instruction is executed again.

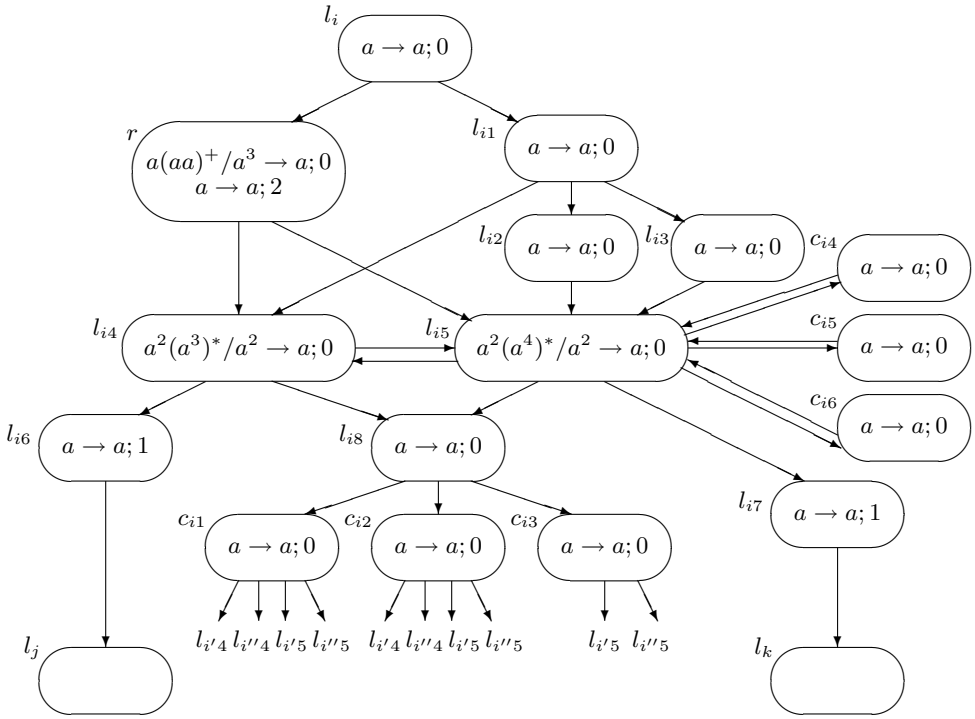


Fig. 8. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

The SUB module (Figure 8) used to simulate a subtraction instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is initiated (just like the ADD module) when a spike enters the neuron representing the instruction label l_i. The initiating spike causes the neuron l_i to immediately fire sending a spike to neurons l_{i1} and r. Neuron l_{i1} immediately sends a spike to neurons l_{i2}, l_{i3}, and l_{i4}.

At the same time, if neuron r is not empty, the rule $a(aa)^+/a^3 \to a; 0$ will be applied and a spike is sent to neurons l_{i4} and l_{i5} with no delay. (In this process, neuron l_{i5} has 1 spike added during one step and does not spike.) Now neuron l_{i4} will contain two spikes causing it to fire sending a spike to neurons l_{i5}, l_{i6}, and l_{i8}. During the same time step, both neurons l_{i2} and l_{i3} will fire and each send a spike to neuron l_{i5}. (At this point, neuron l_{i5} has gained three additional spikes meaning the total number of contained spikes is of the form $(a^4)^*$.) Now neuron l_{i8} will fire initiating the clean-up processes described later while neuron l_{i6} fires initiating the instruction l_j (after a delay of 1 time step to finish the clean-up process).

If neuron r was initially empty (corresponding to register r containing a zero count), then the rule $a \to a; 2$ is applied and a spike is sent to neuron l_{i4} and neuron l_{i5} with a delay of two time steps. At the same time, neuron l_{i1} fires sending a spike to neurons l_{i2}, l_{i3}, and l_{i4}. (In this process, neuron l_{i4} has one spike added during one step and does not spike.) Neurons l_{i2} and l_{i3} fire during the next step sending two spikes to neuron l_{i5}. Neuron l_{i5} fires sending a spike to neurons l_{i4}, l_{i7}, l_{i8}, c_{i4}, c_{i5}, and c_{i5} during the next time step and the delayed spikes from neuron r are received. (At this point, neuron l_{i4} has gained two additional spikes meaning the total number of contained spikes is of the form $(a^3)^*$. Also, the contents of the neuron l_{i5} is of the form $a(a^4)^*$ due to the received spike from neuron r.) During the next step, the neurons l_{i7}, l_{i8}, c_{i4}, c_{i5}, and c_{i6} all fire. This initiates the clean-up process and the instruction l_k. It also sends three spikes to neuron l_{i5} leaving it with $(a^4)^*$ spikes.

A clean-up process is needed because neuron r is a shared neuron between modules. For any ADD module that uses the neuron r, no interference problems occur. However, multiple SUB modules mean that when r spikes, all neurons $l_{i'4}$ and $l_{i'5}$ where $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$ will receive a single spike during the computation of instruction l_i. To guarantee that these additional spikes do not create problems when instruction $l_{i'}$ is executed, we need to make sure that each neuron $l_{i'4}$ is left with $(a^3)^*$ spikes and each neuron $l_{i'5}$ is left with $(a^4)^*$ spikes. Each of these neurons originally has the correct form and during the computation of instruction l_i they both gain a single spike (which will not cause any neuron to fire). Therefore, at the end of computation, we must add two spikes to each $l_{i'4}$ and three spikes to each $l_{i'5}$ during a single step. This is done using the neurons c_{i1}, c_{i2}, and c_{i3} which send the appropriate number of spikes to each neuron $l_{i'4}$ and $l_{i'5}$.

After the computation of the entire module is complete, all neurons except r, l_{i4}, and l_{i5} are left with zero spikes. Cell r is left with an even number of spikes. Cell l_{i4} is left with a contents of the form $(a^3)^*$ and neuron l_{i5} is left with a contents of the form $(a^4)^*$. For each instruction $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$, neuron $l_{i'4}$ is left with a contents of the form $(a^3)^*$ and neuron $l_{i'5}$ is left with a contents of the form $(a^4)^*$. This allows the module to be run repeatedly without adverse effects.

The ADD and SUB modules simulate the computation of M, but we still must output the number generated by the computation. This is handled with the FIN module (Figure 9), which is triggered when M reaches the l_h : `HALT` instruction. At this point a single spike is sent to neuron 1, which thus contains an odd number

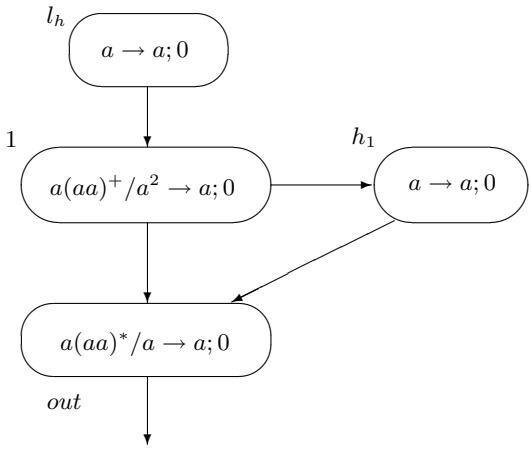


Fig. 9. The FIN module

of spikes. This causes the neuron to spike once every time unit deleting 2 spikes each time. The spikes of neuron 1 are sent to neurons h_1 and *out*. Neuron *out* will initially spike one step after neuron 1 first spikes and it will spike a second time one step after neuron 1 spikes for the last time. (These are the two times when neuron *out* contains an odd number of spikes.)

The equality $N(M) = N_2(\Pi)$ is clear. Let us now note that the maximal delay used in Π is 1 and that the outdegree can be reduced to 2 in the way indicated in the previous section. These observations conclude the proof.

The previous construction makes an essential use of the possibility of leaving spikes in the system after halting, hence it cannot be extended to the case of the strong halting.

7 Simplifying the Regular Expressions

Let us now pass to the problem of simplifying the regular expressions used in the firing rules. Already in the proofs from [2], [7], as well as in Theorems 2 and 5, one always uses rather simple expressions, in most cases checking the parity of the number of spikes from neurons. Rather surprisingly, still simpler expressions can be used – actually, the simplest over alphabet $\{a\}$: $a^i, i \geq 1$, and a^+.

In fact, the proof construction shows that even stronger restriction can be imposed. Every neuron contains at most one rule $a^+/a^r \to a; t$ or $a^r \to a; t$, and at most one rule $a^s \to \lambda$, with $s < t \leq 2$, with the only exception being neuron c_4 in Figure 11, which must be non-deterministic in order to simulate non-deterministic register machine. This result can have a biological interpretation (hence motivation): each neuron fires whenever its inner potential (measured in

number of spikes) reaches or exceeds the threshold r. If the threshold is not reached within a time unit, then the inner potential can spontaneously decay using the forgetting rule.

We give this result in the stronger form, already for the case of the outdegree bounded by two. Simultaneously we also improve the numerical parameters *cons* and *forg* in [2, 7], necessary for reaching the computational universality. We remind that the class of number sets generated by spiking neural P systems with simple regular expression is denoted by $Spik^{\beta}_{\alpha}P_*(rule^*_k, cons_p, forg_q, dley_r, outd_s)$. The notation $rule^*$ indicates the fact that regular expression are restricted to the forms $a^i, i \geq 1$, or a^+, while the other parameters are as in Section 3.

Theorem 6. *$Spik^{\beta}_{2}P_*(rule^*_2, cons_2, forg_1, dley_2, outd_2) = NRE$, where either $\beta = h$ or β is omitted.*

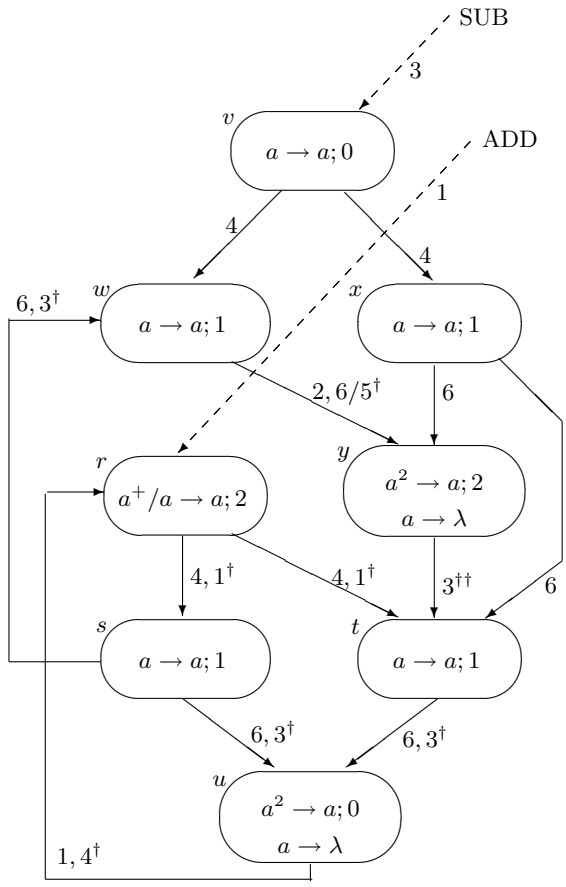
Proof. The proof generally follows the same principles applied already in Section 4. We start again from a register machine $M = (m, H, l_0, l_h, I)$, and construct a spiking neural P system Π simulating M and spiking only twice, at an interval of time which corresponds to a number computed by M. The crucial part of the construction is the implementation of registers with *dynamical circulation of spikes*. The construction is described in Figure 10. The numbers attached to edges denote moments of emitting spikes.

Step / Neuron	1	2	3	4	5	6	7 (=1)	...
r	a_{l_1}	$a^+/a \to a; 2$ — R	— — R	! —	— —	— —	— a_u	...
s	—	— —	— —	— a_r	$a \to a; 1$ — R	! —	— —	...
t	—	— —	— —	— a_r	$a \to a; 1$ — R	! —	— —	...
u	—	— —	— —	— —	— —	— $a_s a_t$	$a^2 \to a!$ —	...

Table 1. The functioning of the dynamical register storing the value 1

The number of spikes in the closed cycle $r - s, t - u$ corresponds to the number n stored in the register, if we count the pair of spikes simultaneously received and later emitted by neurons s and t as one spike. This is an important difference from the previous universality proofs, where the number n was represented by $2n$ spikes.

Assume first that there is a spike sent to neuron r at step 1 (this represents the operation of increment). The behavior of the register storing the value 1 is described in Table 1. The rules in the table denote firing of neurons, while the marks ! denote moments of emitting spikes; R in the table means that the neuron is



† if the register stores a number $n > 1$

†† if the register stores a number $n \leq 1$

Fig. 10. A register with a dynamical circulation of spikes, storing a value $n \geq 1$

in its "refractory" state in that clock cycle. One can observe that the computation is cyclic, repeated every six steps. Notice that until neuron v receives a spike from outside (which represents the operation of decrement), neurons v, x, and y cannot fire and hence cannot influence the behavior of the register.

Now, let us assume that the register stores a value $n > 1$. The corresponding computation is described in Table 2. One can notice that in this case the six steps cycle actually consists of two identical halves as steps 1, 2, and 3 are identical with 4, 5, and 6, respectively.

S. N.	1	2	3	4 (=1)	5 (=2)	6 (=3)	7 (=1)
r	! $a_u a^{n-2}$	$a^+/a \to a; 2$ a^{n-2} R	— a^{n-2} R	! $a_u a^{n-2}$	$a^+/a \to a; 2$ a^{n-2} R	— a^{n-2} R	! $a_u a^{n-2}$
s	— a_r	$a \to a; 1$ — R	! —	— a_r	$a \to a; 1$ — R	! —	— a_r
t	— a_r	$a \to a; 1$ — R	! —	— a_r	$a \to a; 1$ — R	! —	— a_r
u	$a^2 \to a!$ —	— —	— $a_s a_t$	$a^2 \to a!$ —	— —	— $a_s a_t$	$a^2 \to a!$ —

Table 2. The functioning of the dynamical register storing a value $n > 1$

S. N.	6	1	2	3	4	5	6
r	— a^{n-2} R	! $a_u a^{n-2}$	$a^+/a \to a; 2$ a^{n-2} R	— a^{n-2} R	! a^{n-2}	$a^+/a \to a; 2$ a^{n-3} R	— a^{n-3} R
s	! —	— a_r	$a \to a; 1$ — R	! —	— a_r	$a \to a; 1$ — R	! —
t	! a_x	$a \to a; 1$ — R	! —	— —	— a_r	$a \to a; 1$ — R	! —
u	— $a_s a_t$	$a^2 \to a!$ —	— a_t	$a \to \lambda$ a_s	$a \to \lambda$ —	— —	— $a_s a_t$

Table 3. Operation SUB removing one spike from the register storing a value $n > 2$

Finally, the operation of decrement can be implemented by *de-synchronizing* the spikes received by neuron u. Assume that the register stores a value $n > 2$. Let a spike be emitted to neuron v at step 3. Consequently v spikes at step 4 and x at step 6. Then the neuron t fires at step 1 and spikes at step 2 of the next cycle and the spike sent at the same time from r to t at step 1 is lost since t is closed during that clock cycle (refractory period). Therefore neuron t spikes at step 2 while s spikes at step 3. Consequently, both these spikes emitted to u are forgotten by the rule $a \to \lambda$. Table 3 summarizes the described behavior. When comparing the situation at step 6 of the first and the second cycle, one can observe that indeed one spike was removed from the register. Notice that in this case neuron y does not spike at all. The reason is that w receives a spike from s at step 3 and hence spikes at step 5. The spike emitted from v to w at step 4 is lost as w is in the refractory period that step. Both spikes emitted from w and x to y at steps 5 and 6, respectively, are forgotten.

The case $n = 2$ is analogous, the only difference is that neuron r does not fire at step 5 and keeps $n - 2$ (i.e., zero spikes) during steps 5 and 6. The next 6 steps are thus different as at step 1 we will not have r spiking, thus s and t will

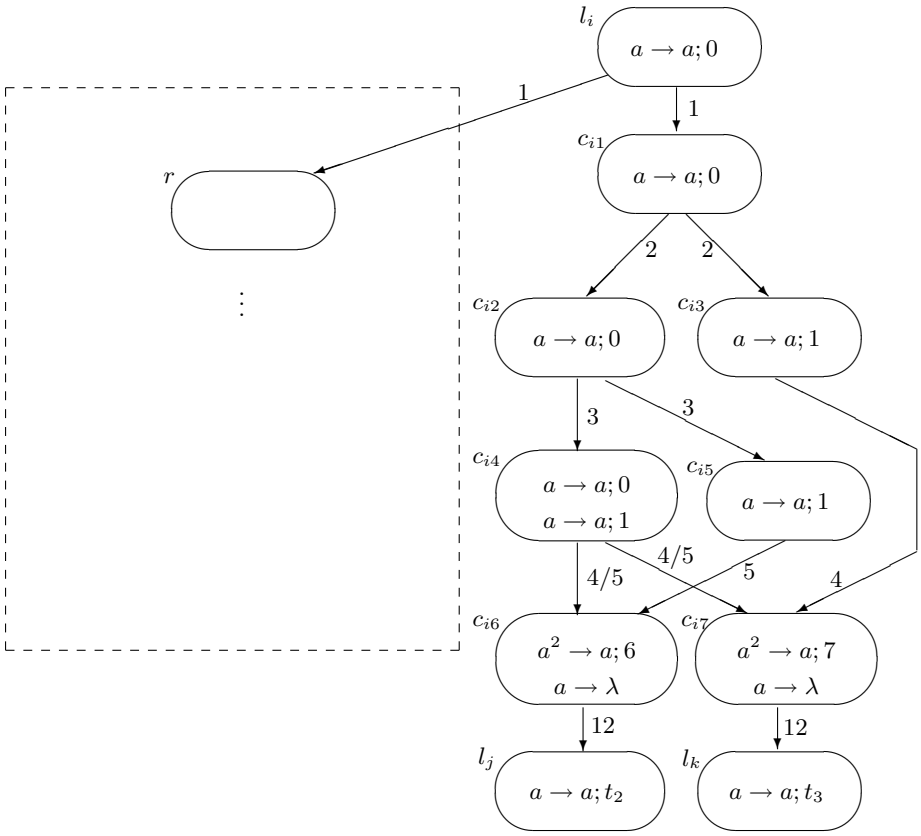


Fig. 11. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

not receive any spike and they will remain inactive up until step 4 when r will spike next. The case $n = 1$ differs mainly by the fact that neuron y spikes at step 3. Consequently, at step 4 neuron t is blocked when r spikes, and the last spike circulating in the register is lost. Details are left to the reader.

Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ – module ADD (Figure 11).

The function of the module ADD is similar to that described in the proof of Theorem 2, Figure 1. At step 1 neuron l_i emits a spike to r which, by the above description of a register, causes its increment. Simultaneously the spike is passed to neuron c_{i1} and then to c_{i2}, c_{i3}, c_{i4}, and c_{i5}. Neuron c_{i4} chooses non-deterministically whether it emits a spike at step 4 or 5. In the former case neuron c_{i7} fires at step 5 and the computation continues by instruction l_k. In the later case neuron c_{i6} fires at step 6 and the computation continues by instruction l_j.

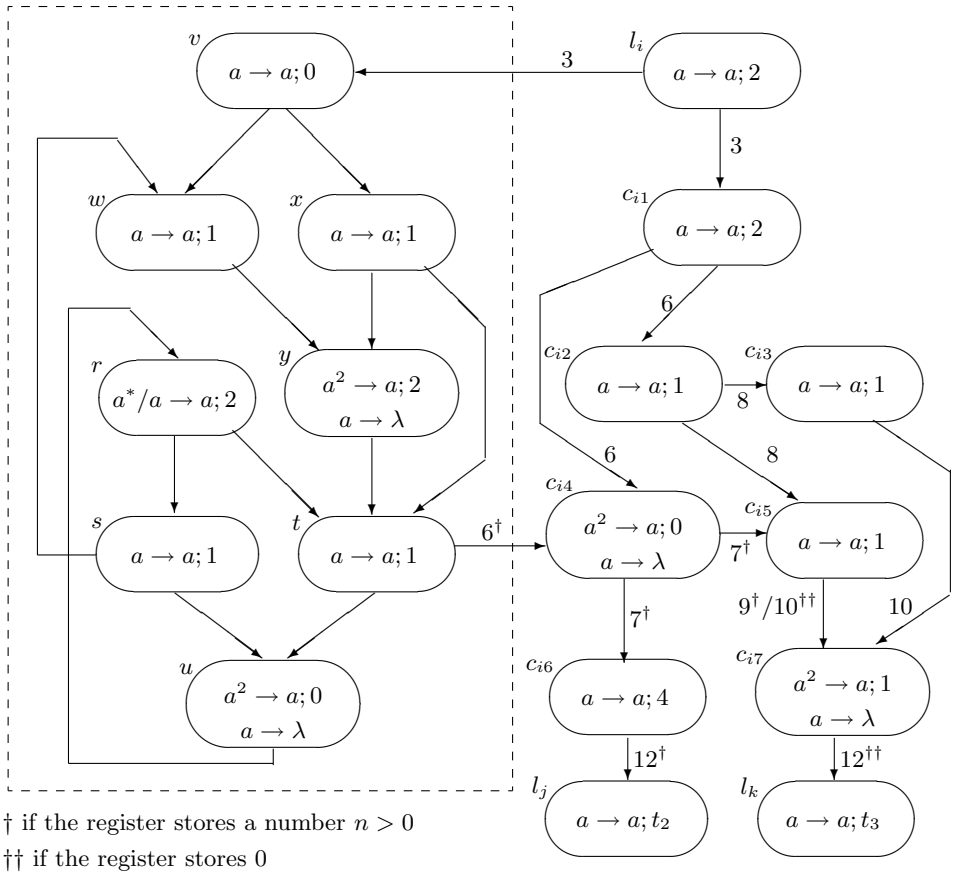


Fig. 12. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

Note that the whole simulation is artificially prolonged to 12 steps; the reason for which we need to use 12 steps will be explained below. Also, in Figure 11 we did not specify the register r because the output register 1 (which can be only incremented but never decremented) has a different structure than an "ordinary" register as described in Figure 10.

Simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ – module SUB (Figure 12).

Let us assume that at step 3 a spike is emitted from neuron l_i to v. This spike starts the de-synchronization of the register described in Table 3. Hence, if the register stores a value $n > 0$, then it is correctly decremented by one. Simultaneously, in this case neuron t emits a spike at step 6, consequently c_{i4} spikes at step 7 and the computation continues by instruction l_j. At the same

time, the spike emitted by c_{i4} prevents c_{i7} from firing at step 11 by the same de-synchronizing method as explained above (by putting c_{i5} in refractory state for step 8, thus the spike from c_{i2} to c_{i5} is lost). Then c_{i7} receives one spike at step 9 and one in step 10, both of which are forgotten.

If the register stores zero, then neuron c_{i4} does not spike at step 7. Consequently, c_{i7} receives two spikes at step 10, fires, and the computation continues by instruction l_k.

On the other branch, c_{i4} receives only one spike at step 6, which is "forgotten" at step 7, thus no spike arrives in l_j. It is easy to note that if the neuron t spikes at time 6 only (as it does for $n = 1$) or spikes at step 3 and 6 (this happens for $n > 2$) or at times 6 and 5 (for simulating SUB for $n = 2$) etc., all these times the spike(s) is/are forgotten at the next step unless we have a second spike from c_{i1} to c_{i6}.

Ending a computation – module FIN (Figure 13).

Assume now that the computation in M halts. For Π this means that the neuron l_h receives a spike. At that moment, the cycle consisting of neurons 1 and c_1 contains n spikes, for n being the contents of register 1 of M. Recall that register 1 is never decremented. Also, since we do not care about the computed value 0, we can assume that $n \geq 1$. Thus, at least one instruction $l_i : (\texttt{ADD}(1), l_j, l_k)$ had to be performed during the computation of M. By the above description, every instruction ADD or SUB is simulated in exactly 12 steps of Π. Furthermore, a register is incremented such that it receives a spike at step 1 of this 12 steps cycle. Thus, neuron 1 fires every even step if $n = 1$, and every step if $n > 1$.

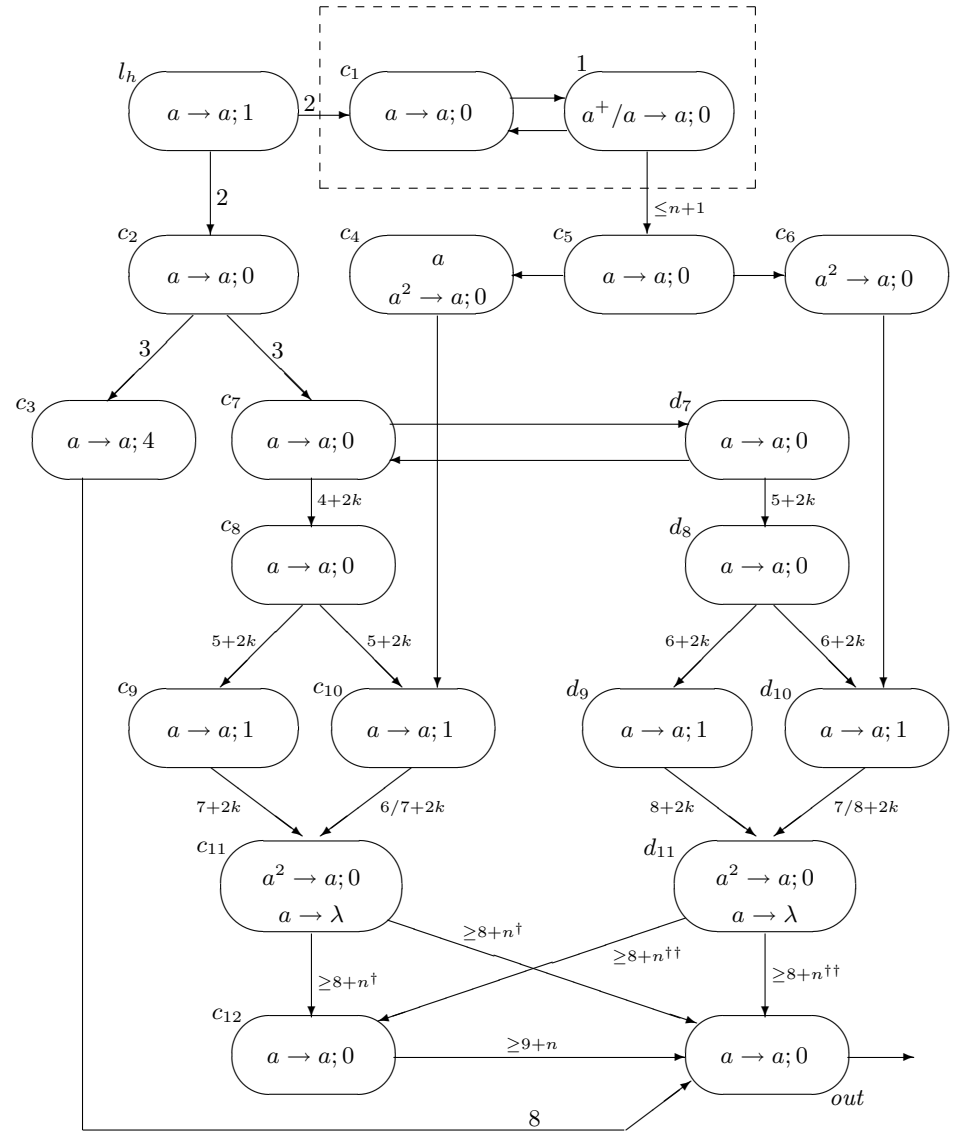
Let us pay now attention to neurons c_4, c_5, and c_6. Observe that c_4 contains already one spike at the beginning of the computation of Π. There are two possible patterns of behavior.

(i) If $n = 1$, then c_5 emits spikes every odd step. Hence, measured from the beginning of the instruction $l_i : (\texttt{ADD}(1), l_j, l_k)$, neuron c_4 spikes at steps $4k$, $k \geq 1$, and neuron c_6 spikes at steps $4k + 2$, $k \geq 1$.
(ii) If $n > 1$, then c_5 emits spikes every step. Hence, similarly, neuron c_4 spikes at steps $2k + 2$, $k \geq 1$, and neuron c_6 spikes at steps $2k + 3$, $k \geq 1$.

The above periodicity is the reason for fixing the length of ADD and SUB instructions to 12 steps. As "ordinary" registers work in 6 or 3 steps cycle, and register 1 works in 4 or 2 steps cycle, the least common multiple is 12.

Let us return now to the simulation of the instruction HALT. After its activation, the neuron l_h emits a spike to c_1 and c_2 at step 2. Neuron c_1 gets simultaneously another spike from neuron 1 and it is blocked – cannot fire any more. Hence neuron 1 decreases step by step the number of spikes it contains, and it fires the last time at step $n+1$. The reader can verify that if $n = 1$, then neuron c_4 fires the last time at step 4, and neuron c_6 does not fire any more. If $n > 1$, the situation is more complicated:

(a) if n is odd, then c_4 fires the last time at step $n + 3$ and c_6 at step $n + 2$;



† for an even $n \geq 2$ †† for an odd $n \geq 3$

Fig. 13. Module FIN (ending the computation)

(b) if n is even, then c_4 fires the last time at step $n + 2$ and c_6 at step $n + 3$.

The above observations can be generalized as follows: for an arbitrary $n \geq 1$, the last spike any of the neurons c_4, c_6 emits is sent at step $n + 3$.

Let us focus now on the rest of the module FIN. After its activation, the neuron c_2 fires at step 3. Consequently one spike starts to circulate between neurons c_7 and d_7 such that c_7 fires at steps $4 + 2k$, $k \geq 0$, and d_7 fires at steps $5 + 2k$, $k \geq 0$. Spikes from c_7 and d_7 are further emitted to c_8 and d_8, respectively.

We can now observe that the groups of neurons $c_8 - c_{11}$ and $d_8 - d_{11}$ are de-synchronizing circuits exactly as those used for implementing the instruction SUB. Whenever neurons c_4 and c_6 fire, these circuits stay de-synchronized and neither of the neurons c_{11}, d_{11} can emit a spike. Only after c_4 and c_6 emit their last spike at step $n + 3$, the pairs c_9 and c_{10} (or d_9 and d_{10}) can fire simultaneously at step $n + 6$ and, consequently, exactly one of c_{11} and d_{11} fires at step $n + 8$. From that step on, c_{11} fires at every even step and d_{11} fires at every odd step.

Finally, the neuron c_3 spikes at step 8 and hence neuron *out* spikes first time at step 9. Later, due to the above explanation, it receives another spike from c_{11} or d_{11} at step $n + 8$ and spikes a second time at step $n + 9$. Simultaneously at step $n + 9$ the neuron *out* receives two spikes (one from c_{12} and one from either c_{11} or d_{11}) and cannot spike any more. Hence the system Π correctly simulates M and outputs exactly the value n calculated by M.

Final proof remarks

The whole program I of M can be represented by a spiking neural P system Π consisting of modules ADD, SUB, and FIN presented above. These modules correspond to instructions ADD, SUB, and HALT and correctly simulate their execution, as shown above. Notice that at the beginning of computation, there are two spikes in Π, one in the neuron l_0 corresponding to the initial instruction of P, and another one in the neuron c_4 of the module FIN. (Because I contains a single instruction HALT, it follows that Π contains a single module FIN.)

To conclude the proof, a few more technical observations need to be made. First, we used in the above described modules a few neurons with the spike delay more than two (namely c_3 from FIN, c_6 from SUB, and c_6, c_7 from ADD) to simplify the construction. However, each of them can be trivially replaced by a sequence of "delaying" neurons with delays ≤ 2. Hence the parameter *dley* in the theorem statement is reduced to two.

Second, the modules ADD, SUB, and FIN use only neurons with outdegree ≤ 2. However, one should notice that if there existed $k > 1$ instructions SUB decrementing the same register r, then we would need multiple connections from neuron t described in Figure 12 to neurons c_4 corresponding to these instructions. In this case the outdegree of neuron t can be reduced to two by the construction proposed in Figure 6. However, this construction would introduce a certain delay in the simulation of the instruction SUB proportional to $\log_2 k$. As each instruction must be simulated in a multiple of 12 steps, we would have to increase the delay to the closest higher multiple of 12 and increase accordingly also the delay of neuron c_{i1} from Figure 12. The delay in c_{i1} can be performed using the suggestion from the previous paragraph, thus the overall delay will still be kept at 2.

Third, the described P system Π does not necessarily halt after generating the output – emitting two spikes from the neuron *out*. There still can be circulation of spikes in the modules corresponding to registers with nonzero values, and also between neurons c_7 and d_7 of the module FIN. If we wanted the system Π to halt, we should modify our construction as follows.

We add a connection from neuron c_{12} to c_7 in the module FIN described in Figure 13. From step $n+9$ on, the neuron c_{12} fires at every step. Hence in one of the steps $n+9$ and $n+10$ neuron c_7 receives simultaneously two spikes (one from c_{12} and one from d_7) and cannot fire any more. After emitting the last spike from neuron d_7 and passing it through the cascade of neurons $d_8 - d_{11}$ and c_{12}, the system halts. (Remember that we assume the register machine M to halt with all registers empty, excepting register 1.)

Based on the above construction, we have shown that $NRE \subseteq Spik_{\underline{2}}^{\beta} P_*(rule_2^*, cons_2, forg_1, dley_2, outd_2)$, where $\beta = h$ or β is omitted. The reverse inclusion follows by the Church-Turing thesis, and this concludes the proof.

We note that Theorem 6 remains valid when the number of spikes accumulated in the output neuron is considered as the output of the system. In this case the whole module FIN will be reduced to a single (output) neuron 1 which will accumulate spikes during the computation of Π.

It remains an open problem whether the above described normal form also holds in the case of (deterministic) accepting SN P systems.

8 Final Remarks

We have shown in this paper that the Turing completeness of spiking neural P systems is preserved even if we only work with systems with delay 0 in the firing rules, and/or with the maximal outdegree of the synapse graph being two, or with the regular expressions from the firing rules of the simplest form, $a^i, i \geq 1$, or a^+. Also, universality was obtained when no forgetting rules have been used. These results have been achieved at the price of slightly increasing other parameters, in particular, the number of spikes consumed in the firing rules. In general, finding the optimal number of spikes which are consumed remains as an open problem.

We have also shown that the limitation of the maximal outdegree to two can be combined with delay 0 in the firing rules, or with the simplest form of regular expressions without losing the computational universality. It remains an open problem whether the simultaneous limitation to delay 0 and to the simplest form of regular expressions would still be computationally universal.

Similarly, as already pointed out in the end of Section 5, it is an open problem whether we can also bound the indegree of the synapse graph without losing the universality.

Another interesting research problem is to extend the results from this paper to SN P systems with a bounded number of spikes in their neurons, hence to prove again the characterization of $NREG$ from [2], [7] for SN P systems without

delay, with outdegree 2, or with regular expressions as in Theorem 6. Also the extension to processing infinite sequences of bits, as investigated in [8], should be investigated.

References

1. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
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.*, to appear (also available at [10]).
8. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
9. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
10. The P Systems Web Page: `http://psystems.disco.unimib.it`.