Extended SNP Systems with States

Artiom Alhazov[1], Rudolf Freund[2], and Sergiu Ivanov[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[3] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. We consider (extended) spiking neural P systems with states, where the applicability of rules in a neuron not only depends on the presence of sufficiently many spikes (yet in contrast to the standard definition, no regular checking sets are used), but also on the current state of the neuron. Moreover, a spiking rule not only sends spikes, but also state information to the connected neurons. We prove that this variant of the original model of extended spiking neural P systems can simulate register machines with only two states, even in the basic non-extended variant.

1 Introduction

In the area of P systems, the model of *spiking neural P systems* was introduced in [7]. Whereas the basic model of membrane systems, see [11], reflects hierarchical membrane structures, in spiking neural P systems the cells are arranged in a tissue-like manner, with the contents of a cell (neuron) consisting of a number of so-called *spikes*, i.e., of a multiset over a single object. The rules assigned to a neuron allow us to send information to other neurons in the form of electrical impulses (also called spikes) which are summed up at the target neuron; the application of the rules depends on the contents of the neuron and in the general case is described by regular sets. As inspired from biology, the neuron sending out spikes may be "closed" for a specific time period corresponding to the refraction period of a neuron; during this refraction period, the neuron is closed for new input and cannot get excited ("fire") for spiking again.

The length of the axon may also cause a time delay before a spike arrives at the target. Moreover, the spikes coming along different axons may cause effects of different magnitude. All these biologically motivated features were included in the model of extended spiking neural P systems considered in [3], the most

important theoretical feature being that neurons can send spikes along the axons with different magnitudes at different moments of time.

In this paper, we consider a variant of the model of extended spiking neural P systems which not only uses spikes to be sent to other neurons when some neuron spikes, but also allows for sending some additional information called "state" along the axons. All these state informations arriving in a neuron then determine the next state of the neuron. On the other hand, we do not use the regular checking sets for the current number of spikes in the neuron any more, which decreases the amount of information a spiking rule may use. Hence, the spiking rules now depend on the current states of the neurons and the availability of sufficiently many spikes. This variant of extended spiking neural P systems with states has been inspired by the variant of spiking neural P systems with polarizations, see [16], where the states are called polarizations, and the underlying model of extended spiking neural P systems was the basic one with a fixed connection structure, only extended by allowing more than one spike to be sent along the axons. There it was shown that computational completeness (i.e., simulation of register machines) can be obtained with only three polarizations. In this paper we now show that computational completeness can already be obtained with only two states, i.e., with two polarizations, even for the basic non-extended variant as considered in [16], which solves an open problem raised at the Brainstorming Week on Membrane Computing in Sevilla at the beginning of February 2016.

The rest of the paper is organized as follows: In the next section, we recall some preliminary notions and definitions from formal language theory, especially the definition and some well-known results for register machines. In Section 3 we recall the definitions of the extended model of spiking neural P systems as considered in [3] and then define the model of *spiking neural P systems with states* as considered in this paper. In Section 4, we prove our main result and show that spiking neural P systems with only two states (0 and 1) can simulate register machines; the complexity of the construction depends on the features we require the spiking neural P systems to have, but the result even holds true for the basic non-extended variant of spiking neural P systems with states, where the connection structure between the neurons is fixed and does not depend on the spiking rules applied in the neurons. Moreover, we can use a very simple global state composition function computing the new state of a neuron from the state information having arrived from the input neurons in the simplest way by going to the "activated state 1" if and only if at least one such activating signal 1 has come in the previous step. In Section 5, we show how small universal spiking neural P systems with states can be constructed based on the results obtained in this paper. A short summary of the results we obtained concludes the paper.

2 Preliminaries

In this section we recall the basic elements of formal language theory and especially the definitions and results for register machines; we also refer to the corresponding

section from [3] and [2]. For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [14]. We just list a few notions and notations:

V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; for any $w \in V^*$, $|w|$ denotes the number of symbols in w (the *length* of w). $\mathbb{N}_+$ denotes the set of positive integers (natural numbers), $\mathbb{N}$ is the set of non-negative integers, i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$.

2.1 Register Machines

The proofs of the results establishing computational completeness in the area of P systems are often based on the simulation of register machines; we refer to [9] for original definitions, and to [6] for the definitions we use in this paper:

An *n-register machine* is a tuple $M = (n, B, l_0, l_h, P)$, where n is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and P is the set of instructions bijectively labeled by elements of B. The instructions of M can be of the following forms:

- $p : (\texttt{ADD}(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq j \leq n$.
  Increases the value of register r by one, followed by a non-deterministic jump to instruction q or s. This instruction is usually called *increment*.
- $p : (\texttt{SUB}(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq j \leq n$.
  If the value of register r is zero then jump to instruction s; otherwise, the value of register r is decreased by one, followed by a jump to instruction q. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(s_1, \cdots, s_\beta)$ of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first β registers containing the numbers $s_1, \cdots, s_\beta$ (and all other registers being empty).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate or accept exactly the sets of vectors of non-negative integers which can be generated by Turing machines.

3 Extended Spiking Neural P Systems

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [10] and [12]; comprehensive information can be found on the P systems

web page [15]. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [7] as well as to the corresponding Chapter 13 in the Handbook of Membrane Computing [12]. For the definition of an *extended spiking neural P system* we refer to [3].

We now extend the model of *extended spiking neural P systems* to the new model of *extended spiking neural P systems with states*, i.e., the neurons can be in different states, and depending on the current state of a neuron, different spiking rules may be applicable.

Definition 1. *An* extended spiking neural P system with states *(of degree $m \geq 1$) (an* ESNPS system *for short) is a construct $\Pi = (N, S, I, R, f)$ where*

- *N is the set of* cells *(or* neurons*); the neurons may be uniquely identified by a number between* 1 *and m or by an alphabet of m symbols;*
- *S is the set of* states*;*
- *I describes the* initial configuration *by assigning an initial value (of spikes) and an* initial state *to each neuron; for the sake of simplicity, we assume that at the beginning of a computation we have no pending packages along the axons between the neurons;*
- *R is a finite set of* rules *of the form $\left(i, s_i : E/a^k \to P; d\right)$ such that $i \in \{1, \cdots, m\}$ (specifying that this rule is assigned to neuron i), $s_i \in S$ is the current state of neuron i, $E \subseteq REG(\{a\})$ is the* checking set *(the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, d is the* delay *(the "refraction time" when neuron i performs this rule), and P is a (possibly empty) set of* productions *of the form (l, w, s, t) where $l \in [1..m]$ (thus specifying the target neuron), $w \in \{a\}^*$ is the* weight *of the energy sent along the axon from neuron i to neuron l, $s \in S$ is the* state *sent along the axon from neuron i to neuron l, and t is the time needed before the information sent from neuron i arrives at neuron l (i.e., the* delay along the axon*);*
- *f is the state composition function, which for each neuron allows for computing the new state of a neuron from its current state and the state signals having arrived in the neuron in the previous step.*

Definition 2. *A* configuration *of the ESNPS system is described as follows:*

- *for each neuron, the actual number of spikes in the neuron is specified;*
- *in each neuron i, we may find an "activated rule" $\left(i, s_i, E/a^k \to P; d'\right)$ waiting to be executed where d' is the remaining time until the neuron spikes;*
- *in each axon to a neuron l, we may find pending packages of the form (l, w, s, t') where t' is the remaining time until $|w|$ spikes have to be added to neuron l provided it is not closed for input at the time this package arrives.*

A transition *from one configuration to another one now works as follows:*

- *for each neuron i, we first check whether we find an "activated rule" $\left(i, s_i, E/a^k \to P; d'\right)$ waiting to be executed; if $d' = 0$, then neuron i "spikes",*

i.e., for every production (l, w, s, t) *occurring in the set* P *we put the corresponding package* (l, w, s, t) *on the axon from neuron* i *to neuron* l*, and after that, we eliminate this "activated rule"* $\big(i, s_i, E/a^k \to P; 0\big)$*;*

- *for each neuron* l*, we now consider all packages* (l, w, t') *on axons leading to neuron* l*; provided the neuron is not closed, i.e., if it does not carry an activated rule* $\big(i, s_i, E/a^k \to P; d'\big)$ *with* $d' > 0$*, we then sum up all weights* w *in such packages where* $t' = 0$ *and add this sum of spikes to the corresponding number of spikes in neuron* l *as well as remember the corresponding state signals* s *for eventually computing the next state of the neuron; in any case, the packages with* $t' = 0$ *are eliminated from the axons, whereas for all packages with* $t' > 0$*, we decrement* t' *by one;*
- *for each neuron* i*, we now again check whether we find an "activated rule"* $\big(i, s_i, E/a^k \to P; d'\big)$ *(with* $d' > 0$*) or not; if we have not found an "activated rule", we now compute the new state of the neuron by using the state composition function for the underlying neuron* i*. Then we may apply any rule* $\big(i, s_i, E/a^k \to P; d\big)$ *from* R *for which neuron* i *is in state* s_i *and the current number of spikes in the neuron is in* E*, and then put a copy of this rule as "activated rule" for this neuron into the description of the current configuration; on the other hand, if there still has been an "activated rule"* $\big(i, s_i, E/a^k \to P; d'\big)$ *in the neuron with* $d' > 0$*, then we replace* d' *by* $d' - 1$ *and keep* $\big(i, s_i, E/a^k \to P; d' - 1\big)$ *as the "activated rule" in neuron* i *in the description of the configuration for the next step of the computation.*

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A computation *is a sequence of configurations starting with the initial configuration given by* I*. A computation is called* successful *if it halts, i.e., if no pending package can be found along any axon, no neuron contains an activated rule, for no neuron, a rule can be activated, and no neuron would change its state in the next step (thus making other spiking rules applicable).*

In the original model introduced in [7], we have only one state, so we can omit the states in the description of spiking rules in this paragraph. In the productions (l, w, t) of a rule $\big(i, E/a^k \to \{(j, w_j, t_j) \mid j \in J\}; d\big)$, only $w = a$ (for *spiking rules*) or $w = \lambda$ (for *forgetting rules*) as well as $t = 0$ was allowed (and for forgetting rules, the checking set E had to be finite and disjoint from all other sets E in rules assigned to neuron i). Moreover, reflexive axons, i.e., leading from neuron i to neuron i, were not allowed, hence, for (l, w, t) being a production in a rule $\big(i, E/a^k \to P; d\big)$ for neuron i, $l \neq i$ was required. Yet the most important extension was that different rules for neuron i may affect different axons leaving from it whereas in the original model the structure of the axons (called synapses there) was fixed. In [3], the sequence of substeps leading from one configuration to the next one together with the interpretation of the rules from R was taken in such a way that the original model can be interpreted in a consistent way within the extended model introduced in that paper. As mentioned in [3], from a mathematical point

of view, another interpretation would have been even more suitable: whenever a rule $(i, E/a^k \to P; d)$ is activated, the packages induced by the productions (l, w, t) in the set P of a rule $(i, E/a^k \to P; d)$ activated in a computation step are immediately put on the axon from neuron i to neuron l, whereas the delay d only indicates the refraction time for neuron i itself, i.e., the time period this neuron will be closed. The delay t in productions (l, w, t) can be used to replace the delay in the neurons themselves in many of the constructions elaborated, for example, in [7], [13], and [4]. Yet as in (the proofs of computational completeness given in) [3], we shall not need any of the delay features in this paper, hence we need not go into the details of these variants of interpreting the delays in more details. Finally, we mention that as in [5], the notion of *extended* spiking neural P systems often is used only taking into account that more than one spike can be sent along all axons with one spiking rule.

Depending on the purpose the ESNP(S) system is to be used, some more features have to be specified: for generating k-dimensional vectors of non-negative integers, we have to designate k neurons as *output neurons*; the other neurons then will also be called *actor neurons*. As in [3], also in this paper, we take the number of spikes at the end of a successful computation in the neuron as the output value.

Definition 3. *Obviously, extending an already very powerful model with an additional feature (states) yields a model which is at least as powerful, even with respect to descriptional complexity. Hence, besides completely omitting delays, as in [16] we also omit the checking sets (in fact, this means taking all checking sets to be* $\{a\}^+$*). Thus, for a spiking rule* $(i, s_i : E/a^k \to P; d)$ *we write* $i : (s_i, a^k) \to P$*. Moreover, the set* P *will not be written as a set, but just by concatenating its elements of the form* (l, w, s)*, where* l *is the target neuron,* w *describes the number of spikes sent to* l *and* s *is the state signal sent to* l*.*

The following example illustrates the computational power of ESNPS systems with two states by showing how exponential number languages can be generated.

Example 1. We will construct the ESNPS system

$$\Pi_{4^n} = (\{\sigma_1, \sigma_1', \sigma_2, \sigma_2'\}, \{0, 1\}, I, R, f)$$

generating the multiset language $\left\{a^{4^n} \mid n > 1\right\}$ in the output neuron σ_1. Initially, σ_1 and σ_1' are in state 1 and contain one and two spikes, respectively. On the other hand, neurons σ_2 and σ_2' initially are in state 0 and are empty.

The state composition function f, for every neuron, is given as follows: If any state signal 1 has arrived in the previous step, the state of the neuron is 1, and 0 otherwise.

To illustrate the rules in the ESNPS system, we use the following graphical notation: Each rule is represented by an arrow with a single tail but with multiple

heads; the branching point is highlighted by a black bullet. The left-hand side of the rule is written on the segment preceding the bullet and each right-hand side on the corresponding arrow head. When writing the right-hand sides, we omit the names of the target neurons (because they are pointed at by the arrow heads).

Π_{4^n} works in a two-phase cycle. In the first phase, all the spikes from σ_1 are transferred in two copies into σ_2; this phase is controlled by σ'_1. The second phase is symmetric: the spikes from σ_2 are doubled and moved into σ_1, under the control of σ'_2.

The first phase is governed by the following rules in neurons σ_1 and σ'_1:

$$\begin{aligned}
&\sigma_1 : (1, a) \to \left(\sigma_2, a^2, 0\right) (\sigma'_1, a, 1)\,,\\
&\sigma'_1 : (1, a) \to (\sigma_1, \lambda, 1)\,,\\
&\sigma'_1 : (0, a) \to (\sigma_2, \lambda, 1) \left(\sigma'_2, a^2, 1\right).
\end{aligned}$$

The graphical representation of these rules is given in Figure 1.

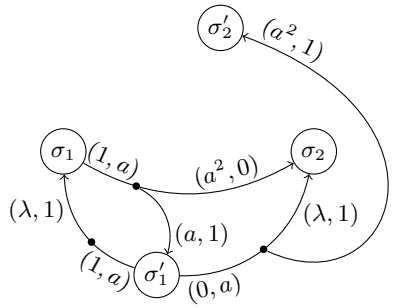

Fig. 1. Multiplication by 2 in ESNPS with two states.

The loop between σ_1 and σ'_1 ensures that, while there are still spikes in σ_1, both neurons stay in state 1. When there are no more spikes in σ_1, σ'_1 must pass into state 0 and will have to use its last spike (of the two it normally contains) to apply the rule $(0, a) \to (\sigma_2, \lambda, 1) \left(\sigma'_2, a^2, 1\right)$. This rule puts two spikes into the control neuron σ'_2 and switches both neurons σ_2 and σ'_2 to state 1, thereby starting the second phase of the cycle. The second phase is totally symmetric and is governed by the following rules in neurons σ_2 and σ'_2:

$$\begin{aligned}
&\sigma_2 : (1, a) \to \left(\sigma_1, a^2, 0\right) (\sigma'_2, a, 1)\,,\\
&\sigma'_2 : (1, a) \to (\sigma_2, \lambda, 1)\,,\\
&\sigma'_2 : (0, a) \to (\sigma_1, \lambda, 1) \left(\sigma'_1, a^2, 1\right).
\end{aligned}$$

Finally, to ensure that the system halts after the second phase of the cycle, we add the following rule to the control neuron σ'_2:

$$\sigma'_2 : (0, a) \to (\sigma_1, \lambda, 0)\,.$$

Thus, σ_2' may choose between restarting the loop by switching the state of σ_1 and σ_1', or just forgetting the last control spike, thus effectively bringing the system to a halt, with 4^n copies of a in σ_1, where n is the number of times the cycle has run.

4 Simulating Register Machines with Extended Spiking Neural P Systems with only Two States

We now consider an arbitrary *n-register machine* $M = (n, B, l_0, l_h, P)$ and provide a first simple proof how to simulate the computations of such a register machine by an extended spiking neural P system with only two states:

For all neurons, we use one global state composition function, as in the example, i.e., if any state signal 1 has arrived in the previous step, the state of the neuron is 1, and 0 otherwise.

$p : (\texttt{ADD}(r), q, s)$ is simulated by neuron p with the rules

p: $(0, a) \to (q, a, 0)(r, a, 0)$ and
p: $(0, a) \to (s, a, 0)(r, a, 0)$.

meaning that neuron p (always staying in state 0) consumes one spike and sends one spike and state 0 to neuron q or neuron s and to neuron r (the neuron representing register r). Figure 2 illustrates these rules graphically.

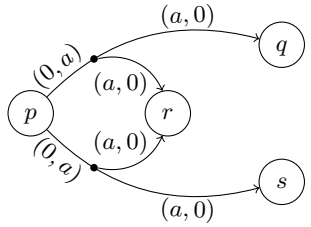

Fig. 2. Simulation of *ADD* relying on a flexible connection structure and on sending zero spikes.

The rule in register r allowing for simulating SUB-instructions is:
r: $(1, a) \to \prod_{l \in SUB(r)} (l'', \lambda, 1)$

In case the register is non-empty, in the activated state 1 one spike is eliminated and state 1 is sent to every neuron l'' for every label l of a SUB-instruction, yet no spike a is sent.

$p : (\texttt{SUB}(r), q, s)$ is simulated by neurons p, p', p'' with the rules

p: $(0, a) \to (p', a, 1)\,(r, \lambda, 1)$,
p': $(1, a) \to (p'', a, 0)$, as well as
p'': $(0, a) \to (s, a, 0)$ and

p'': $(1, a) \to (q, a, 0)$.

The rules are illustrated in Figure 3. The dashed arrow represents the family of right-hand sides broadcasting spikes from neuron r to neurons l'', $l \in SUB(r)$.

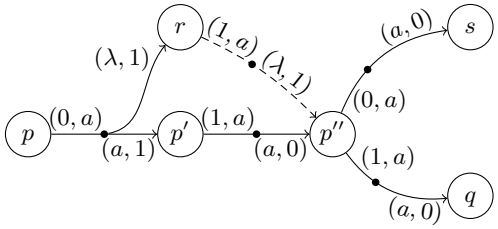

Fig. 3. Simulation of SUB relying on a flexible connection structure and on sending zero spikes.

The simple construction described above obeys to the following features:

- We only need two states!
- We do not use self-loops.
- We send the same state to all neurons in each rule!
- Exactly one spike is consumed by each rule!
- Wo do not use forgetting rules!
- But on the other hand, we allow to also send *zero* spikes to a neuron!
- The connection structure only depends on the state in case of deterministic register machines! Yet by using a more complicated construction for the simulation of non-deterministic ADD-instructions we can even obtain that feature in general:
  $p : (\texttt{ADD}(r), q, s)$ is simulated by neurons p, p', p'' with the rules
  p: $(0, a) \to (p', a, 0)\,(r, a, 0)$,
  p': $(0, a) \to (p'', a, 0)$ and
  p': $(0, a) \to (p'', a, 1)$, as well as
  p'': $(0, a) \to (q, a, 0)$ and
  p'': $(1, a) \to (s, a, 0)$.
  See Figure 4 for a graphical illustration of these rules.

For comparison with the model considered in [16] (where states are called *polarizations*) we have to ask the following question: *Can we have a completely static connection structure even not depending on the state of the neuron?*

We first show that a non-deterministic ADD-instruction can be simulated within a fixed connection structure, now using forgetting rules, yet also using the initial neuron p in the activated state 1:

$p : (\texttt{ADD}(r), q, s)$ is simulated by neurons p, p' with the rules

p: $(1, a) \to (p', a, 0)\,(r, a, 0)$,

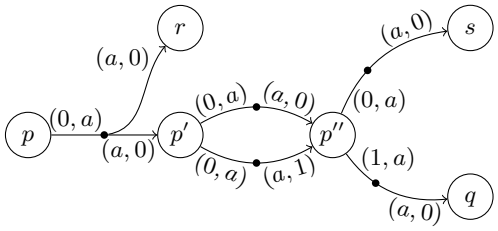

Fig. 4. Simulation of *ADD* using a connection structure which only depends on states.

p': $(0,a) \to \left(0''_q, a, 0\right)(1''_s, a, 0)$ and
p': $(0,a) \to \left(0''_q, a, 1\right)(1''_s, a, 1)$,
together with the following rules in the neurons $0''_l, 1''_l$, for every label $l \in B$:

$0''_l$: $(0,a) \to (l, a, 1)$ and
$0''_l$: $(1,a) \to \lambda$,
$1''_l$: $(1,a) \to (l, a, 1)$ and
$1''_l$: $(0,a) \to \lambda$.

Figure 5 gives a graphical illustration of these rules. Forgetting rules which only consume spikes without sending anything anywhere are depicted as dangling arrows.

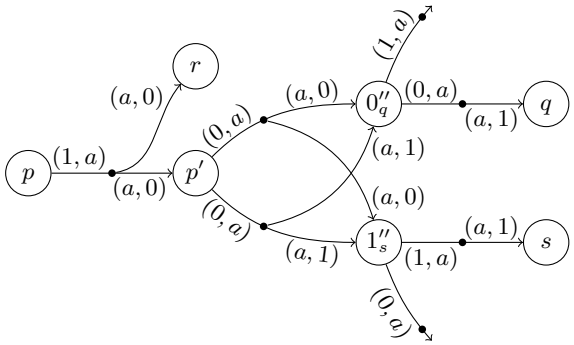

Fig. 5. Simulation of *ADD* using a fixed connection structure.

Instead of showing how SUB-instructions can also be simulated within a fixed connection structure, we also attack the last remaining non-standard feature at the same time, i.e.: *Can we avoid sending zero spikes?*

$p : (\texttt{SUB}(r), q, s)$ is simulated by the neurons $p, \tilde{p}, \tilde{p}', \hat{p}, \hat{p}', \hat{p}'', \bar{p}, \bar{p}'$ with the rules

p: $(1,a) \to (\tilde{p}, a, 1)(\hat{p}, a, 1)(r, a, 1)$,

$\tilde{p}$: $(1,a) \to (\tilde{p}',a,0)$,
$\tilde{p}'$: $\left(1,a^2\right) \to (q,a,1) \prod_{l \in SUB(r) \setminus \{p\}} \left(\hat{l}'',a,1\right)$,
$\tilde{p}'$: $(0,a) \to \lambda$ as well as

$\hat{p}$: $(1,a) \to (\hat{p}',a,1)$,
$\hat{p}'$: $(1,a) \to (\hat{p}'',a,0)$,
$\hat{p}''$: $(0,a) \to (r,a,1)\,(\bar{p},a,1)$,
$\hat{p}''$: $\left(1,a^2\right) \to \lambda$,
$\bar{p}$: $(1,a) \to (\bar{p}',a,1)$, and
$\bar{p}'$: $(1,a) \to (s,a,1) \prod_{l \in SUB(r)} \left(\hat{l}'',a,1\right)$

together with the following rules for the register neuron r and for the additional neuron r':

r: $\left(1,a^2\right) \to (r',a,1) \prod_{l \in SUB(r)} \left(\tilde{l}',a,1\right)$ and

r': $(1,a) \to \prod_{l \in SUB(r)} \left(\hat{l}'',a,1\right)$.

The rules are graphically illustrated in Figure 6. As before, dangling arrows represent rules sending nothing, while dashed arrows correspond to families of right-hand sides. For readability, we highlight neurons p, q, and s, which represent the states of the simulated machine. The dotted line going into neuron $\hat{p}''$ represents the family of right-hand sides which broadcast spikes and states to all neurons $\hat{l}''$, *except* neuron $\hat{p}''$ ($l \in SUB(r) \setminus \{p\}$). Finally, to avoid line intersections, the picture uses a "clone" of neuron r (the small grey r between $\hat{p}''$ and $\bar{p}$), which represents the same neuron r.

The main idea of this construction is to start both decrement and zero-check case in parallel and then, depending on the signal from r and r' take the necessary action, including to *reset* register r to the 0 if the additional spike sent there did not lead to a spiking action of neuron r in case the value stored in the register was zero. Moreover, all actor neurons affected by signals from neuron r not belonging to the current label p have to be reset without allowing them to act in a non-desired way:

For the neurons $\tilde{p}'$ this happens *automatically* as with only one spike a they cannot spike in state 1, yet after one step the state goes back to 0 and then allows the spike to be forgotten.

For the neurons $\hat{p}''$ this happens if the state signal 1 and the spike from neuron r' are accompanied by a second spike which allows for resetting the neuron by using the forgetting rule $\left(1,a^2\right) \to \lambda$.

5 Universal (Extended) Spiking Neural P Systems with Two States

We simulate the strongly universal register machine U_{22} of Korec, see [8]. Rather than performing a direct simulation which would yield $9 \times 1 + 8 \times 1 + 13 \times 4 = 69$

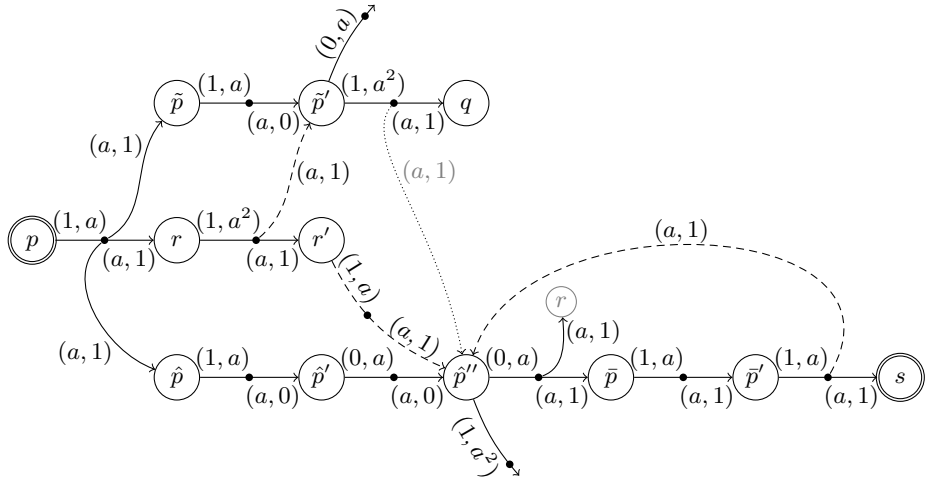

Fig. 6. Simulating SUB using a fixed connection structure and without sending zero spikes.

rules, we notice that simulation of ADD-instructions does not require separate rules, because it can be done as a part of the simulation of SUB-instructions. More exactly, increments are built into the transitions to q and s of $p : (\mathtt{SUB}(r), q, s)$. This has been formalized as *generalized register machine* (GRM for short), see [1]. The rules of U_{22} in the GRM form are given below.

$$
\begin{aligned}
q_1 &: (\mathtt{SUB}(1), \mathtt{ADD}(7)q_1, \mathtt{ADD}(6)q_4)\,,\\
q_4 &: (\mathtt{SUB}(5), \mathtt{ADD}(6)q_4, q_7)\,,\\
q_7 &: (\mathtt{SUB}(6), \mathtt{ADD}(5)q_{10}, q_4)\,,\\
q_{10} &: (\mathtt{SUB}(7), \mathtt{ADD}(1)q_7, q_{13})\,,\\
q_{13} &: (\mathtt{SUB}(6), \mathtt{ADD}(6)q_{14}, q_1)\,,\\
q_{14} &: (\mathtt{SUB}(4), q_1, q_{16})\,,\\
q_{16} &: (\mathtt{SUB}(5), q_{18}, q_{23})\,,\\
q_{18} &: (\mathtt{SUB}(5), q_{20}, q_{27})\,,\\
q_{20} &: (\mathtt{SUB}(5), \mathtt{ADD}(4)q_{16}, \mathtt{ADD}(2)\mathtt{ADD}(3)q_{32})\,,\\
q_{23} &: (\mathtt{SUB}(2), q_{32}, q_{25})\,,\\
q_{25} &: (\mathtt{SUB}(0), q_1, q_{32})\,,\\
q_{27} &: (\mathtt{SUB}(3), q_{32}, \mathtt{ADD}(0)q_1)\,,\\
q_{32} &: (\mathtt{SUB}(4), q_1, q_h)\,.
\end{aligned}
$$

We note that also the first step of the simulation of a generalized SUB-instruction can be embedded into the last step of the preceding simulation. Moreover, note that in this case we may start with one spike in neuron q''_{13}. It is easy to see that it suffices to have 3 rules per each of the 13 generalized conditional decrement instructions and 1 rule per each of the 8 registers, yielding a total

of only **47** rules, associated to register neurons, primed instruction neurons and double-primed instruction neurons.

Register neurons

$0 : (1, a) \to (q''_{25}, \lambda, 1)$,
$1 : (1, a) \to (q''_{1}, \lambda, 1)$,
$2 : (1, a) \to (q''_{23}, \lambda, 1)$,
$3 : (1, a) \to (q''_{27}, \lambda, 1)$,
$4 : (1, a) \to (q''_{14}, \lambda, 1)\,(q''_{32}, \lambda, 1)$,
$5 : (1, a) \to (q''_{4}, \lambda, 1)\,(q''_{16}, \lambda, 1)\,(q''_{18}, \lambda, 1)\,(q''_{20}, \lambda, 1)$,
$6 : (1, a) \to (q''_{7}, \lambda, 1)\,(q''_{13}, \lambda, 1)$,
$7 : (1, a) \to (q''_{10}, \lambda, 1)$.

The **primed instruction neurons** have the rules

$$q'_i : (1, a) \to (q''_i, a, 0) \text{ for } i \in \{1, 4, 7, 10, 13, 14, 16, 18, 20, 23, 25, 27, 32\}.$$

We give the rules of **double-primed instruction neurons** in the table below, the row representing the neuron, the column representing the left side of a rule, and their intersection containing the right side of that rule.

	$(0, a)$	$(1, a)$
q''_1	$(q'_4, a, 1)\,(5, \lambda, 1)(6, a, 0)$	$(q'_1, a, 1)\,(1, \lambda, 1)(7, a, 0)$,
q''_4	$(q'_7, a, 1)\,(6, \lambda, 1)$	$(q'_4, a, 1)\,(5, \lambda, 1)(6, a, 0)$,
q''_7	$(q'_4, a, 1)\,(5, \lambda, 1)$	$(q'_{10}, a, 1)\,(7, \lambda, 1)(5, a, 0)$,
q''_{10}	$(q'_{13}, a, 1)\,(6, \lambda, 1)$	$(q'_7, a, 1)\,(6, \lambda, 1)(1, a, 0)$,
q''_{13}	$(q'_1, a, 1)\,(1, \lambda, 1)$	$(q'_{14}, a, 1)\,(4, \lambda, 1)(6, a, 0)$,
q''_{14}	$(q'_{16}, a, 1)\,(5, \lambda, 1)$	$(q'_1, a, 1)\,(1, \lambda, 1)$,
q''_{16}	$(q'_{23}, a, 1)\,(2, \lambda, 1)$	$(q'_{18}, a, 1)\,(5, \lambda, 1)$,
q''_{18}	$(q'_{27}, a, 1)\,(3, \lambda, 1)$	$(q'_{20}, a, 1)\,(5, \lambda, 1)$,
q''_{20}	$(q'_{32}, a, 1)\,(4, \lambda, 1)(2, a, 0)(3, a, 0)$	$(q'_{16}, a, 1)\,(5, \lambda, 1)(4, a, 0)$,
q''_{23}	$(q'_{25}, a, 1)\,(0, \lambda, 1)$	$(q'_{32}, a, 1)\,(4, \lambda, 1)$,
q''_{25}	$(q'_{32}, a, 1)\,(4, \lambda, 1)$	$(q'_1, a, 1)\,(1, \lambda, 1)$,
q''_{27}	$(q'_1, a, 1)\,(1, \lambda, 1)(0, a, 0)$	$(q'_{32}, a, 1)\,(4, \lambda, 1)$,
q''_{32}	$(q_h, a, 0)$	$(q'_1, a, 1)\,(1, \lambda, 1)$.

This construction uses a total of $8 + 2 \times 13 + 1 =$ **35** neurons. The halting neuron q_h can be avoided, e.g., by changing the right side of the rule with $q''_{32} : (0, a)$ to, e.g., $(4, \lambda, 1)$. Indeed, the register machine halts with register 4 being empty, so the P system will halt after neuron 4 has reset its state to 0. We also remark that this construction does not respect the requirement of all states on the right side being equal. This requirement can be fulfilled by replacing $(r, a, 0)$ by $(r', a, 1)$ for $r \in \{0, 1, 4, 5, 6, 7\}$ and $(2, a, 0)(3, a, 0)$ by $(\langle 2, 3\rangle', a, 1)$ in the right sides of the rules above, and adding 7 additional neurons, each having one rule: $r' : (1, a) \to (r, a, 0)$ for $r \in \{0, 1, 4, 5, 6, 7\}$ and the neuron $\langle 2, 3\rangle$ with the rule $\langle 2, 3\rangle' : (1, a) \to (2, a, 0)(3, a, 0)$, yielding a total of **54** rules.

If we consider the most restricted variant of spiking neural P systems with states elaborated at the end of Section 4, which is comparable with the model considered in [16] using the notion of *polarizations* instead of the notion *states*, when again embedding the first step of the simulation of a generalized SUB-instruction into the last step of the preceding simulation, a straight-forward calculation yields two neurons and two rules per register as well as 7 neurons and 9 rules per generalized conditional decrement instruction, which yields a total of $16 + 7 \times 13 = \mathbf{107}$ neurons as well as $16 + 9 \times 13 = \mathbf{133}$ rules.

It only remains to argue the correctness of the construction. As before, embedded ADD-instructions translate exactly into sending additional spikes to register neurons by existing rules simulating the SUB-instructions. Hence, we only explain the latter, i.e., the decrement case and the zero-test case.

Suppose we simulate a rule $p : (\mathtt{SUB}(r), q, s)$, and the next instruction performs the $\mathtt{SUB}\,(r_q)$ or the $\mathtt{SUB}\,(r_s)$ command, respectively. The configuration of the P system consists of the description of (the state of and the number of spikes in) each of its neurons. Clearly, the neurons in state 0 and without spikes present no interest; we also omit the register neurons different from r. By l we denote any element of SUB_r other than r.

Furthermore, we underline the spikes which are to be removed from the firing neuron, and we will highlight in bold the neurons which are idle and whose state switches from 1 to 0.

Step	p	$\tilde{p}$	$\tilde{p}'$	$\hat{p}$	$\hat{p}'$	$\hat{p}''$	r	r'	$\tilde{l}'$	$\hat{l}''$
1	$(1, \underline{a})$						$(0, a^{n_r})$			
2		$(1, \underline{a})$		$(1, \underline{a})$			$(1, a^{n_r-1+\underline{2}})$			
3			$(1, \underline{a^2})$		$(1, \underline{a})$		$(0, a^{n_r-1})$	$(1, \underline{a})$	$(\mathbf{1}, \mathbf{a})$	
4						$(1, \underline{a^2})$	$(0, a^{n_r-1})$		$(0, \underline{a})$	$(1, \underline{a^2})$
5							$(0, a^{n_r-1})$			

Fig. 7. The trace of the **decrement case**.

The trace of the decrement case is presented in Figure 7. What is not reflected on this figure is that line 4 of the trace of instruction p is superposed with line 2 of instruction q. However, since the neurons are disjoint, no interference happens. Indeed, the last step of simulation of instruction p removes the spikes from neurons $\hat{p}''$, $\hat{l}''$, and $\tilde{l}'$, while the simulation of instruction q acts upon neurons $\tilde{q}$, $\hat{q}$ and r_q. We observe that (if $p \neq q$) a spike is removed from $\tilde{q}'$ simultaneously with a spike being sent to $\tilde{q}'$, which of course causes no interference, since the spikes do not meet. As mentioned before, the first step is omitted (by embedding it into the initial configuration and into every preceding step), so the simulation starts at line 2, taking just 2 steps to produce line 2 for the next instruction, and one more step to remove the superfluous spikes.

Step	p	$\tilde{p}$	$\tilde{p}'$	$\hat{p}$	$\hat{p}'$	$\hat{p}''$	$\bar{p}$	$\bar{p}'$	r	r'	$\tilde{l}'$	$\hat{l}''$
1	$(1, \underline{a})$											
2		$(1, \underline{a})$		$(1, \underline{a})$					$(\mathbf{1}, \mathbf{a})$			
3			$(\mathbf{1}, \mathbf{a})$		$(1, \underline{a})$				$(0, a)$			
4			$(0, \underline{a})$			$(0, \underline{a})$			$(0, a)$			
5							$(1, \underline{a})$		$(1, \underline{a^2})$			
6			$(\mathbf{1}, \mathbf{a})$					$(1, \underline{a})$		$(1, \underline{a})$	$(\mathbf{1}, \mathbf{a})$	
7			$(0, \underline{a})$			$(1, \underline{a^2})$					$(0, \underline{a})$	$(1, \underline{a^2})$
8												

Fig. 8. The trace of the **zero-test case**.

The trace of the zero-test case is presented in Figure 8. Similarly to the decrement case, it does not reflect that line 7 of the trace of instruction p is superposed with line 2 of instruction s. Hence, the arguments about the correctness of the decrement case also hold for the zero-test case, with the following differences. The last step of simulation of instruction p also removes two spikes from neuron $\tilde{p}'$. As mentioned before, the first step is omitted, so the simulation starts at line 2, taking 5 steps to produce line 2 for the next instruction, and one more step to remove the superfluous spikes. This completes the explanation of the correctness.

It is easy to see that forgetting rules may be replaced by sending a spike to one additional dummy neuron.

6 Conclusion

We have shown that only two states (or polarizations as they are called in [16]) are needed for obtaining computational completeness with (extended) spiking neural P systems with states, thus solving an open problem raised at the Brainstorming Week on Membrane Computing in Sevilla at the beginning of February 2016.

References

1. A. Alhazov and R. Freund. Variants of small universal P systems with catalysts. *Fundamenta Informaticae*, 138(1-2):227–250, 2015.
2. A. Alhazov, R. Freund, S. Ivanov, M. Oswald, and S. Verlan. Extended spiking neural P systems with white holes. In L. Macías-Ramos, Gh. Păun, A. Riscos-Núñez, and L. Valencia-Cabrera, editors, *Proceedings of the 13th Brainstorming Week on Membrane Computing*, pages 45–62. Fénix Editora, Sevilla, 2015.
3. A. Alhazov, R. Freund, M. Oswald, and M. Slavkovik. Extended spiking neural P systems. In *Workshop on Membrane Computing*, volume 4361 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2006.

4. H. Chen, R. Freund, M. Ionescu, Gh. Păun, and M. Pérez-Jiménez. On string languages generated by spiking neural P systems. In *Prooceedings of the Fourth Brainstorming Week on Membrane Computing*, volume 1, pages 169–194. Fénix Editora, Sevilla, 2006.
5. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, and M. J. Pérez-Jiménez. Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7(2):147–166, 2007.
6. R. Freund and M. Oswald. A short note on analysing P systems. *Bulletin of the EATCS*, 78:231–236, 2002.
7. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2,3):279–308, 2006.
8. I. Korec. Small universal register machines. *Theoretical Computer Science*, 168:267–301, 1996.
9. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
10. Gh. Păun. *Membrane Computing: An Introduction*. Natural Computing Series Natural Computing. Springer, 2002.
11. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.
12. Gh. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
13. Gh. Păun, Y. Sakakibara, and T. Yokomori. P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60:635–660, 2006.
14. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1-3. Springer, 1997.
15. The P Systems Website.
16. T. Wu, A. Păun, Z. Zhang, and L. Pan. Spiking neural P systems with polarizations. *submitted*, 2015.