The Microsoft Research - University of Trento Centre for Computational and Systems Biology

Technical Report CoSBi 09/2007

Asynchronous Spiking Neural P Systems

Matteo Cavaliere

Microsoft Research-University of Trento
Centre for Computational and Systems Biology
Trento, Italy

cavaliere@cosbi.eu

Omer Egecioglu, Oscar H. Ibarra, Sara Woodworth

Department of Computer Science, University of California
Santa Barbara, USA

omer, ibarra, swood@cs.ucsb.edu

Mihai Ionescu

Research Group on Mathematical Linguistics, Universitat Rovira i Virgili
Tarragona, Spain

armandmihai.ionescu@urv.net

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
Bucharest, Romania, and
Department of Computer Science and AI, University of Sevilla
Sevilla, Spain

george.paun@imar.ro, gpaun@us.es

This is the preliminary version of a paper that will appear in
Theoretical Computer Science, 410, 24-25, 2009.
available at http://www.elsevier.com

Abstract

We consider here spiking neural P systems with a non-synchronized (i.e., asynchronous) use of rules: in any step, a neuron can apply or not apply its rules which are enabled by the number of spikes it contains (further spikes can come, thus changing the rules enabled in the next step). Because the time between two firings of the output neuron is now irrelevant, the result of a computation is the number of spikes sent out by the system, not the distance between certain spikes leaving the system. The additional non-determinism introduced in the functioning of the system by the non-synchronization is proved not to decrease the computing power in the case of using extended rules (several spikes can be produced by a rule). That is, we obtain again the equivalence with Turing machines (interpreted as generators of sets of (vectors of) numbers). However, this problem remains open for the case of restricted spiking neural P systems, whose rules can only produce one spike. On the other hand we prove that asynchronous systems, with extended rules, and where each neuron is either bounded or unbounded, are not computationally complete.

For these systems, the configuration reachability, membership (in terms of generated vectors), emptiness, infiniteness, and disjointness problems are shown to be decidable. However, containment and equivalence are undecidable.

1 Introduction

Spiking neural P systems (SN P systems, for short) were introduced in [11] with the aim of incorporating specific ideas from spiking neurons into membrane computing. Currently, neural computing based on spiking is a field that is being heavily investigated (see, e.g., [5], [13], [14]).

In short, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along the arcs of the graph (they are called *synapses*). Thus, the architecture is that of a tissue-like P system, with only one kind of object present in the cells (the reader is referred to [18] for an introduction to membrane computing and to [23] for the up-to-date information about this research area). The objects evolve by means of *standard spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes. *Extended rules* were considered in [4], [17]: these rules are of the form $E/a^c \to a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain a generalization of both standard spiking and forgetting rules.

In this paper we consider extended spiking rules with restrictions on the type of the regular expressions used. In particular, we consider two types of rules. The first type are called *bounded rules* and are of the form $a^i/a^c \to a^p; d$, where $1 \leq c \leq i$, $p \geq 0$, and $d \geq 0$. We also consider *unbounded rules* of the form $a^i(a^j)^*/a^c \to a^p; d$, where $i \geq 0, j \geq 1, c \geq 1, p \geq 0, d \geq 0$. A neuron is called *bounded* if it has only bounded rules, while it is *unbounded* if it has only unbounded rules. A neuron is then called *general* if it has both bounded and unbounded rules. An SN P system is called bounded if it has only bounded neurons, while it is called unbounded if each neuron is either bounded or unbounded. A general SN P system is a system with general neurons. It was shown in [10] that general SN P systems are universal.

An SN P system (of any type) works in the following way. A global clock is assumed, and in each time unit, each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when (at least) a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it is infinite if the computation does not stop.

With a spike train we can associate various numbers, which can be considered as *computed* (we also say *generated*) by an SN P system. For instance, in [11] only the distance between the first two spikes of a spike train was considered, then in [20] several extensions were examined: the distance between the first k spikes of a spike train, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, all computations or only halting computations, etc.

An SN P system can also work in the accepting mode: a neuron is designated as the *input neuron* and two spikes are introduced in it, at an interval of n steps; the number n is accepted if the computation halts.

Two main types of results were obtained (for general systems, with standard rules): computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semilinear sets of numbers in the case when a bound was imposed. In [11] it is proved that synchronized SN P systems using standard rules characterize NRE; improvements in the form of the regular expressions, removing the delay, or the forgetting rules can be found in [10]. The result is true both for the generative and the accepting case.

In the proof of these results, the synchronization plays a crucial role, but both from a mathematical point of view and from a neuro-biological point of view it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory. Even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used. The neuron may choose to remain unfired, maybe receiving spikes from the neighboring neurons. If the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused. If the new spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now).

This way of using the rules applies also to the output neuron, so the distance in time between the spikes sent out by the system is no longer relevant. Hence, for non-synchronized SN P systems, the result of the computation is the total number of spikes sent out to the environment. This makes it necessary to consider only halting computations. (The computations which do not halt are ignored and provide no output.)

We stress the fact that we count *all* spikes sent out. A possibility which we do not consider is to only count the steps when at least one spike exits the system. Moreover, it is also possible to consider systems with several output neurons. In this case one counts the spikes emitted by the output neurons and collect them as vectors.

The synchronization is in general a powerful feature, useful in controlling the work of a computing device. However, it turns out that the loss in power entailed by removing the synchronization is compensated in the case of general SN P systems where extended rules are used. In fact, we prove that such systems are still equivalent with Turing machines (as generators of sets of (vectors of) natural numbers).

On the other hand, we also show that a restriction which looks, at first sight, rather minor, has a crucial influence on the power of the systems and decreases their computing power: specifically, we prove that unbounded SN P systems are not computationally complete (as mentioned above, for bounded systems this result is already known from [11]).

Moreover, for unbounded systems, the configuration reachability, membership (in terms of generated vectors), emptiness, infiniteness, and disjointness problems can be decided. However, containment and equivalence are undecidable. Note that, for general SN P systems, even reachability and membership are undecidable, because these systems are universal (in a constructive way).

However, universality remains *open* for non-synchronized SN P systems using standard rules. We find this problem worth investigating (a non-universality result – as we expect it will be the case – can show an interesting difference between synchronized and non-synchronized devices, with the loss in power compensated by the additional "programming capacity" of extended rules). The non-synchronized case remains to be considered also for other issues specific to SN P systems, such as looking for small universal systems as in [17], for normal forms as in [10], for generating languages or processing finite or infinite sequences, [3], [4], [21], characterizations of multi-dimensional semilinear sets of numbers as in [8], using the rules in the in exhaustive mode, as in [12], etc.

Another mode of computation of an SN P system that has been studied earlier [9] is the sequential mode. In this mode, at every step of the computation, if there is at least one neuron with at least one rule that is fireable, we only allow one such neuron and one such rule (both chosen non-deterministically) to be fired. It was shown in [9] that certain classes of sequential SN P systems are equivalent to partially blind counter machines, while others are universal.

2 Prerequisites

We assume the reader to have some familiarity with (basic elements of) language and automata theory, e.g., from [22], and introduce only a few notations and the definitions related to SN P systems (with extended rules).

For an alphabet V, V^* is the free monoid generated by V with respect to the concatenation operation and the identity λ (the empty string); the set of all nonempty strings over V, that is, $V^* - \{\lambda\}$, 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 (it is the family of length sets of recursively enumerable languages) and the family of Turing computable sets of vectors of natural numbers is denoted by $PsRE$.

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

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

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);

2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:

a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;

b) R_i is a finite set of *extended rules* of the following form:

$$E/a^c \to a^p; d,$$

where E is a regular expression with a the only symbol used, $c \geq 1$, and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$, too.

3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);

4. $out \in \{1, 2, \dots, m\}$ indicates the *output neuron.*

A rule $E/a^c \to a^p; d$ with $p \geq 1$ is called *extended firing* (we also say *spiking*) *rule*; a rule $E/a^c \to a^p; d$ with $p = d = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule.* If $L(E) = \{a^c\}$, then the rules are written in the simplified form $a^c \to a^p; d$ and $a^c \to \lambda$. A rule of the type $E/a^c \to a; d$ and $a^c \to \lambda$ is said to be *restricted* (or *standard*).

In this paper, we investigate extended spiking rules using particular types of regular expressions.

A rule is *bounded* if it is of the form $a^i/a^c \to a^p; d$, where $1 \leq c \leq i$, $p \geq 0$, and $d \geq 0$. A neuron is *bounded* if it contains only bounded rules. A rule is called *unbounded* if is of the form $a^i(a^j)^*/a^c \to a^p; d$, where $i \geq 0, j \geq 1, c \geq 1, p \geq 0, d \geq 0$. (In all cases, we also assume $c \geq p$; this restriction rules out the possibility of "producing more than consuming", but it plays no role in arguments below and can be omitted.) A neuron is *unbounded* if it contains only unbounded rules. A neuron is *general* if it contains both bounded and unbounded rules. An SN P system is *bounded* if all the neurons in the system are bounded. It is *unbounded* if it has bounded *and* unbounded neurons. Finally, an SN P system is *general* if it has general neurons (i.e., it contains at least one neuron which has both bounded and unbounded rules).

One can allow rules like $\alpha_1 + \ldots + \alpha_n \to a^p; d$ in the neuron, where all α_i's have bounded (resp., unbounded) regular expressions as defined above. But such a rule is equivalent to putting n rules $\alpha_i \to a^p; d$ $(1 \leq i \leq n)$ in the neuron. Moreover, it is known that any regular set over a 1-letter symbol a can be expressed as a finite union of regular sets of the form $\{a^i(a^j)^k \mid k \geq 0\}$ for some $i, j \geq 0$. Note that such a set is finite if $j = 0$.

The rules are applied as follows: if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a^p; d \in R_i$ is enabled and it can be applied. This means that c spikes are consumed, $k - c$ spikes remain in the neuron, the neuron is fired, and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately, if $d = 1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t+1, t+2, \dots, t+d-1$ the neuron is *closed*; this means that during these steps it uses no rule and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In step $t + d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t + d + 1$). The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one of the form $E/a^c \to \lambda$ then, when it is applied, $c \geq 1$ spikes are removed.

In the synchronized mode, considered up to now in the SN P systems investigations, a global clock is considered, marking the time for all neurons, and in each time unit, in each neuron which can use a rule, a rule should be used. Because two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ 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 then one of them is chosen non-deterministically. Note that the neurons work in parallel (synchronously), but each neuron processes sequentially its spikes, using only one rule in each time unit.

In the non-synchronized case considered here the definition of a computation in an SN P system is easy: in each time unit, *any neuron is free to use a rule or not.* Even if enabled, a rule is not necessarily applied, the neuron can remain still in spite of the fact that it contains rules which are enabled by its contents. If the contents of the neuron is not changed, a rule which was enabled in a step t can fire later. If new spikes are received, then it is possible that other rules will be enabled – and applied or not.

It is important to point out that when a neuron spikes, its spikes immediately leave the neuron and reach the target neurons simultaneously (as in the synchronized systems, there is no time needed for passing along a synapse from one neuron to another neuron).

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ representing the initial number of spikes present in each neuron. Using the rules as suggested above, we can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation.*

A computation is *successful* if it reaches a configuration where all bounded and unbounded neurons are open but none is fireable (i.e., the SN P system has halted)

Because now "the time does not matter", the spike train can have arbitrarily many occurrences of 0 between any two occurrences of 1, hence the result of a computation can no longer be defined in terms of the steps between two consecutive spikes as in the standard SN P system definition. That is why, the result of a computation is defined here as the total number of spikes sent into the environment by the output neuron.

Specifically, a number x is then generated by the SN P system if there is a successful computation of the system where the output neuron emits exactly x spikes (if several spikes are emitted by the output neuron, at the same time, all of them are counted). Because of the non-determinism in using the rules, a given system computes in this way a *set of numbers.*

Successful computations which send no spike out can be considered as generating number zero, but in what follows we adopt the convention to ignore number zero when comparing the computing power of two devices.

Of course, a natural definition of the result of a computation can also be the number of spikes present in a specified neuron in the halting configuration. This is much closer to the traditional style of membrane computing, but there is no difference with respect to the previous definition: consider an additional neuron, which receives the spikes emitted by the previous output neuron and has no rule inside. When the computation halts, the contents of this additional neuron is the result of the computation.

SN P systems can also be used for generating sets of vectors, by considering several output neurons, $\sigma_{i_1}, \ldots, \sigma_{i_k}$. In this case, the system is called a *k-output SN P system.* Here a vector of numbers, $(n_1, \ldots, n_k)$, is generated by counting the number of spikes sent out by neurons $\sigma_{i_1}, \ldots, \sigma_{i_k}$ respectively during a successful computation.

We denote by $N^{nsyn}_{gen}(\Pi)$ $[Ps^{nsyn}_{gen}(\Pi)]$ the set [the set of vectors, resp.] of numbers generated in the non-synchronized way by a system Π, and by $NSpik_{tot}EP^{nsyn}_m(\alpha, del_d)$ $[PsSpik_{tot}EP^{nsyn}_m(\alpha, del_d)]$, $\alpha \in \{gen, unb, boun\}, d \geq 0$, the family of such sets of numbers [sets of vectors of numbers, resp.] generated by systems of type α (*gen* stands for general, *unb* for unbounded, *boun* for bounded), with at most m neurons and rules having delay at most d. (The subscript *tot* reminds us of the fact that we count all spikes sent to the environment.)

A *0-delay SN P system* is one where the delay in all the rules of the neurons is zero. Because in this paper we always deal with 0-delay systems, the delay ($d = 0$) is never specified in the rules.

An SN P system working in the non-synchronized manner can also be used in the accepting way: a number n is introduced in the system, in the form of n spikes placed in a distinguished *input neuron*, and if the computation eventually stops, then n is accepted. In what follows we will only occasionally mention the accepting case. Because there is no confusion, in this paper, non-synchronized SN P systems are often simply called SN P systems.

The examples from the next section will illustrate and clarify the above definitions.

3 Three Examples

In order to clarify the previous definitions, we start by discussing some **examples**, which are also of an interest *per se*. In this way, we also introduce the standard way to pictorially represent a configuration of an SN P system, in particular, the initial configuration. Specifically, each neuron is represented by a "membrane", marked with a label and having inside both the current number of spikes (written explicitly, in the form a^n for n spikes present in a neuron) and the evolution rules. The synapses linking the neurons are represented by directed edges (arrows) between the membranes. The output neuron is identified by both its label, *out*, and pictorially by a short arrow exiting the membrane and pointing to the environment. pointing to the environment.

Example 1. The first example is the system Π_1 given in Figure 1. We have only two neurons, initially each of them containing one spike. In the synchronized manner, Π_1 works forever, with both neurons using a rule in each step – hence the output neuron sends one spike out in each step, i.e., the spike train is the infinite sequence of symbols 1, written 1^ω.

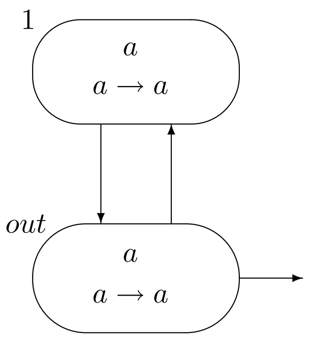

Figure 1: An example of an SN P system where synchronization matters

In the non-synchronized mode, the system can halt in any moment: each neuron can wait an arbitrary number of steps before using its rule; if both neurons fire at the same time, then the computation continues, if not, one neuron consumes its spike and the other one gets two spikes inside and can never use its rule.

Consequently, $N^{nsyn}_{gen}(\Pi_1) = \mathbf{N}$, the set of natural numbers.

It is worth noting that synchronized systems with one or two neurons characterize the finite sets of numbers (see [11]), hence we already have here an essential difference between the two modes of using the rules: in the non-synchronized mode, systems with two neurons can generate infinite sets of numbers.

Example 2. The two neurons of the system above can be synchronized by means of a third neuron even when they do not work synchronously, and this is shown in Figure 2.

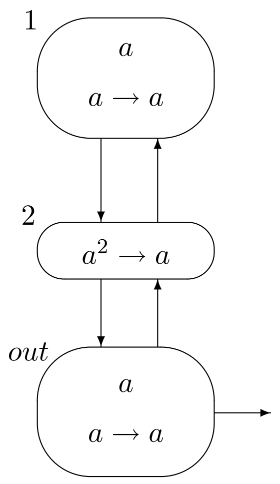

Figure 2: An SN P system functioning in the same way in both modes

This time, the intermediate neuron σ_2 stores the spikes produced by the two neurons σ_1, σ_{out}, so that only after both these neurons spike they receive spikes back. Both in the synchronized and the non-synchronized way, this system never halts, and the number of spikes sent out is infinite in both cases.

Example 3. A slight (at the first sight) change in the neuron σ_2 from the previous example will lead to a much more intricate functioning of the system – this is the case with the system Π_3 from Figure 3.

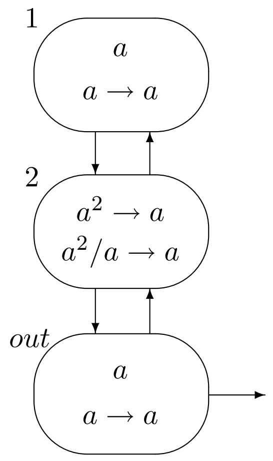

Figure 3: A version of the system from Figure 2

The system behaves like that from Figure 2 as long as neuron σ_2 uses the rule $a^2 \to a$.

If, instead, rule $a^2/a \to a$ is used, then either the computation stops (if both σ_1 and σ_{out} spike, then σ_2 will get 3 spikes and will never spike again), or it continues working forever. In this latter case, there are two possibilities: σ_2 will cooperate with σ_1 or with σ_{out} (the neuron which spikes receives one spike back, but the other one gets two spikes and is blocked; σ_2 continues by using the rule $a^2/a \to a$, otherwise the computation halts, because σ_2 will get next time only one spike). If the computation continues between σ_2 and σ_1, then no spike will be sent outside; if the cooperation is between σ_2 and σ_{out}, then the system sends out an arbitrary number of spikes.

Again the number of spikes sent out is the same both in the synchronized and the non-synchronized modes (the generated set is again $\mathbf{N}$), but the functioning of the system is rather different in the two modes.

4 Computational Completeness of General SN P Systems

We pass now to prove that the power of general neurons (where extended rules, producing more than one spike at a time, are used) can compensate the loss of power entailed by removing the synchronization.

In the following proof we use the characterization of NRE by means of multicounter machines (abbreviated CM and also called register machines) [15].

Such a device – in the non-deterministic version – is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of counters, 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.

When it is useful, a label can be seen as a *state* of the machine, l_0 being the initial state, l_h the final/accepting state.

The labeled instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to counter r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if counter r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A counter machine M generates a set $N(M)$ of numbers in the following way: we start with all counters empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of counters). If we reach the halt instruction, then the number n present in counter 1 at that time is said to be generated by M. It is known (see, e.g., [15]) that counter machines generate all sets of numbers which are Turing computable.

A counter machine can also accept a set of numbers: a number n is accepted by M if, starting with n in counter 1 and all other counters empty, the computation eventually halts (without loss of generality, we may assume that in the halting configuration all counters are empty). Deterministic counter machines (i.e., with ADD instructions of the form $l_i : (\texttt{ADD}(r), l_j)$) working in the accepting mode are known to be equivalent with Turing machines.

It is also possible to consider counter machines producing sets of vectors of natural numbers. In this case a distinguished set of k counters (for some $k \geq 1$) is designated as the output counters. A k-tuple $(n_1, \ldots, n_k) \in \mathbf{N}^k$ is generated if M eventually halts and the contents of the output counters are $n_1, \ldots, n_k$, respectively.

Without loss of generality we may assume that in the halting configuration all the counters, except the output ones, are empty. We also assume (without loss of generality) that the output counters are non-decreasing and their contents are only incremented (i.e., the output counters are never the subject of SUB instructions, but only of ADD instructions).

We will refer to a CM with k output counters (the other counters are auxiliary counters) as a k-output CM. It is well known that a set S of k-tuples of numbers is generated by a k-output CM if and only if S is recursively enumerable. Therefore they characterize $PsRE$. We shall refer to a 1-output CM as simply a CM.

Theorem 1 $Spik_{tot}EP_*^{nsyn}(gen, del_0) = NRE$.

Proof. We only have to prove the inclusion $NRE \subseteq Spik_{tot}EP_*^{nsyn}(gen, del_0)$, and to this aim, we use the characterization of NRE by means of counter machines used in the generating mode.

Let $M = (m, H, l_0, l_h, I)$ be a counter machine with m counters, having the properties specified above: the result of a computation is the number from counter 1 and this counter is never decremented during the computation.

We construct a spiking neural P system Π as follows.

For each counter r of M let t_r be the number of instructions of the form $l_i : (\mathtt{SUB}(r), l_j, l_k)$, i.e., all SUB instructions acting on counter r (of course, if there is no such a SUB instruction, then $t_r = 0$, which is the case for $r = 1$). Denote

$$T = 2 \cdot \max\{t_r \mid 1 \leq i \leq m\} + 1.$$

For each counter r of M we consider a neuron σ_r in Π whose contents correspond to the contents of the counter. Specifically, if the counter r holds the number $n \geq 0$, then the neuron σ_r will contain $3Tn$ spikes.

With each label l of an instruction in M we also associate a neuron σ_l. Initially, all these neurons are empty, with the exception of the neuron σ_{l_0} associated with the start label of M, which contains $3T$ spikes. This means that this neuron is "activated". During the computation, the neuron σ_l which receives $3T$ spikes will become active. Thus, simulating an instruction $l_i : (\mathtt{OP}(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the counter r as requested by $\mathtt{OP}$, then introducing $3T$ spikes in one of the neurons $\sigma_{l_j}, \sigma_{l_k}$, which becomes in this way active. When activating the neuron σ_{l_h}, associated with the halting label of M, the computation in M is completely simulated in Π; we will then send to the environment a number of spikes equal to the number stored in the first counter of M. Neuron σ_1 is the output neuron of the system.

Further neurons will be associated with the counters and the labels of M in a way which will be described immediately. All of them are initially empty.

The construction itself is not given in symbols, but we present modules associated with the instructions of M (as well as the module for producing the output) in the graphical form introduced in the previous section. These modules are presented in Figures 4, 5, 6. Before describing these modules and their work, let us remember that the labels are injectively associated with the instructions of M, hence each label precisely identifies one instruction, either an ADD or a SUB one, with the halting label having a special situation – it will be dealt with by the FIN module. Remember also that counter 1 is never decremented.

As mentioned before, because the system we construct has only rules with delay 0, the delay is not specified in the figures below.

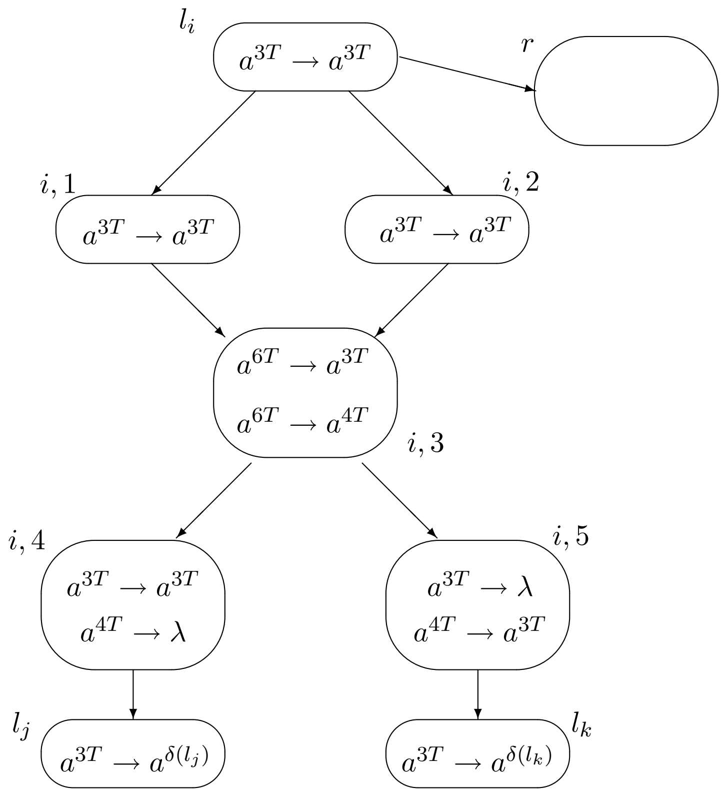

Figure 4: Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ – module ADD (Figure 4).

The initial instruction, that labeled with l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, with $3T$ spikes present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neuron, except those neurons associated with the counters. Having $3T$ spikes inside, neuron σ_{l_i} can fire, and at some time it will do it, producing $3T$ spikes. These spikes will simultaneously go to neurons $\sigma_{i,1}$ and $\sigma_{i,2}$ (as well as to neuron σ_r, thus simulating the increase of the value of counter r with 1). Also these neurons can spike at any time. If one of them is doing it, then $3T$ spikes arrive in neuron $\sigma_{i,3}$, which cannot use them. This means that neuron $\sigma_{i,3}$ must wait until further $3T$ spikes come from that neuron $\sigma_{i,1}, \sigma_{i,2}$ which fired later. With $6T$ spikes inside, neuron $\sigma_{i,3}$ can fire, by using one of its rules, non-deterministically chosen. These rules determine the non-deterministic choice of the neuron $\sigma_{l_j}, \sigma_{l_k}$ to activate. If, for instance, the rule $a^{6T} \to a^{3T}$ was used, then both $\sigma_{i,4}$ and $\sigma_{i,5}$ receive $3T$ spikes. Only $\sigma_{i,4}$ can use them for spiking, while $\sigma_{i,5}$ can forget them. Eventually $\sigma_{i,4}$ fires, otherwise the computation does not halt. If this ADD instruction is simulated again and further spikes are sent to neuron $\sigma_{i,5}$ although it has not removed its spikes, then it will accumulate at least $6T$ spikes and will never fire again. This means that no "wrong" step is done in the system Π because of the non-synchronization. If in $\sigma_{i,3}$ one uses the rule $a^{6T} \to a^{4T}$, then the computation proceeds in a similar way, eventually activating neuron σ_{l_k}. Consequently, the simulation of the ADD instruction is possible in Π and no computation in Π will end and will provide an output (see also below) if this simulation is not correctly completed.

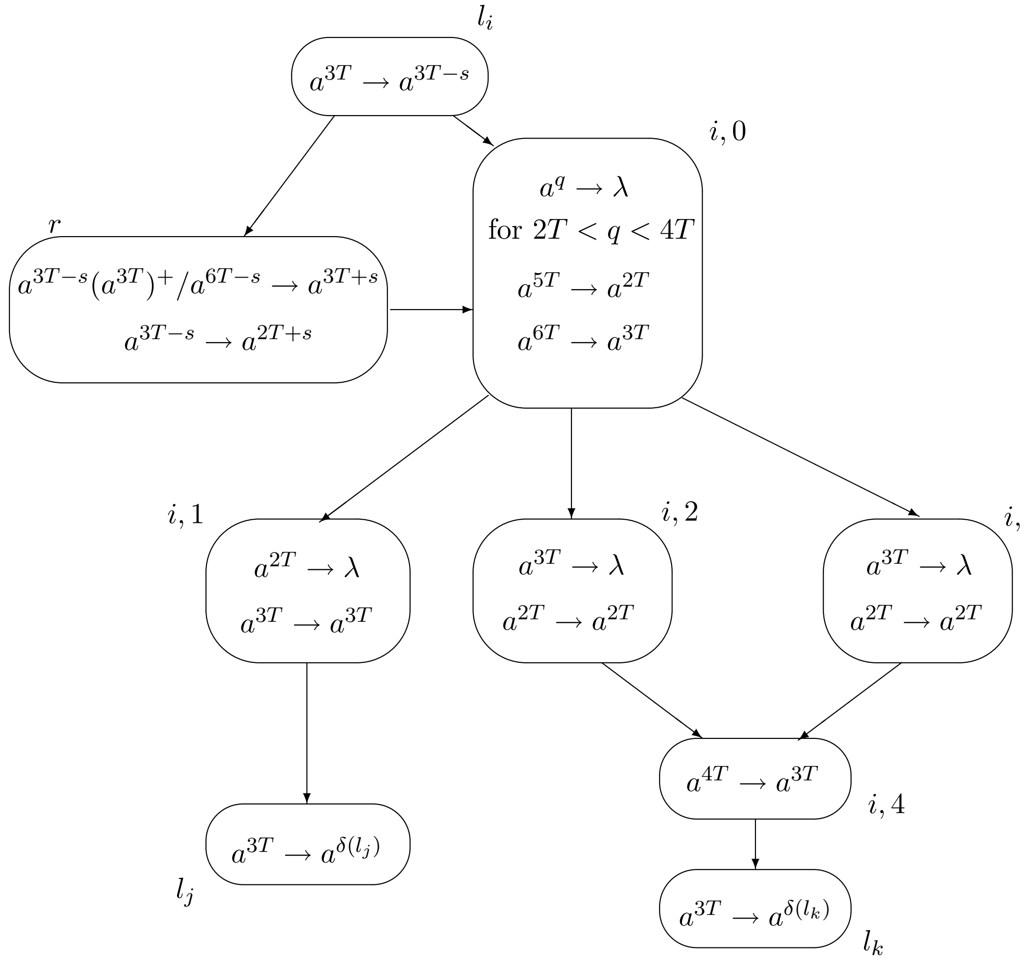

Figure 5: Module SUB (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$)

Simulating a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ – module SUB (Figure 5).

Let us examine now Figure 5, starting from the situation of having $3T$ spikes in neuron σ_{l_i} and no spike in other neurons, except neurons associated with counters; assume that neuron σ_r holds a number of spikes of the form $3Tn$, $n \geq 0$. Assume also that this is the sth instruction of this type dealing with counter r, for $1 \leq s \leq t_r$, in a given enumeration of instructions (because l_i precisely identifies the instruction, it also identifies s).

Sometimes, neuron σ_{l_i} spikes and sends $3T - s$ spikes both to σ_r and to $\sigma_{i,0}$. These spikes can be forgotten in this latter neuron, because $2T < 3T - s < 4T$. Sometimes, also neuron σ_r will fire, and will send $2T + s$ of $3T + s$ spikes to neuron $\sigma_{i,0}$. If no spike is here, then no other action can be done, also these spikes will eventually be removed, and no continuation is possible (in particular, no spike is sent out of the system; remember that number zero is ignored, hence we have no output in this case).

If neuron $\sigma_{i,0}$ does not forget the spikes received from σ_{l_i} (this is possible, because of the non-synchronized mode of using the rules), then eventually neuron σ_r will send here either $3T+s$ spikes – in the case where it contains more than $3T-s$ spikes (hence counter r is not empty), or $2T+s$ spikes – in the case where its only spikes are those received from σ_{l_i}. In either case, neuron $\sigma_{i,0}$ accumulates more than $4T$ spikes, hence it cannot forget them.

Depending on the number of spikes accumulated, either $6T$ or $5T$, neuron $\sigma_{i,0}$ eventually spikes, sending $3T$ or $2T$ spikes, respectively, to neurons $\sigma_{i,1}$, $\sigma_{i,2}$, and $\sigma_{i,3}$. The only possible continuation of neuron $\sigma_{i,1}$ is to activate neuron σ_{l_j} (precisely in the case where the first counter of M was not empty). Neurons $\sigma_{i,2}$ and $\sigma_{i,3}$ will eventually fire and either forget their spikes or send $4T$ spikes to neuron $\sigma_{i,4}$, which activates neuron σ_{l_k} (in the case where the first counter of M was empty).

It is important to note that if any neuron $\sigma_{i,u}, u = 1, 2, 3$, skips using the rule which is enabled and receives further spikes, then no rule can be applied there anymore and the computation is blocked, without sending spikes out.

The simulation of the SUB instruction is correct in both cases, and no "wrong" computation is possible inside the module from Figure 5.

What remains to examine is the possible interferences between modules.

First, let us consider the easy issue of the exit labels of the instructions of M, which can be labels of either ADD or SUB instructions, or can be l_h. To handle this question, in both the ADD and the SUB modules we have written the rules from the neurons $\sigma_{l_j}, \sigma_{l_k}$ in the form $a^{3T} \to a^{\delta(l_u)}$, where δ is the function defined on H as follows:

$$\delta(l) = \begin{cases} 3T, & \text{if } l \text{ is the label of an ADD instruction,} \\ 3T - s, & \text{if } l \text{ is the label of the } s\text{th SUB instruction} \\ & \text{dealing with a counter } r \text{ of } M, \\ 1 & \text{if } l = l_h. \end{cases}$$

What is more complicated is the issue of passing spikes among modules, but not through the neurons which correspond to labels of M. This is the case with the neurons σ_r for which there are several SUB instructions, and this was the reason of considering the number T in writing the contents of neurons and the rules. Specifically, each σ_r for which there exist t_r SUB instructions can send spikes to all neurons $\sigma_{i,0}$ as in Figure 5. However, only one of these target neurons also receives spikes from a neuron σ_{l_i}, the one identifying the instruction which we want to simulate.

Assume that we simulate the sth instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, hence neuron σ_r sends $3T+s$ or $2T+s$ spikes to all neurons of the form $\sigma_{i',0}$ for which there is an instruction $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$ in M. These spikes can be forgotten, and this is the correct continuation of the computation (note that $2T < 2T + s < 3T + s < 4T$ hence there is a forgetting rule to apply in each $\sigma_{i',0}$). If these spikes are not forgotten and at a subsequent step of the computation neuron $\sigma_{i',0}$ receives further spikes from the neuron σ_r (the number of received spikes is $3T+s'$ or $2T+s'$, for some $1 \leq s' \leq t_r$), then we accumulate a number of spikes which will be bigger than $4T$ (hence no forgetting rule can be used) but not equal to $5T$ or $6T$ (hence no firing rule can be used). Similarly, if these spikes are not forgotten and at a subsequent step of the computation the neuron $\sigma_{i',0}$ receives spikes from the neuron $\sigma_{l_{i'}}$ (which is associated with $\sigma_{i',0}$ in a module SUB as in Figure 5), then again no rule can ever be applied here: if $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$ is the s'th SUB instruction acting on counter

r, then $s \neq s'$ and the neuron accumulates a number of spikes greater than $4T$ (we receive $3T - s'$ spikes from $\sigma_{l_{i'}}$) and different from $5T$ and $6T$. Consequently, no computation can use the neurons $\sigma_{i',0}$ if they do not forget the spikes received from σ_r. This means that the only computations in Π which can reach the neuron σ_{l_h} associated with the halting instruction of M are the computations which correctly simulate the instructions of M and correspond to halting computations in M.

Ending a computation – module FIN (Figure 6).

When the neuron σ_{l_h} is activated, it (eventually) sends one spike to neuron σ_1, corresponding to the first counter of M. From now on, this neuron can fire, and it sends out one spike for each $3T$ spikes present in it, hence the system will emit a number of spikes which corresponds to the contents of the first counter of M in the end of a computation (after reaching instruction l_h : `HALT`).

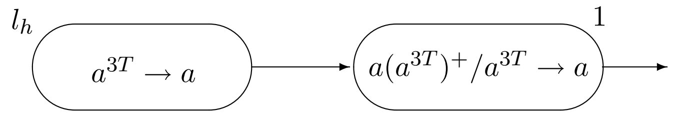

Figure 6: Module FIN (ending the computation)

Consequently, $N_{gen}^{nsyn}(\Pi) = N(M)$ and this completes the proof. □

Clearly, the previous construction is the same for the accepting mode, and can be carried out for deterministic counter machines (the ADD instructions are of the form l_i : (`ADD`$(r), l_j$)), hence also the obtained system is deterministic. Similarly, if the result of a computation is defined as the number of spikes present in a specified neuron in the halting configuration, then the previous construction is the same, we only have to add one further neuron which is designated as the output neuron and which collects all spikes emitted by neuron σ_1.

Theorem 1 can be easily extended by allowing more output neurons and then simulating a k-output CM, producing in this way sets of vectors of natural numbers.

Theorem 2 $PsSpik_{tot}EP_*^{nsyn}(gen, del_0) = PsRE$.

Note that the system Π constructed in the proof of Theorem 1 is general: neurons σ_r involved in SUB instructions contain both bounded and unbounded rules.

5 Unbounded SN P Systems

As mentioned in the Introduction, bounded SN P systems characterize the semilinear sets of numbers and this equivalence is proven in a constructive manner – see, e.g., [11]. Thus, the interesting case which remains to investigate is that of unbounded SN P systems.

In the following constructions we restrict the SN P systems syntactically to make checking a valid computation easier. Specifically, for an SN P system with unbounded neurons $\sigma_1, \ldots, \sigma_k$ (one of which is the output neuron) we assume as given non-negative integers $m_1, \ldots, m_k$, and for the rules in each σ_i we impose the following restriction: *If $m_i > 0$, then $a^{m_i} \notin L(E)$ for any regular expression E appearing in a rule of neuron σ_i.* This restriction guarantees that if neuron σ_i contains m_i spikes, then the neuron is not fireable. It follows that when the following conditions are met during a computation, the system has halted and the computation is valid:

1. All bounded neurons are open, but none is fireable.
2. Each σ_i contains *exactly* m_i spikes (hence none is fireable, too).

This way of defining a successful computation, based on a vector $(m_1, \ldots, m_k)$, is called *μ-halting*. In the notation of the generated families we add the subscript μ to N or to Ps, in order to indicate the use of μ-halting.

As defined earlier, a non-synchronized SN P system is one in which at each step, we select zero or more neurons to fire. Clearly, for 0-delay SN P systems, selecting *zero* or more neurons to fire at each step is equivalent to selecting *one* or more neurons to fire at each step. This is due to the fact that there are no delays. Hence, if we select no neuron to fire, the entire configuration of the system will remain the same.

5.1 0-Delay Unbounded SN P Systems and Partially Blind Counter Machines

In this section we give a characterization of 0-delay unbounded SN P systems in terms of partially blind counter machines.

A *partially blind k-output* CM (k-output PBCM) [7] is a k-output CM, where the counters cannot be tested for zero. The counters can be incremented by 1 or decremented by 1, but if there is an attempt to decrement a zero counter, the computation aborts (i.e., the computation becomes invalid). Note that, as usual, the output counters are nondecreasing. Again, by definition, a successful generation of a k-tuple requires that the machine enters an accepting state with *all* non-output counters zero.

We denote by $NPBCM$ the set of numbers generated by PBCMs and by $PsPBCM$ the family of sets of vectors of numbers generated by using k-output PBCMs.

It is known that k-output PBCMs can be simulated by Petri nets, and vice-versa [7]. Hence, PBCMs are not universal.

We shall refer to a 1-output PBCM simply as PBCM.

We show that unbounded 0-delay SN P systems with μ-halting are equivalent to PBCMs. This result generalizes to the case when there are k outputs. First, we describe a basic construction.

Basic Construction:

Let C be a partially blind counter. It is operated by a finite-state control. C can only store nonnegative integers. It can be incremented/decremented but when it is decremented and the resulting value become negative, the computation is aborted. Let i, j, d be given fixed nonnegative integers with $i \geq 0, j > 0, d > 0$. Initially, C is incremented (from zero) to some $m \geq 0$.

Depending on the finite-state control (which is non-deterministic), one of the following operations is taken at each step:

(1) C remains unchanged.

(2) C is incremented by 1.

(3) If the contents of C is of the form $i + kj$ (for some $k \geq 0$), then C is decremented by d.

Note that in (3) we may not know whether $i + jk$ is greater than or equal to d, or what k is (the multiplicity of j), since we cannot test for zero. But if we know that C is of the form $i + jk$, when we subtract d from it and it becomes negative, it aborts and the computation is invalid, so we are safe. Note that if C contains $i + jk$ and is greater than or equal to d, then C will contain the correct value after the decrement of d.

We can implement the above computation using only finite-state control in addition to C.

$i < j$ case

Define a modulo-j counter to be a counter that can count from 0 to $j - 1$. We can think of the modulo-j counter as an undirected circular graph with nodes 0, 1, ..., $j - 1$, where node s is connected to node $s + 1$ for $0 \leq s \leq j - 2$ and $j - 1$ is connected to 0. Node s represents count s. We increment the modulo-j counter by going through the nodes in a "clockwise" direction. So, e.g., if the current node is s and we want to increment by 1, we go to $s + 1$, provided $s \leq j - 2$; if $s = j - 1$, we go to node 0. Similarly, decrementing the modulo-j counter goes in the opposite direction, i.e., "counter-clockwise" – we go from s to $s - 1$; if it is 0, we go to $s - 1$.

The parameters of the machine are the triple (i, j, d) with $i \geq 0, j > 0, d > 0$. We associate with counter C a modulo-j counter, J, which is initially in node (count) 0. During the computation, we keep track of the current visited node of J. Whenever we increment/decrement C, we also increment/decrement J. Clearly, the requirement that the value of C has to be of the form $i + kj$ for some $k \geq 0$ in order to decrement by d translates to the J being in node i, which is easily checked.

$i \geq j$ case

Suppose $i = r + sj$ where $s > 0$ and $0 \leq r < j$.

Subcase 1:

If $d > i - j$, then we run $i < j$ case described above with parameters (r, j, d). When we want to perform a decrement-d, it is enough to check that the counter is of the form $r + kj$ for some $k \geq 0$. Note that if $r + kj < r + sj$, then the machine will abort, so the computation branch is not successful anyway.

Subcase 2:

If $d \leq i - j$, then we run $i < j$ case described above with parameters (r, j, d) with the following difference. When we want to perform a decrement-d, we make sure that the counter is of the form $r + kj$ for some $k \geq 0$. Then first subtract $i - j + 1$ from the counter (and if the machine aborts, nothing is lost), then add back $(i - j + 1 - d)$ to the counter. The intermediate step of subtracting $i - j + 1$ from the counter is accomplished by a suitably modified copy of the original machine.

We are now ready to prove the following result.

Lemma 5.1 *$N_\mu Spik_{tot} EP_*^{nsyn}(unb, del_0) \subseteq NPBCM$.*

Proof. We describe how a PBCM M simulates an unbounded 0-delay SN P system Π. Let B be the set of bounded neurons; assume that there are $g \geq 0$ such neurons. The bounded neurons can easily be simulated by M in its finite control. So we focus more on the simulation of the unbounded neurons. Let $\sigma_1, \ldots, \sigma_k$ be the unbounded neurons (one of which is the output neuron). M uses counters $C_1, \ldots, C_k$ to simulate the unbounded neurons. M also uses a nondecreasing counter C_0 to keep track of the spikes sent by the output neuron to the environment. Clearly, the operation of C_0 can easily be implemented by M. We introduce another counter, called ZERO (initially has value 0), whose purpose will become clear later.

Assume for the moment that each bounded neuron in B has only one rule, and each unbounded neuron σ_t ($1 \leq t \leq k$) has only one rule of the form $a^{i_t}(a^{j_t})^*/a^{d_t} \to a^{e_t}$. M incorporates in its finite control a modulo-j_t counter, J_t, associated with counter C_t (as described above). One step of Π is simulated in five steps by M as follows:

1. Non-deterministically choose a number $1 \leq p \leq g + k$.

2. Non-deterministically select a subset of size p of the neurons in $B \cup \{\sigma_1, \ldots, \sigma_k\}$.

3. Check if the chosen neurons are fireable. The neurons in B are easy to check, and the unbounded neurons can be checked as described above, using their associated J_t's (modulo-j_t counters). If at least one is not fireable, abort the computation by decrementing counter ZERO by 1.

4. Decrement the chosen unbounded counter by their d_t's and update their associated J_t's, as described above. The chosen bounded counters are also easily decremented by the amounts specified in their rules (in the finite control).

5. Increment the chosen bounded counters and unbounded counters by the total number of spikes sent to the corresponding neurons by their neighbors (again updating the associated J_t's of the chosen unbounded counters). Also, increment C_0 by the number of spikes the output neuron sends to the environment.

At some point, M non-deterministically guesses that Π has halted: It checks that all bounded neurons are open and none is fireable, and the unbounded neurons have their specified values of spikes. M can easily check the bounded neurons, since they are stored in the finite control. For the unbounded neurons, M decrements the corresponding counter by the specified number of spikes in that neuron. Clearly, $C_0 = x$ (for some number x) with all other counters zero if and only if the SN P system outputs x with all the neurons open and non-fireable (i.e., the system has halted) and the unbounded neurons containing their specified values.

It is straightforward to verify that the above construction generalizes to when the neurons have more than one rule. An unbounded neuron with m rules will have associated with it m modulo-j_{t_m} counters, one for each rule and during the computation, and these counters are operated in parallel to determine which rule can be fired. A bounded neuron with multiple rules is easily handled by the finite control. We then have to modify item 3 above to:

> Non-deterministically select a rule in each chosen neuron. Check if the chosen neurons with selected rules are fireable. The neurons in B are easy to check, and the unbounded neurons can be checked as described above, using the associated J_t's

(modulo-j_t counters) for the chosen rules. If at least one is not fireable, abort the computation by decrementing counter ZERO by 1.

We omit the details. □

Clearly, Lemma 5.1 generalizes to the following.

Corollary 1 $Ps_{\mu}Spik_{tot}EP_{*}^{nsyn}(unb, del_0) \subseteq PsPBCM.$

We now show the converse of Lemma 5.1.

Lemma 5.2 $NPBCM \subseteq N_{\mu}Spik_{tot}EP_{*}^{nsyn}(unb, del_0).$

Proof. To simulate a PBCM we need to be able to simulate an addition instruction, a subtraction instruction, and a halting instruction (but we do not need to test for zero). The addition instruction will add one to a counter. The halting instruction will cause the system to halt. The subtraction instruction will subtract one from a counter and cause the system to abort if the counter was zero.

Also, from our definition of a "valid computation", as a μ-halting computation, for the output of the SN P system to be valid, the system must halt and be in a valid configuration – we will see that in our construction, all neurons (bounded and unbounded) will contain zero spikes, except the output neuron which will contain *exactly* one spike. This means that any computation that leaves the non-output neurons with a positive spike count is invalid.

To create a 0-delay unbounded SN P system Π that will simulate a PBCM M we follow the simulation in the proof of Theorem 1. To simulate an instruction of the form $l_i = (\mathtt{ADD}(r), l_j, l_k)$, we create the same ADD module as in the proof of Theorem 1. It is important to note that all neurons in this module are bounded. Also, when the instruction is done executing, all neurons in the module contain zero spikes if the module executed in a valid manner. (There are some alternate computations which leave some spikes in some of these neurons. These computations are invalid and the system will not generate any output. This is explained more precisely in the proof of Theorem 1.)

To simulate an instruction of the form $l_i = (\mathtt{SUB}(r), l_j, l_k)$, we use the SUB module from the proof of Theorem 1 with a few small changes. In this module we remove the instruction $a^{3T-s} \rightarrow a^{2T+s}$ from neuron σ_r. Before, the neuron was a general neuron, but by removing all the finite rules we are only left with rules of the form $a^i(a^j)^*/a^d \rightarrow a^p; t$ and hence the neuron is unbounded. Note that all of the other neurons in the module are bounded. This rule change still allows neuron r (representing counter σ_r) to fire if it stored a^{3Tn} spikes for some n (representing a positive count in the counter) before instruction l_i is executed. In this case, the firing of neuron σ_r continues the computation. However, if neuron σ_r contained no spikes before the execution of instruction l_i (representing a count of zero), neuron σ_r will not fire causing the system to eventually halt (after other neurons forget). In this case, M tried to decrement a zero counter and so the system aborted. In the simulation, Π has halted in an invalid configuration since no neuron is fireable but neuron σ_r is not empty and still contains $3T - s$ spikes. (Also, no output was generated by the system).

The final change to the SUB module is that the instruction $a^{5T} \rightarrow a^{2T}$ is changed to $a^{6T} \rightarrow a^{2T}$ causing the next instruction (l_j or l_k) to be chosen non-deterministically if the subtraction simulation was successfully. Note that a correct execution of this module also leaves all the neurons (other than σ_r) with zero spikes.

To simulate an instruction of the form l_i : `HALT`, we again create the same HALT module given in the proof of Theorem 1. To generalize this simulation for a k-output PBCM we modify the HALT module slightly to trigger all of the k output neurons. This is done by creating extra synapses from neuron σ_{l_h} to the neurons $\sigma_2, \ldots, \sigma_k$. In this case, an accepting configuration leaves all non-output neurons with zero spikes and all output neurons with exactly one spike. □

Again, Lemma 5.2 generalizes to:

Corollary 2 $PsPBCM \subseteq Ps_{\mu}Spik_{tot}EP_{*}^{nsyn}(unb, del_0)$.

From Corollaries 1 and 2, we have the main result of this section:

Theorem 3 $Ps_{\mu}Spik_{tot}EP_{*}^{nsyn}(unb, del_0) = PsPBCM$.

It is known that PBCMs with only one output counter can only generate semilinear sets of numbers. Hence:

Corollary 3 0*-delay unbounded SN P systems with* μ*-halting can only generate semilinear sets of numbers.*

Theorem 3 is the best possible result we can obtain, since if we allow bounded rules and unbounded rules in the neurons, SN P systems become universal, as shown in Theorem 1, where the subtraction module (Figure 5) has a neuron with the following rules

$$a^{6T-s}(a^{3T})^{*}/a^{6T-s} \to a^{3T+s} \quad \text{and} \quad a^{3T-s} \to a^{2Ts}.$$

5.2 Closure Properties and Decision Problems

The following theorem is known:

Theorem 4

1. *(Union, intersection, complementation) The sets of k-tuples generated by k-output PBCMs are closed under union and intersection, but not under complementation.*

2. *(Membership) It is decidable to determine, given a k-output PBCM M and a k-tuple α (of integers), whether M generates α.*

3. *(Emptiness) It is decidable to determine, given a k-output PBCM, whether it generates an empty set of k-tuples.*

4. *(Infiniteness) It is decidable to determine, given a k-output PBCM, whether it generates an infinite set of k-tuples.*

5. *(Disjointness) It is decidable to determine, given two k-ouput PBCMs, whether they generate a common k-tuple.*

6. *(Containment, equivalence) It is undecidable to determine, given two k-output PBCMs, whether the set generated by one is contained in the set generated by the other (or whether they generate the same set).*

7. *(Reachability) It is decidable to determine, given a PBCM with k output counters and m auxiliary counters (thus a total of $k+m$ counters) and configurations $\alpha = (i_1, \ldots, i_k, j_1, \ldots, j_m)$ and $\beta = (i'_1, \ldots, i'_k, j'_1, \ldots, j'_m)$ (the first k components correspond to the output), whether α can reach β.*

Then, from Theorem 3 and Theorem 4 parts 1–5, we have:

Corollary 4 *Theorem 4 parts 1–6 also hold for 0-delay unbounded k-output SN P systems with μ-halting.*

In the construction of the PBCM from SN P system in the proof of Lemma 5.1, we only provided counters for the unbounded neurons and a counter to keep track of the number of spikes that the output neuron sends to the environment. The bounded neurons are simulated in the finite control of the PBCM. We could have also allocated a partially blind counter for each bounded neuron (for manipulating a bounded number) and use the finite control to make sure that these added counters never become negative. Then the PBCM will have $m+1$ counters, where m is the total number of neurons (bounded and unbounded) in the SN P system and σ_1 corresponds to the output. In the case of k-output SN P system, the PBCM will have $m+k$ counters. Then from Theorem 4 part 7, we have:

Corollary 5 *It is decidable to determine, given a 0-delay unbounded k-output SN P system with m neurons, and configurations $\alpha = (i_1, \ldots, i_k, j_1, \ldots, j_m)$ and $\beta = (i'_1, \ldots, i'_k, j'_1, \ldots, j'_m)$ (the first k components correspond to the output), whether α can reach β.*

Note that for the above corollary, we do not need to define what is a halting configuration for the SN P system, as we are only interested in reachability and not the set of tuples the system generates.

6 Final Remarks

We have considered spiking neural P systems with a non-synchronized use of rules: in any step, a neuron can apply or not its rules which are enabled by the number of spikes it contains (further spikes can come, thus changing the rules enabled in the next step). Asynchronous spiking neural P systems have been proved to be universal when using extended rules (several spikes can be produced by a rule) and neurons containing both bounded and unbounded rules.

Moreover, we have given a characterization of a class of spiking neural P systems – the unbounded ones, with μ-halting – in terms of partially blind counter machines.

SN P systems operating in sequential mode were studied earlier in [9]. In this mode, at every step of the computation, if there is at least one neuron with at least one rule that is fireable we only allow one such neuron and one such rule (both chosen non-deterministically) to be fired. It was shown in [9] that certain classes of sequential SN P systems are equivalent to partially blind counter machines, while others are universal. Thus, in some sense, non-synchronized and sequential modes of computation are equivalent.

Many issues remain to be investigated for the non-synchronized SN P systems, starting with the main *open problem* whether or not SN P systems with standard rules (rules can only produce one spike) are Turing complete also in this case. Then, most of the questions considered for synchronized systems are relevant also for the non-synchronized case. We just list some of them: associating strings to computations (if $i \geq 1$ spikes exit the output neuron, then the symbol b_i is generated); finding universal SN P systems, if possible, with a small number of neurons; considering restricted classes of systems (e.g., with a bounded number of spikes present at a time in any neuron). In the bibliography below we indicate papers dealing with each of these issues for the case of synchronized SN P systems.

In the proof of the equivalence of asynchronous unbounded SN P systems with partially blind counter machines e have used the μ-halting way of defining successful computations; the resulting decidability consequences are also based on this condition. Can μ-halting be replaced with the usual halting (hence ignoring the contents of neurons in the halting configuration) without losing these results?

Then, a natural question is to investigate the class of systems for which "the time does not matter", for instance, such that $N_{gen}^{syn}(\Pi) = N_{gen}^{nsyn}(\Pi)$ (like in the second example from Section 3). Suggestions in this respect can be found, e.g., in [1], [2].

Acknowledgements

The work of the authors was supported as follows. O. Egecioglu, O.H. Ibarra and S. Woodworth were supported in part by NSF Grants CCF-0430945 and CCF-0524136. M. Ionescu was supported by the fellowship "Formación de Profesorado Universitario" from the Spanish Ministry of Education, Culture and Sport. Gh. Păun was partially supported by the project BioMAT 2-CEx06-11-97/19.09.06. This research was in part carried out during a visit of M. Ionescu and Gh. Păun at the Microsoft Research-University of Trento Center for Computational and Systems Biology, Trento, Italy.

References

[1] M. Cavaliere, V. Deufemia: Further results on time-free P systems. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 69–90.

[2] M. Cavaliere, D. Sburlan: Time-independent P systems. In *Membrane Computing. International Workshop WMC5, Milano, Italy, 2004*, LNCS 3365, Springer, 2005, 239–258.

[3] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [6], Vol. I, 169–194.

[4] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [6], Vol. I, 241–265.

[5] W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.

[6] M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.

[7] S. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7, 3 (1978), 311–324.

[8] O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. *Proc. 7th Workshop on Membrane Computing, Leiden, July 2006*, LNCS 4361, Springer, Berlin, 2006, 424–442.

[9] O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. *Proc. 5th International Conference on Unconventional Computation*, LNCS 4135, Springer, Berlin, 2006, 113–129.

[10] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [6], Vol. II, 105–136, and *Theoretical Computer Sci.*, to appear.

[11] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.

[12] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. of Unconventional Computing*, to appear.

[13] W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.

[14] W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.

[15] M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.

[16] M. Minsky: Recursive unsolvability of Post's problem of tag and other topics in theory of Turing machines. *Annals of Mathematics*, 74, 3 (1961), 437–455.

[17] A. Păun, Gh. Păun: Small universal spiking neural P systems. In [6], Vol. II, 213–234, and *BioSystems*, in press.

[18] Gh. Păun: *Membrane Computing – An Introduction.* Springer, Berlin, 2002.

[19] Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. In *Proceedings of Developments in Language Theory Conference, DLT 2006*, Santa Barbara, CA, June 2006, LNCS 4036, Springer, Berlin, 2006, 20–35.

[20] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.

[21] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted 2005.

[22] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.

[23] The P Systems Web Page: `http://psystems.disco.unimib.it`.