# Asynchronous Spiking Neural P Systems: Decidability and Undecidability

Matteo Cavaliere[1], Omer Egecioglu[2], Oscar H. Ibarra[2], Mihai Ionescu[3], Gheorghe Păun[4], and Sara Woodworth[2]

[1] Microsoft Research-University of Trento CoSBi, Italy
cavaliere@cosbi.eu
[2] Dept. of Computer Science, University of California, Santa Barbara, USA
{omer, ibarra, swood}@cs.ucsb.edu
[3] Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, Spain
armandmihai.ionescu@urv.cat
[4] Institute of Mathematics of the Romanian Academy, Bucharest, Romania
george.paun@imar.ro, gpaun@us.es

**Abstract.** In search for "realistic" bio-inspired computing models, we consider asynchronous spiking neural P systems, in the hope to get a class of computing devices with decidable properties. However, although the non-synchronization is known in general to decrease the computing power, in the case of using extended rules (several spikes can be produced by a rule) we obtain again the equivalence with Turing machines (interpreted as generators of sets of vectors of numbers). The 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 spiking neural P systems, with a specific way of halting, using extended rules and where each neuron is either bounded or unbounded, are equivalent to partially blind counter machines and, therefore, have many decidable properties.

## 1 Spiking Neural P Systems – An Informal Presentation

In the present paper we continue the investigation of spiking neural P systems (SN P systems, in short). A survey of results and the biological motivations for these systems can be found in [5] and [2]. In the meantime, two main research directions were particularly active in this area of membrane computing: looking for classes of systems with tractable (for instance, decidable) properties, and looking for the possibility of using SN P systems for efficiently solving computationally hard problems. Along the second research line are the investigations related to the possibility of simulating an SN P system by a Turing machine with a polynomial slowdown (preliminary results can be found in [3]) and those trying to improve the efficiency of SN P systems, e.g., by enhancing the parallelism of the system (see, for instance, [7]).

In this paper we report several recent results concerning the first topic mentioned above – specifically, removing the synchronization (common in many membrane computing models), calling them asynchronous SN P systems. These systems were introduced in [6] with the aim of incorporating, into membrane computing, specific ideas from spiking neurons, a field that is being heavily investigated in neural computing (see, e.g., [8]).

An SN P system consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*) 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 [10,11] for an introduction to membrane computing and to [12] for the up-to-date information about this research area). The objects evolve by means of *spiking rules* placed in the nodes and enabled when the (number of) spikes present in the nodes fulfill specified regular expressions. When a spiking rule is executed in a neuron, spikes are produced and sent to all neurons connected by an outgoing synapse from the neuron where the rule was applied.

Two main types of results were obtained for synchronous (i.e, with obligatory use of the rules) systems using standard rules (producing one spike): computational completeness ([6]) 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. Improvements in the form of the regular expressions and normal forms can be found in [4].

In the proofs 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 (even if a neuron has a rule enabled in a given time unit, this rule is not necessarily used).

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 SN P systems where extended rules (producing several spikes) are used: such systems are still equivalent with Turing machines.

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: in particular, we identify a class of asynchronous SN P systems equivalent to partially blind counter machines (i.e., not computationally complete) and for which the configuration reachability, membership (in terms of generated vectors), emptiness, infiniteness, and disjointness problems can be decided.

## 2 SN P Systems – Formal Definitions

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 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. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
4. $out \in \{1, 2, \ldots, m\}$ 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*).

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).

If the neuron $\sigma_i$ 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 $t, t+1, t+2, \ldots, t+d-1$ 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 $\sigma_i$ are replicated and they go to all neurons $\sigma_j$ such that $(i, j) \in syn$ (each $\sigma_j$ 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 an asynchronous SN P system in each time unit *any neuron is free to use a rule or not* (a global clock, marking the time for all neurons, is considered). Hence, in each time unit, each neuron can use either zero or one rule. Even if enabled, a rule is not necessarily applied, the neuron can remain still not used 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 (i.e., 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 $n_1, \ldots, n_m$ 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). The *result* of a computation is defined here as the total number of spikes sent into the environment by the output neuron.

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

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 $\Pi$, 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 $\alpha$ (*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. Because there is no confusion, in this paper, asynchronous SN P systems are often simply called SN P systems.

In the next section we present a module of the construction from the proof of the universality theorem, and this can illustrate and clarify the above definitions. On that occasion we also use the standard way to pictorially represent a configuration of an SN P system. 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. Examples of SN P systems working in an asynchronous way can be found in the technical report [1].

## 3 Computational Completeness of General SN P Systems

We now show that the power of general neurons (with extended rules) can compensate the loss of power entailed by removing the synchronization.

**Theorem 1.** *$NSpik_{tot}EP^{nsyn}_{*}(gen, del_0) = NRE$.*

*Proof.* (sketch) We only prove that $NRE \subseteq Spik_{tot}EP^{nsyn}_{*}(gen, del_0)$ and to this aim, we use the characterization of $NRE$ (i.e., the family of sets of numbers computed by Turing machines) by means of counter machines (abbreviated CM), [9]. Let $M = (m, H, l_0, l_h, I)$ be a counter machine with $m$ counters, such that the result of a computation is the number stored in counter 1 and this counter is never decremented during the computation. We construct a spiking neural P system $\Pi$ 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 by

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

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

With each label $l$ of an instruction in $M$ we also associate a neuron $\sigma_l$. Initially, all these neurons are empty, with the exception of the neuron $\sigma_{l_0}$ associated with the start label of $M$, which contains $3T$ spikes. This means that this neuron is "activated". During the computation, the neuron $\sigma_l$ 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 $\sigma_{l_i}$ 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 $\sigma_{l_h}$, associated with the halting label of $M$, the computation taking place in the counter machine $M$ is completely simulated in $\Pi$; we will then send to the environment a number of spikes equal to the number stored in the first counter of $M$. Neuron $\sigma_1$ is the output neuron of the system. Further neurons will be associated with the counters and the labels of $M$; all of them being initially empty.

The construction consists of modules simulating the ADD and SUB instructions of $M$, as well as a final module. We present here, in Figure 1, only the SUB module.

Let us start with $3T$ spikes in neuron $\sigma_{l_i}$ and no spike in other neurons, except neurons associated with counters; assume that neuron $\sigma_r$ holds a number of spikes of the form $3Tn$, $n \geq 0$. Assume also that this is the $s$th 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 $s$).

Some time, neuron $\sigma_{l_i}$ spikes and sends $3T - s$ spikes both to $\sigma_r$ and to $\sigma_{i,0}$. These spikes can be forgotten in this latter neuron, because $2T < 3T - s < 4T$. At a certain time, also neuron $\sigma_r$ will fire, and will send $2T + s$ or $3T + s$ 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).

If neuron $\sigma_{i,0}$ does not forget the spikes received from $\sigma_{l_i}$ (this is possible, because of the non-synchronized mode of using the rules), then eventually neuron $\sigma_r$ 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 $\sigma_{l_i}$. In either case, $\sigma_{i,0}$ accumulates more than $4T$ spikes, hence it cannot forget them.

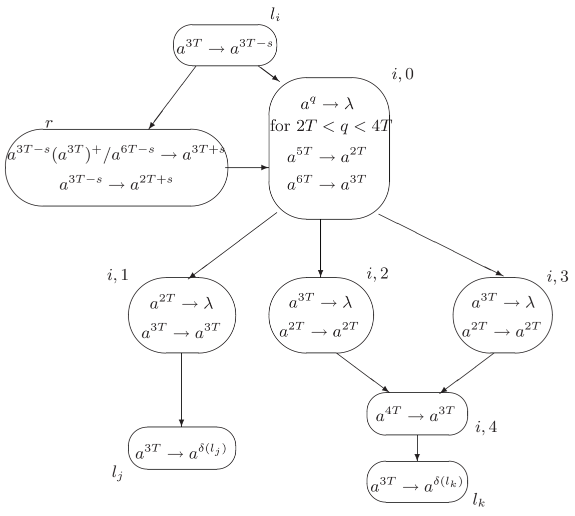

**Fig. 1.** Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

Depending on the number of spikes accumulated, either $6T$ or $5T$, neuron $\sigma_{i,0}$ eventually spikes, sending $3T$ or $2T$ spikes, respectively, to $\sigma_{i,1}$, $\sigma_{i,2}$, and $\sigma_{i,3}$. The only possible continuation of neuron $\sigma_{i,1}$ is to activate neuron $\sigma_{l_j}$ (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 $\sigma_{l_k}$ (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 1. What remains to examine is the possible interferences between modules, for instance, between neurons $\sigma_r$ 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 $\sigma_r$ for which there exist $t_r$ SUB instructions can send spikes to all neurons $\sigma_{i,0}$ as in Figure 1. However, only one of these target neurons also receives spikes from a neuron $\sigma_{l_i}$, the one identifying the instruction which we want to simulate. By a careful analysis of the number of spikes a neuron can receive, the reader can check that the only computations in $\Pi$ which can reach the neuron $\sigma_{l_h}$ associated with the halting instruction of $M$ are the computations which correctly simulate the instructions of $M$ and correspond to halting computations in $M$.

In a similar way we construct the ADD and FIN modules of an SN P system $\Pi$ (the reader can find the detailed construction in the technical report [1]). Hence $N^{nsyn}_{gen}(\Pi) = N(M)$.

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

## 4 Characterization of Unbounded SN P Systems by Partially Blind Counter Machines

For the constructions in this section, 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 $\sigma_i$ 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 $\sigma_i$.* This restriction guarantees that if neuron $\sigma_i$ 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, and (2) each $\sigma_i$ 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 *$\mu$-halting.* In the notation of the generated families we add the subscript $\mu$ to N or to Ps, in order to indicate the use of $\mu$-halting.

A *partially blind $k$-output* CM ($k$-output PBCM) 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. Hence, PBCMs are not universal.

**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 computation using only $C$ and the finite-state control as follows:

**Case:** $i < j$. 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 $0, 1, \ldots, j-1$, 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.
**Case:** $i \geq j$. Suppose $i = r + sj$ where $s > 0$ and $0 \leq r < j$. There are two subcases: $d > i-j$ and $d \leq i-j$. We can show (we omit the "tricky" consruction here) that both subcases can also be implemented.

Using the above construction we get the following, rather surprising result.

**Theorem 2.** *$N_\mu Spik_{tot} EP_*^{nsyn}(unb, del_0) = NPBCM$.*

*Proof.* (sketch) We describe how a PBCM $M$ simulates an unbounded 0-delay SN P system $\Pi$. 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 $\sigma_t$ $(1 \leq t \leq k)$ has only one rule of the form $a^{i_t}(a^{j_t})^*/a^{d_t} \rightarrow a^{e_t}$. $M$ incorporates in its finite control a modulo-$j_t$ counter, $J_t$, associated with counter $C_t$, implemented by using the above basic construction. One step of $\Pi$ 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 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. 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 $\Pi$ 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 described 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:
3'. 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 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.

The proof of the converse, which we omit (for lack of space), is an intricate modification of the simulation in the proof of Theorem 1. Because each neuron can only have either bounded rules or unbounded rules (but *not* both), the simulation by PBCM is possible.

Theorem 2 can be generalized to the case with multiple outputs:

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

This 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 (Theorem 1).

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

**Corollary 1.** *Unbounded* 0*-delay SN P systems with* $\mu$*-halting can only generate semilinear sets of numbers.*

The results in the following corollary can be obtained using Theorem 3 and the fact that they hold for $k$-output PBCMs.

**Corollary 2.** *1. The sets of* $k$*-tuples generated by* $k$*-output unbounded 0-delay SN P systems with* $\mu$*-halting is closed under union and intersection, but not under complementation.*

2. *The membership, emptiness, infiniteness, disjointness, and reachability problems are decidable for $k$-output unbounded 0-delay SN P systems with $\mu$-halting (for reachability, we do not need to define what is a halting configuration as we are not interested in tuples the system generates); but containment and equivalence are undecidable.*

## 5 Final Remarks

Many issues remain to be investigated for asynchronous SN P systems. We only mention two of them: whether or not asynchronous SN P systems with standard rules (i.e., that can only produce one spike) are Turing complete and whether or not the decidability results proved in Section 4 can be proved by using the usual halting (i.e., by removing the $\mu$-halting).

**Acknowledgments.** The work of O. Egecioglu, O.H. Ibarra, and S. Woodworth was supported in part by NSF Grants CCF-0430945 and CCF-0524136. The work of M. Ionescu was supported by “Formación de Profesorado Universitario” fellowship from MEC, Spain.

## References

1. Cavaliere, M., Egecioglu, O., Ibarra, O.H., Woodworth, S., Ionescu, M., Păun, Gh.: Asynchronous spiking neural P systems. Tech. Report 9/2007 Microsoft Research - University of Trento, Centre for Computational and Systems Biology, `http://www.cosbi.eu`
2. Chen, H., Ionescu, M., Ishdorj, T.-O., Păun, A., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with extended rules: Universality and languages, Natural Computing (special issue devoted to DNA12 Conf.) (to appear)
3. Gutiérrez-Naranjo, M.A., et al.: Proceedings of Fifth Brainstorming Week on Membrane Computing, Fenix Editora, Sevilla (in press, 2007)
4. Ibarra, O.H., Păun, A., Păun, Gh., Rodríguez-Patón, A., Sosik, P., Woodworth, S.: Normal forms for spiking neural P systems. Theoretical Computer Sci. (to appear)
5. Ionescu, M., Păun, A., Păun, Gh., Pérez-Jiménez, M.J.: Computing with spiking neural P systems: Traces and small universal systems. In: Mao, C., Yokomori, T., Zhang, B.-T. (eds.) DNA Computing. LNCS, vol. 4287, pp. 32–42. Springer, Heidelberg (2006)
6. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2-3), 279–308 (2006)
7. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems with exhaustive use of rules. Intern. J. of Unconventional Computing (to appear)
8. Maass, W., Bishop, C. (eds.): Pulsed Neural Networks. MIT Press, Cambridge (1999)
9. Minsky, M.: Computation – Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
10. Păun, Gh.: Membrane Computing – An Introduction. Springer, Berlin (2002)
11. Păun, Gh.: Introduction to membrane computing. In: Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing, Springer, Heidelberg (2006)
12. The P Systems Web Page, `http://psystems.disco.unimib.it`