International Journal of Foundations of Computer Science
Vol. 17, No. 4 (2006) 975–1002

World Scientific
www.worldscientific.com

SPIKE TRAINS IN SPIKING NEURAL P SYSTEMS

GHEORGHE PĂUN
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: george.paun@imar.ro, gpaun@us.es

MARIO J. PÉREZ-JIMÉNEZ
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: marper@us.es

GRZEGORZ ROZENBERG
Leiden Institute of Advanced Computer Science, LIACS
University of Leiden, Niels Bohr Weg 1
2333 CA Leiden, The Netherlands, and
Department of Computer Science, University of Colorado at Boulder
Boulder, CO 80309-0430, USA
E-mail: rozenber@liacs.nl

Received 5 December 2005
Accepted 23 January 2006
Communicated by O. H. Ibarra

We continue here the study of the recently introduced spiking neural P systems, which mimic the way that neurons communicate with each other by means of short electrical impulses, identical in shape (voltage), but emitted at precise moments of time. The sequence of moments when a neuron emits a spike is called the spike train (of this neuron); by designating one neuron as the output neuron of a spiking neural P system Π, one obtains a spike train of Π. Given a specific way of assigning sets of numbers to spike trains of Π, we obtain sets of numbers computed by Π. In this way, spiking neural P systems become number computing devices. We consider a number of ways to assign (code) sets of numbers to (by) spike trains, and prove then computational completeness: the computed sets of numbers are exactly Turing computable sets. When the number of spikes present in the system is bounded, a characterization of semilinear sets of numbers is obtained. A number of research problems is also formulated.

Keywords: Spiking neuron, membrane computing, P System, register machine, semilinear set.

2000 Mathematics Subject Classification: 68Q10, 68Q42, 68Q45

1. Introduction

The idea of spiking neurons, currently an active research topic in neural computing (see, e.g., [4], [6], [7]), was recently incorporated in membrane computing (see [5]) – the resulting formal systems are called *spiking neural P systems*, abbreviated as SN P systems.

The structure of an SN P system has a form of a directed graph with nodes representing neurons, and edges representing synapses. A neuron (node) sends signals (spikes) along its outgoing synapses (edges). We use a reserved symbol/letter a to represent a spike. Each neuron has its own rules for either sending spikes (firing rules) or for internally consuming spikes (forgetting rules). In the initial configuration a neuron σ stores the initial number of spikes, and at any time moment the currently stored number of spikes in σ (*current contents* of σ) is determined by the initial contents of σ and the history of functioning of σ (the spikes it received from other neurons, the spikes it sent out, and the spikes it internally consumed/forgot). For each firing rule r there is a set of numbers, S_r enabling this rule: i.e., if the current contents of σ is in S_r, then r is enabled. When r is enabled, it can *fire*, i.e., initialize the process of transmitting a spike (*spiking*) by σ along all synapses outgoing from σ – as explained below, the spiking can take place either at the same moment of time as the firing of r or at a later moment. Each firing is coupled with a consumption of a specific number c_r of spikes, which is fixed for each firing rule, where $c_r \leq z$ for each $z \in S_r$. Thus, if the current contents of σ is n, and $n \in S_r$, then rule r fires and consumes c_r spikes, so that $n - c_r$ spikes remain in σ.

Each firing rule r in σ has its delay number d_r (a nonnegative integer) which determines the delay between the firing of r, and the spiking by σ.

If $d_r = 0$, then σ spikes (using r) at the same time moment t that r fires. However, if $d_r > 0$, then σ will spike (using r) d_r moments after r fires. Moreover, σ becomes blocked through the time interval $t, t+1, \ldots, t+(d_r-1)$, becoming unblocked again at time moment $t + d_r$ (when it spikes). When σ is blocked, no input spike enters σ, i.e., if a neuron σ' sends a spike to σ along the synapse from σ' to σ, then this spike is "wasted" as it does not enter σ.

There is no delay along synapses: if σ spikes at a time moment t along a synapse to a neuron σ', then σ' (if not blocked) will receive this spike at the same time moment t.

The whole system is timed by a central global clock which determines the same passage of time moments for all neurons.

The firing rules are given in the form $E/a^c \rightarrow a; d$, where E is a regular expression over $\{a\}$. If a neuron σ contains such a rule, then E specifies the enabling set S_r, c specifies the consumption cost, $c = c_r$, and d specifies the time delay, $d = d_r$.

As mentioned already, besides firing rules, a neuron σ can also have forgetting rules which are of the form $a^s \rightarrow \lambda$. Such a rule is enabled at a moment of time t, if the current contents of σ at t equals s. When an enabled rule is applied, then its effect is just an "internal consumption" of s spikes, refereed to as *forgetting*.

To complete the description of the functioning of an SN P system, we need to describe the order of activities for any neuron σ at a given time moment t, which is as follows. First of all, the current contents of σ at t, denoted $con_\sigma(t)$, is the number of spikes determined by the whole "history" of σ up to and including the time moment $t-1$ (if $t=0$, then $con_\sigma(t)$ is its initial contents).

If σ is blocked at t, then nothing happens to σ at t, and so $con_\sigma(t+1) = con_\sigma(t)$.

If σ is not blocked and no rule of σ is enabled by $con_\sigma(t)$, then neither firing nor forgetting will take place. Consequently, $con_\sigma(t+1)$ will be equal to $con_\sigma(t)$ increased by the number $get_\sigma(t)$, of spikes that σ receives during t.

If σ is not blocked and at least one rule of σ is enabled, then (nondeterministically) exactly one rule r of σ is chosen to act. If r is a firing rule $E/a^c \to a; d$ with $d=0$, then $con_\sigma(t+1) = con_\sigma(t) - c + get_\sigma(t)$, and σ spikes at t (along all outgoing synapses); if $d>0$, then $con_\sigma(t+1) = con_\sigma(t) - c$, and σ spikes at $t+d$ (along all outgoing synapses).

If r is a forgetting rule $a^s \to \lambda$, then $con_\sigma(t+1) = con_\sigma(t) - s + get_\sigma(t)$.

The reader may have noticed already the similarities between the above described structure and the functioning of a network of neurons connected by synapses, and membrane systems (see, e.g., [9]), or, more specifically, tissue and neural membrane systems. Neurons become just elementary membranes containing evolution rules (consisting of firing and forgetting rules) and multisets of molecules, where there is only one molecule, a, in the whole system. The firing rules are used to send molecules out of elementary membranes into the neighboring elementary membranes, and forgetting rules are purely internal evolution rules. However, there are also basic differences between tissue membrane systems and our networks of spiking neurons. For instance, we use here only one type of objects (the spike, a), rules of two particular forms, applied in a sequential manner in each neuron. In contrast to this, in tissue membrane systems the rules are used in parallel (maximal parallelism) and the contents of regions include many kinds of molecules/symbols. There also are basic differences between neural membrane systems (see, e.g., Chapter 6 of [9]) and our networks of spiking neurons. For instance, in neural membrane systems there are states associated with neurons, and, also there, many kinds of molecules/symbols may be used.

Thus, SN P systems should be seen as extending the framework of membrane computing so as to account for some computational principles present in spiking neural networks.

SN P systems can be used in many ways for the purpose of computing. Because of the one letter "spike" alphabet $\{a\}$, it is natural to use them as generators of sets of numbers. This has been done in [5] already, and for this purpose one also designates one of the neurons as the output neuron by providing it with an "outgoing edge": each time that this neuron spikes, a spike is sent to the environment of the system which is a natural place to collect the output.

In the current paper we continue this line of research by investigating in a

systematic fashion a number of mechanisms/methodologies that allow to use SN P systems for computing sets of numbers. The basic notion behind these mechanisms as well as behind the mechanism considered in [5] is the notion of *spike train*, which comes from spiking neural networks. Intuitively speaking, a spike train is the sequence of spikes emitted by the output neuron, where for each spike the time unit when it is emitted is indicated. Then, as the set of numbers associated with a spike train we can consider the times $t_1, t_2, \ldots$ when the spikes are emitted by the output neuron, or the distances between spikes, $t_2 - t_1, t_3 - t_2, \ldots$, with two possibilities: considering all spikes of a spike train or only the first k, for a prescribed k. There also are other possibilities, which we will discuss in Section 4. (It is worth noting that the idea of considering the number of steps elapsed between two events – here, sending out of the system a spike – can be considered also for other types of P systems and, actually, it has been previously investigated, e.g., in [3].)

For all these ways of defining sets of natural numbers computed by an SN P system, we prove here two types of results, thus extending the results from [5]: (i) SN P systems without a bound on the number of spikes present in their neurons characterize the computing power of Turing machines, but, (ii) if such a bound is imposed, then the computing power decreases drastically – we obtain a characterization of semilinear sets of numbers.

In the next section we introduce some notions and notations from computation theory that we need in the sequel. Then, in Section 3 we recall from [5] the definition of spiking neural P systems and fix the notation we will use in this paper. In Section 4 we introduce various sets of numbers that we associate with an SN P system, and in Section 5 we illustrate the basic notions using three examples. The equivalence of SN P systems without a bound on the contents of neurons with Turing machines used for computing sets of numbers is proved in Section 6, while in Section 7 we give a characterization of semilinear sets of numbers in terms of SN P systems with a bound on the contents of neurons. The paper ends with a discussion of open problems and research topics in Section 8.

2. Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as with basic membrane computing. Thus we recall here only some notions that we will use in order to establish the notation for this paper. For a comprehensive reference to formal language theory and membrane computing, we refer the reader to [11] and [9], respectively (and to [12] for the most updated information about membrane computing).

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

We also consider infinite sequences of symbols over a given alphabet V; their

set is denoted by V^ω. When $V = \{a\}$, we write a^ω instead of $\{a\}^\omega$. Throughout the paper, we use "string" to refer to finite strings and "infinite strings/sequences" to refer to the elements of V^ω.

The family of recursively enumerable languages (of finite strings) is denoted by RE and the family of Turing computable sets of natural numbers is denoted by NRE (it is the family of length sets of languages in RE). Similarly, the family of semilinear sets of natural numbers is denoted by $NREG$ (this is the family of the length sets of regular languages).

A *regular expression* over an alphabet V is defined by: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. Clearly, we assume that the parentheses are not in V; as a matter of fact, we will often omit "unnecessary parentheses". Also, $E_1^+ \cup \lambda$ can be written as E_1^*. With each expression E we associate its language $L(E)$ as follows: (i) $L(\lambda) = \{\lambda\}, L(a) = \{a\}$, for $a \in V$, (ii) $L((E_1)(E_2)) = L(E_1)L(E_2)$, $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for all regular expressions E_1, E_2.

The operations used here are the standard union, concatenation, and Kleene $+$. We also need below the operation of *right derivative* of a language $L \subseteq V^*$ with respect to a string $x \in V^*$, which is defined by

$$L/x = \{y \in V^* \mid yx \in L\}.$$

The universality proof from [5] uses the characterization of NRE by register machines (see [8]). We will invoke this proof below, but we do not give here the definition of a register machine. Instead, we recall that such a machine M has a finite set of labeled instructions, which can be ADD instructions (increasing the value of registers by one), SUB instructions (decreasing the value of nonempty registers by one), or the HALT instruction (for ending the successful computations). A number n is generated by M if, starting with all registers empty and executing the instruction with label l_0, the machine reaches the HALT instruction with all registers being empty again, with the exception of register 1, which holds n.

Also, we do not recall any general concept from the membrane computing area, although what follows is related to the so-called neural-like P systems – see details in Chapter 6 of [9].

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

3. Spiking Neural P Systems

We recall now from [5] the computing device which we investigate here, without mentioning again the neural motivation, but recalling informally the basic ideas, those which make an essential difference from usual membrane systems: we work with only one object, denoting a spike, a "quanta" of energy (an impulse of the same intensity/shape) sent by a neuron along its axon to all neurons with which it is linked through a synapse; this means that we have these neurons (single membranes) placed in the nodes of an arbitrary graph, with one of the neurons called the output one; depending on their contents (number of spikes accumulated), the neurons either fire – and immediately or at a subsequent step spike, sending a spike to the neighboring neurons – or forget the spikes they have; as a result, in the environment we get a sequence of spikes, leaving the system (its output neuron) at specific moments of times. This is called *spike train*, and this is the support of information the computation of the system provides.

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

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

where:

(1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
(2) $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:

a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
b) R_i is a finite set of *rules* of the following two forms:

(1) $E/a^c \to a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
(2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;

(3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
(4) $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (i.e., σ_{i_0} is the output neuron).

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a spike after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in step t and

$d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology, [1]), so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then the spike is lost). In step $t+d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used in step $t+d+1$).

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

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

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

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$, of spikes present in each neuron. During a computation, the "state" of the system is described by both by the number of spikes present in each neuron, and by the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again. (Thus, the state of a neuron i is described by a couple of numbers (n_i, t_i), where n_i is the number of spikes present in the neuron, and t_i is the number of steps until having the neuron open; if $t_i = 0$, as it is the case initially, then the neuron is open, it can already use rules and receive spikes from other neurons.)

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

In the spirit of spiking neurons, as the result of a computation, in [5] one takes the number of steps between two spikes sent out by the output neuron, and, for simplicity, one considers as successful only computations whose spike trains contain exactly two spikes. We will generalize this in the next section, also giving more

precise definitions and notations for the result of a computation.

4. Considering Spike Trains

Let us place our discussion in the general framework of SN P systems which can spike a non-bounded number of times along a computation (in particular, the computation can be non-halting), and let us take into consideration all spikes emitted by the output neuron during the computation.

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

We will use $COM(\Pi)$ to denote the set of all computations of Π, and $HCOM(\Pi)$ to denote the set of all halting computations of Π.

Of course, the set $ST(\Pi)$ itself can be considered as the result of the evolution of Π, thus placing the investigation in the framework of infinite sequence processing, considering the spike trains as sequences $t_1, t_2, \ldots$ of natural numbers or as binary sequences, with 1 written in moments $t_1, t_2, \ldots$, and 0 in all intermediate moments. This latter possibility is investigated in [10]. Here we adhere to a more classic framework, considering SN P systems as computing devices, which compute sets of natural numbers (this was also done in [5]).

One can associate a set of numbers with $ST(\Pi)$ in several ways. Perhaps the simplest is to take the set of all numbers $t_1, t_2, \ldots$ from all spike trains. Formally, we get $T(\gamma) = \{t_1, t_2, \ldots \mid st(\gamma) = \langle t_1, t_2, \ldots \rangle\}$ and $T(\Pi) = \bigcup_\gamma T(\gamma)$, where γ ranges over all computations with respect to Π. Then, $T^h(\Pi)$ is the subset of $T(\Pi)$ resulting from all sets $T(\gamma)$ such that γ is a halting computation. We will not investigate this case in what follows (interesting connections with so-called time constructible functions [2] can probably be made).

Then, like in [5], we can consider the intervals between consecutive spikes as numbers computed by a computation, with several alternatives:

- Taking into account only the first two spikes:

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

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

$$N_k(\Pi) = \{n \mid n = t_i - t_{i-1}, \text{ for } 2 \leq i \leq k, \gamma \in COM(\Pi),$$

$$st(\gamma) = \langle t_1, t_2, \ldots \rangle, \text{and } \gamma \text{ has at least } k \text{ spikes}\}.$$

Clearly, $N_2(\Pi)$ is a particular case of $N_k(\Pi)$, but we have formulated it separately because it was considered on its own in [5].

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

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

- Taking into account all intervals of all computations:

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

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

Two subsets of (some of) these sets are also of interest:

- Taking only *halting* computations; this makes sense only for $N_k(\Pi), k \geq 2$, and for $N_{all}(\Pi)$ – the respective subsets are denoted by $N_k^h(\Pi)$ and $N_{all}^h(\Pi)$, respectively.
- Considering *alternately* the intervals:

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

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

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

5. Three Examples

Before investigating the sets defined above, we will consider some examples.

The first example is rather simple – it is the system Π_1 presented in a pictorial way in Figure 1.

As already introduced in [5], we will represent SN P systems as graphs, whose nodes represent neurons, and edges represent synapses. Then in each node we specify all rules as well as the current number of spikes in the neuron represented by this node. We also attach an outgoing arrow to the output neuron. Moreover, we will

use the following simplification in specifying the firing rules of a neuron. If a firing rule is of the form $E/a^c \to a; d$ where $L(E) = \{a^c\}$, then we write this rule in the form $a^c \to a; d$.

With this convention, our first example is given in Figure 1.

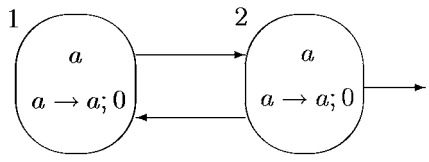

Fig. 1. A simple SN P system with an infinite spike train.

Thus, there are two identical neurons, each one firing and spiking at each moment of time and sending the spike to the other neuron, thus "reloading" each other continuously. Moreover, each time that neuron 2 sends a spike to neuron 1, it also emits a spike to the environment. Therefore, the functioning of the system is deterministic, only one spike train is produced.

Thus,

$$ST(\Pi_1) = \{\langle 1, 2, 3, 4, \ldots \rangle\},$$
$$N_\alpha^\beta(\Pi_1) = \{1\}, \text{ for all } \alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\},$$
$$\text{and either } \beta = a \text{ or } \beta \text{ is omitted},$$
$$N_\alpha^\beta(\Pi_1) = \emptyset, \text{ for all } \alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\}, \text{ and } \beta \in \{h, ha\}.$$

The pair of neurons from Figure 1 will be used as a "sub-system" in many of the systems given in this paper, as a step by step supplier of spikes to other neurons. We use them already in the following example, given in Figure 2, and formally defined as follows:

$$\Pi_2 = (O, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, syn, i_0),$$
$$O = \{a\},$$
$$\sigma_1 = (1, \{a \to a; 0\}),$$
$$\sigma_2 = (1, \{a \to a; 0, \quad a \to a; 1\}),$$
$$\sigma_3 = (0, \{a \to a; 0, \quad a^2 \to \lambda\}),$$
$$\sigma_4 = (1, \{a \to a; 0\}),$$
$$\sigma_5 = (1, \{a \to a; 0\}),$$
$$syn = \{(1,2), (2,1), (1,3), (2,3), (3,4), (3,5), (4,5), (5,4)\},$$
$$i_0 = 4.$$

This system contains two "modules" consisting of pairs of neurons which sustain each other: neurons 4 and 5 are exactly as the two neurons from Π_1, while neurons

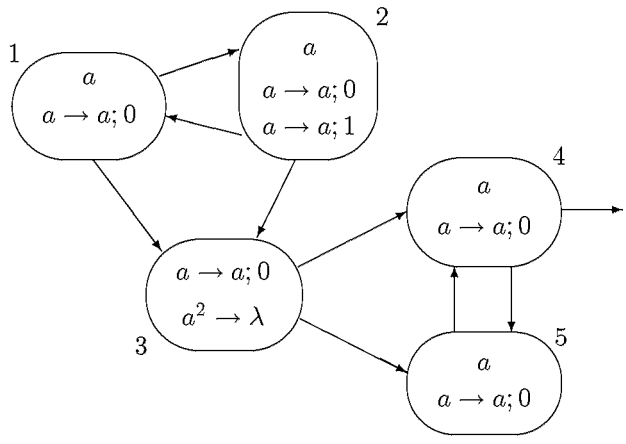

Fig. 2. A non-deterministic SN P system with an infinite spike train.

1 and 2 behave like the two neurons from Π_1 as long as neuron 2 fires by means of the first rule, $a \to a; 0$ (thus, this neuron behaves non-deterministically, and it is the only non-deterministic neuron in Π_2). Let us analyze the functioning of Π_2 beginning with the output. As long as (the output) neuron 4 contains exactly one spike, it fires and sends out a spike. This happens already in the initial moment, hence all spike trains begin with a sequence of 1's. The spiking of neurons 4 and 5 halts only when they get spikes from neuron 3. In turn, neuron 3 spikes only if it contains only one spike, but as long as both neurons 1 and 2 send a spike to neuron 3, it will contain two spikes and use the forgetting rule $a^2 \to \lambda$. Neurons 1 and 2 send two spikes to neuron 3 as long as both of them use the rule $a \to a; 0$.

However, at any moment, starting with the first step, already neuron 2 can use the rule $a \to a; 1$. When this happens, then at this step neuron 2 sends no spike to neurons 1 and 3, and moreover it gets blocked. But then none of neurons 1 and 2 will spike in the next moment, while neuron 3 has only one spike and so it will fire. In the next step, neuron 3 uses the rule $a \to a; 0$, and so it sends a spike to each of neurons 4 and 5. From now on these two neurons are idle, because they do not have rules for more than one spike. Thus, neuron 4 does not emit spikes anymore, and the spike train continues to infinity with 0's.

Note that the internal functioning of Π_2 does not stop after neurons 4, 5 get blocked: the spike of neuron 2, produced by the rule $a \to a; 1$, will be sent in the next step to both neurons 1 and 3, and both these neurons fire again; neuron 1 sends spikes to both neurons 2 and 3, which also fire, and the process continues forever, sending spikes to neurons 4 and 5, which never fire again.

Even if neuron 2 fires in the first step by using the rule $a \to a; 1$, neurons 4 and 5 are blocked only from step 3 on, hence all spike trains start with at least two 1's. Consequently,

$$ST(\Pi_2) = \{\langle 1, 2, 3, \ldots, k\rangle \mid k \geq 2\}.$$

Note that a spike emitted by a neuron can branch into two or more spikes, if there are several synapses leading out of the neuron: the number of branched spikes equals the number of outgoing synapses. Such a branching is present in Π_2 for neurons 1, 2, 3 (also for neuron 4 we have a branching where one spike is emitted as an output).

Let us also consider now the SN P system Π_3 given in Figure 3.

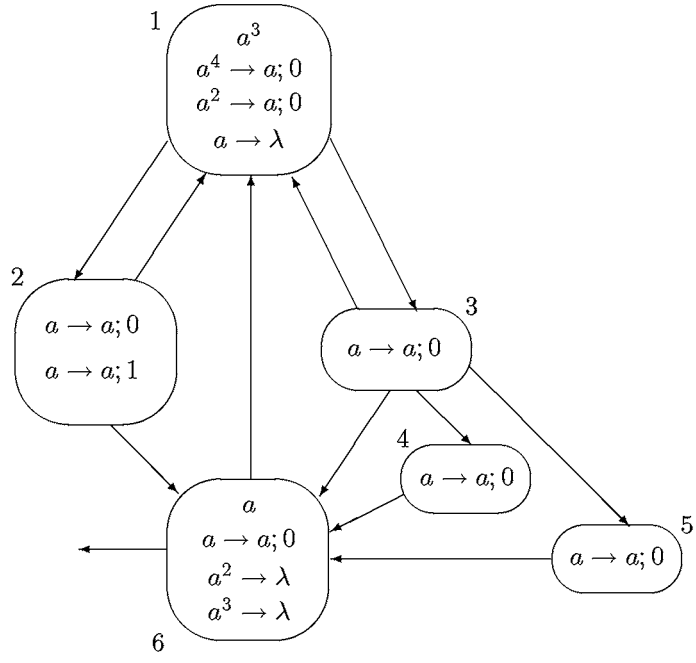

Fig. 3. An SN P system computing an arithmetical progression.

In the initial configuration there are spikes in neurons 1 and 6 (neuron 6 is the output neuron). Neuron 1 cannot use any rule, while neuron 6 fires and spikes immediately. Its spike exits the system and it also reaches neuron 1, which now can fire using the rule $a^4 \to a; 0$. The spike of neuron 1 reaches both neurons 2 and 3.

If neuron 2 uses the rule $a \to a; 0$, then neuron 6 does not fire again, because it has to forget the two spikes (from neurons 2 and 3), but neuron 1 fires and spikes again. The spike of neuron 3 is also sent to neurons 4 and 5, which pass the spikes to neuron 6, which forgets them. The process is repeated an even number of steps.

If neuron 2 uses instead the rule $a \to a; 1$, then in the next step neuron 1 receives only one spike, from neuron 3, and forgets it. In turn, neuron 6 fires, using the spike received from neuron 3; besides the spike sent out, it also sends a spike to neuron 1. In the next step, also the spike of neuron 2 reaches neurons 1 and 6, but neuron 6 receives at the same time the two spikes from neurons 4 and 5. This means that now neuron 1 has two spikes, and neuron 6 has three (it also gets two spikes from neurons 4 and 5). While neuron 1 fires and spikes, neuron 6 forgets its two spikes. This means that the process can be repeated, starting with the spike of neuron 1, which happens in the next step after neuron 6 spikes. But this is exactly as in the initial configuration.

It is easy to see that exactly $2i + 1$ steps, for some $i \geq 1$, elapse between two consecutive spikes of neuron 6. Thus, all computations γ of Π_3 last forever and $N_\alpha^\beta(\Pi_3) = \{1 + 2i \mid i \geq 1\}$, for all $\alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\}$, and for β either missing or equal to a (not for halting cases).

Table 1. A computation of the system from Fig. 3.

Neuron	1	2	3	4	5	6	env
initial	a^3	—	—	—	—	a	
Step 1	—	—	—	—	—	$a \to a; 0$	a
	$a^3 a_6$	—	—	—	—	—	
Step 2	$a^4 \to a; 0$	—	—	—	—	—	
	—	a_1	a_1	—	—	—	
Step 3	—	$a \to a; 0$	$a \to a; 0$	—	—	—	
	$a_2 a_3$	—	—	a_3	a_3	$a_2 a_3$	
Step 4	$a^2 \to a; 0$	—	—	$a \to a; 0$	$a \to a; 0$	$a^2 \to \lambda$	
	—	a_1	a_1	—	—	$a_4 a_5$	
Step 5	—	$a \to a; 1$	$a \to a; 0$	—	—	$a^2 \to \lambda$	
	a_3	—	—	a_3	a_3	a_3	
Step 6	$a \to \lambda$	spiking	—	$a \to a; 0$	$a \to a; 0$	$a \to a; 0$	a
	$a_2 a_6$	—	—	—	—	$a_2 a_4 a_5$	
Step 7	$a^2 \to a; 0$	—	—	—	—	$a^3 \to \lambda$	
	—	a_1	a_1	—	—	—	
Step 8	—	$a \to a; 1$	$a \to a; 0$	—	—	—	
	a_3	—	—	a_3	a_3	a_3	
Step 9	$a \to \lambda$	spiking	—	$a \to a; 0$	$a \to a; 0$	$a \to a; 0$	a
	$a_2 a_6$	—	—	—	—	$a_2 a_4 a_5$	
Step 10	$a^2 \to a; 0$	—	—	—	—	$a^3 \to \lambda$	
	—	a_1	a_1	—	—	—	

Because the work of the system Π_3 is rather intricate to follow, we also provide a

step by step analysis of a possible computation in Table 1. Ten steps are considered here, with spikes sent out in steps 1, 6, and 9 (hence in-between the three spikes we compute numbers 5 and 3, respectively). In each row of the table, we give for each neuron the used rule and below it the spikes present in that neuron after completing the step, with recently received spikes having subscripts which indicate the origin of those spikes; if no rule is used, or no spike remains, then we use a dash. Note that the configuration of the system is the same at the end of steps 2, 7, 10, hence immediately after spiking, which confirms the observation made above about the possibility of iterating indefinitely this behavior of Π_3.

We will return to this example in Section 7.

6. Universality Results in the General Case

As expected (in view of [5]), also when we consider sets $N_\alpha^\beta(\Pi)$, defined in Section 4, we obtain characterizations of Turing computability.

In Table 2 we synthesize all the results we know about the families of sets $N_\alpha^\beta(\Pi)$ ("univ" means "universality", with the indication of the theorem where the respective case is settled, and a line stands for a case which does not make sense). The notations for the respective families are as follows: $Spik_\alpha^\beta P_m(rule_k, cons_p, forg_q)$ is the family of sets $N_\alpha^\beta(\Pi)$, for all systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, and each forgetting rule removing at most q spikes; then, $\alpha \in \{all, \omega\} \cup \{k, \underline{k} \mid k \geq 2\}$, and β either omitted or belonging to the set $\{h, a, ha\}$. As usual, a parameter m, k, p, q is replaced with $*$ if it is not bounded. Note that in Table 2 we do not give the specific parameters m, k, p, q for families of the form $Spik_\alpha^\beta P_m(rule_k, cons_p, forg_q)$, but these parameters can be found in the statements of the appropriate theorems.

Table 2. Results known about families $Spik_\alpha^\beta P_m(rule_k, cons_p, forg_q)$.

α \ β	arbitrary	halting	alternate	halting & alternate
2	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]
$\underline{2}$	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]
k	univ [Th. 5]	univ [Th. 6]	univ [Th. 5]	univ [Th. 3]
$\underline{k}$	univ [Th. 8]	univ [Th. 6]	univ [Th. 8]	univ [Th. 3]
ω	univ [Th. 5]	—	univ [Th. 2]	—
all	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]	univ [Th. 1]

Several of the results mentioned in Table 2 are a direct consequence of the universality result from [5], of the proof of this result in [5], and of the previous definitions.

Indeed, in the notation of the present paper, the universality result from [5] is written in the following form:

Theorem 1. *$Spik_{\underline{2}}^{\beta}P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 3$, and either $\beta = h$ or β is omitted.*

Then, in the proof of this equality from [5] one constructs an SN P system Π which simulates a register machine M; the computations of Π either never halt (when they correspond to non-halting computations in M), or spike exactly twice, and then halt. This means that for Π we have

$$N_2(\Pi) = N_{\underline{2}}(\Pi) = N_2^{\beta}(\Pi) = N_{\underline{2}}^{\beta}(\Pi), \text{ for all } \beta \in \{h, a, ha\}.$$

This fills in the first two lines from the table (that is why we mention Theorem 1 as the source of these results).

Since for the system Π constructed in the proof from [5] we have $N_\omega(\Pi) = N_\omega^a(\Pi) = \emptyset$, the arguments above also imply the last row of Table 2.

Let us now consider the two universality results mentioned in the ω line. The simplest case is that of the alternate sets of numbers.

Theorem 2. *$Spik_\omega^a P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 3$.*

Proof. This is again a consequence of the proof of Theorem 1 from [5]. Take the system Π constructed there for simulating a register machine M (its work starts from the neuron labeled by l_0, the initial label of M, and it ends with the output neuron, i_0, spiking twice, at an interval of length n for some $n \in N(M)$). We add two new neurons with labels c_1, c_2 as indicated in Figure 4. They have no spikes at the beginning and contain the rule $a^2 \to a; 0$. Each of these neurons receives the spikes emitted by neuron i_0 and sends spikes to the neuron l_0. When i_0 spikes first time, the new neurons do not fire, they keep the spike until i_0 spikes again. At that moment (the computation in Π halts, and) both c_1 and c_2 spike, and this fires again the initial neuron of Π. In this way, another computation of M is simulated.

Let Π' be this new system. Of course, if a computation in the register machine M does not halt, then the computation in Π' does not halt, but it spikes only a finite number of times, hence does not contributes to $N_\omega^a(\Pi')$. However, there is a computation in Π' where we always "guess correctly" and we always start a new computation in M which halts – maybe we start again and again the same computation. Hence it suffices to have $N(M)$ non-empty, and it is not necessary that $N(M)$ is infinite. For such a computation γ, $st(\gamma)$ is infinite and the alternate distances between spikes give exactly the set $N(M) = N_{\underline{2}}^h(\Pi)$, that is, $N_\omega^a(\Pi') = N(M)$.

The observation that the new neurons have rules of the complexity required by the parameters in the statement of the theorem completes the proof. □

The previous construction can be supplemented with a module which can stop

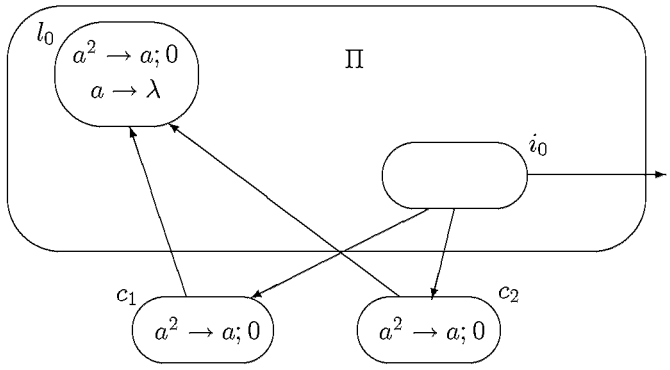

Fig. 4. The idea of the construction from the proof of Theorem 2.

the computation after a given number of spikes, thus also covering the cases $N_\alpha^{ha}(\Pi)$ for $\alpha \in \{k, \underline{k} \mid k \geq 2\}$.

Theorem 3. *$Spik_\alpha^{ha} P_*(rule_h, cons_p, forg_q) = NRE$ for all $h \geq 2, p \geq \max(3, k)$, $q \geq 3$, and $\alpha \in \{k, \underline{k} \mid k \geq 2\}$.*

Proof. As in the previous proof, we start again from the system Π constructed in the proof of Theorem 1 from [5], making the observation that the output neuron of Π has only the rule $a \to a; 0$. Now, the idea is to collect the spikes sent from this neuron in a new neuron, d_1, counting to k; when k spikes are collected, this neuron will fire, its spike will be multiplied by the three neighboring neurons, d_2, d_3, d_4, which spike and send their spikes to neurons c_1, c_2, which are used as in Figure 4 to restart the work of Π; moreover, d_2 and d_3 send spikes also to the output neuron of Π. In this way, neurons c_1, c_2, and i_0 can never use again a rule, because they collected too many spikes insides, which means that Π halts after sending out k spikes. The suggested construction is illustrated in Figure 5.

It is instructive to note that the fact that neuron i_0 gets "flooded" three steps after the kth spike does not cause problems for the system, because also Π needs at least three steps before spiking again: one step when i_0 fires and sends the spikes to c_1, c_2, one step when c_1 and c_2 spike and send their spikes to neuron l_0, and one step when neuron l_0 spikes, restarting the work of Π (we also have $l_0 \neq i_0$). □

The above construction does not work for the halting non-alternating case, which we will discuss later.

We give now a technical lemma which will be useful in settling three more cases from Table 2.

Lemma 4. *Let Π be an SN P system such that each computation γ of Π has $st(\gamma)$ either empty or $st(\gamma) = \langle t, t + n \rangle$. There is an SN P system Π' with the following*

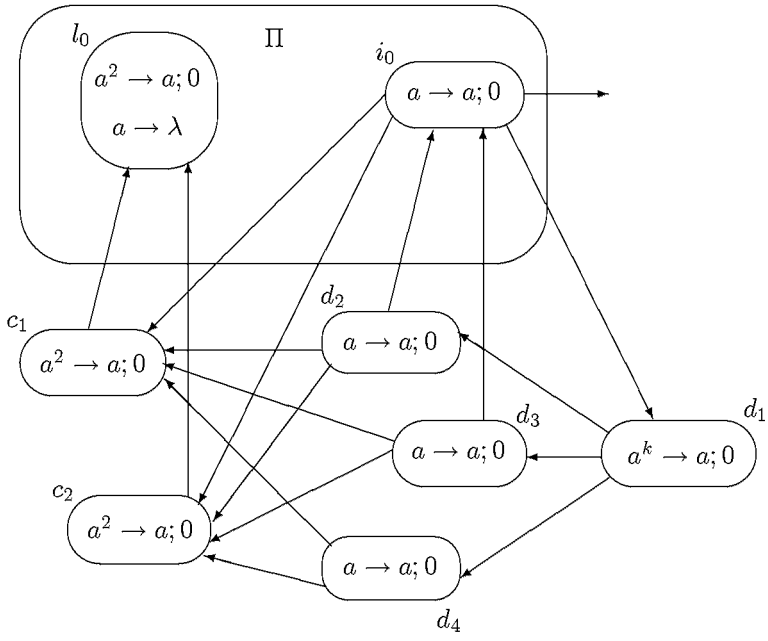

Fig. 5. The idea of the construction from the proof of Theorem 3.

properties:

(1) for each computation γ of Π with $st(\gamma) = \langle t, t+n \rangle$, there is a computation γ' of Π' such that $st(\gamma') = \langle t, t+(n+1), t+2(n+1), \ldots \rangle$;

(2) each computation γ' of Π' either never spikes, or has $st(\gamma') = \langle t, t+(n+1), t+2(n+1), \ldots \rangle$ for $\langle t, t+n \rangle$ being the spike train of a computation of Π.

Proof. We will sketch (somewhat informally) the construction behind the proof – it is illustrated in Figure 6. To the given system Π with the output neuron i_0, we add 13 further neurons, grouped in three modules of four neurons each, as well as a new output neuron, *out*. The three modules, indicated by dashed boxes in Figure 6, work as follows.

The module *Initialize* loads $2n+1$ spikes in neuron m_1, provided that the system Π spikes at times t and $t+n$, for some $n \geq 1$. This happens in the following way. All new neurons are empty at the beginning. When the system Π spikes first time (moment t), a spike arrives in each of the neurons c_1, c_2, c_3. Then c_2 and c_3 fire and spike immediately; their spikes are sent both to neuron m_1 and to each other. Therefore, in the next step, both c_2 and c_3 spike again, and the process is repeated as long as Π does not spike for the second time.

Neuron c_1 keeps the first spike until Π spikes again – this happens at the moment $t+n$. Simultaneously, c_2 and c_3 spike, hence in the next step they have to use the

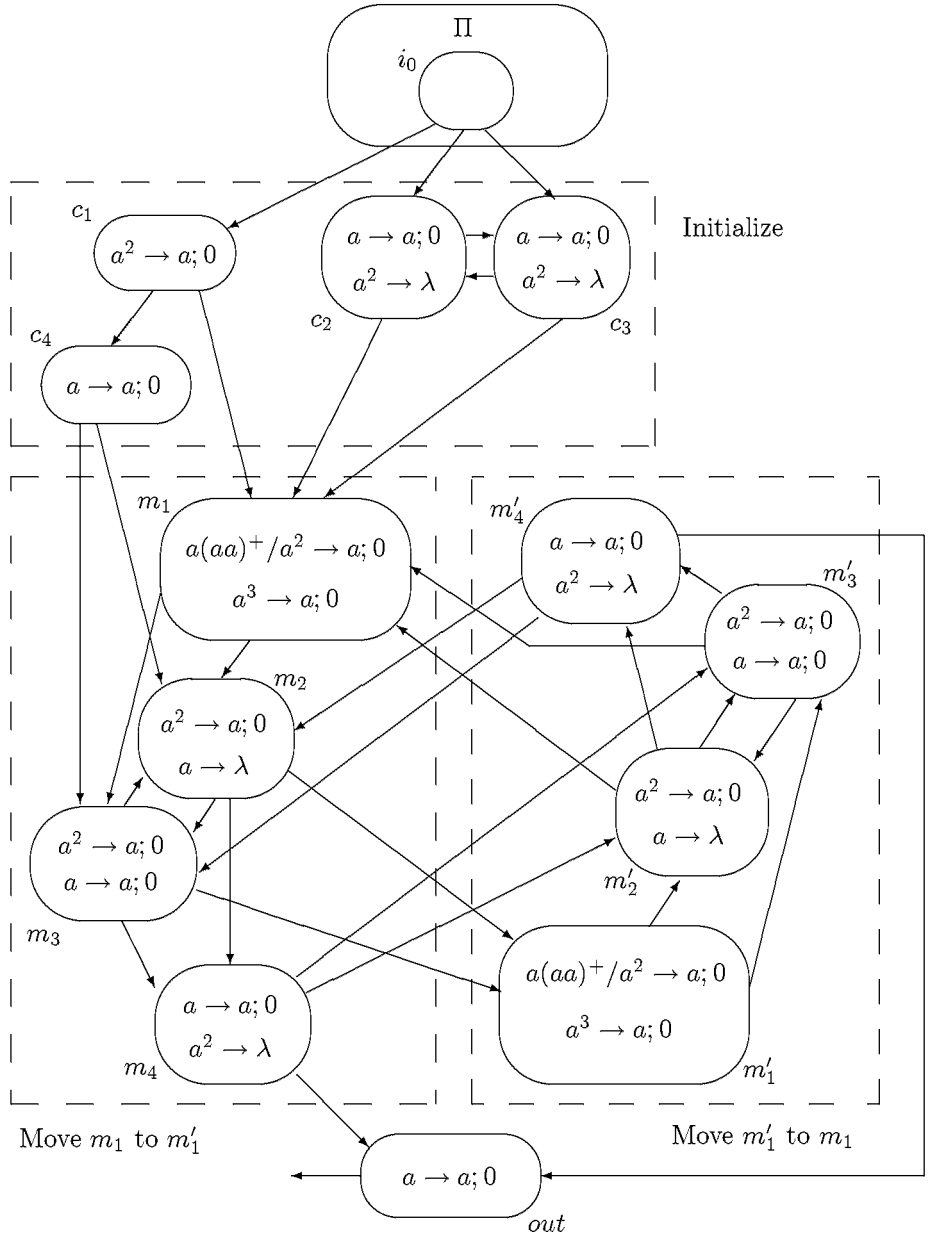

Fig. 6. The idea of the proof of Lemma 4.

forgetting rule $a^2 \to \lambda$, because both of them contain two spikes. At the same time, neuron c_1 fires for the first (and the only) time. This means that altogether neuron m_1 receives $2n + 1$ spikes.

During all this processing, neuron m_1 contains an even number of spikes, except for the last moment, when it gets an odd number of spikes. When neuron m_1 receives a spike from c_1, also c_4 receives a spike at the same step.

This initiates the module *Move* m_1 *to* m'_1. This module moves the contents of neuron m_1 into neuron m'_1, and in the final step of this moving process it sends a spike to the output neuron *out*. This causes system Π' to spike, and also initiates the module *Move* m'_1 *to* m_1.

This whole process is carried out in the following way. At some moment, both neuron m_1 and neuron c_4 spike. This means that both neurons m_2, m_3 get two spikes each. They fire and spike, which results in sending two spikes to neuron m'_1, as well as in sending one spike to each other. This happens simultaneously with the firing of neuron m_1 – which fires as long as at least three spikes are inside (note that after each use of the rule $a(aa)^+/a^2 \to a; 0$ the number of spikes in neuron m_1 remains odd, hence the rule can be used again). The process is repeated, and in each step the contents of m_1 decreases by 2 and that of neuron m'_1 increases by two.

When the contents of neuron m_1 is exhausted, hence the rule $a^3 \to a; 0$ is used, two more spikes are sent to neuron m'_1, but in the next step only neuron m_3 fires (both m_2 and m_3 contain one spike, and m_2 has to forget it). In this way, neuron m'_1 gets $2n + 1$ spikes, and this happens exactly at the moment when also neuron m_4 is ready to spike: as long as both m_2 and m_3 spike, neuron m_4 forgets the two spikes it receives, but when only m_3 spikes, neuron m_4 fires. The spike of m_4 reaches both the output neuron *out* – hence the system spikes, and the neurons m'_2, m'_3 of the module *Move* m'_1 *to* m_1.

The task of this module is analogous to that of the module *Move* m_1 *to* m'_1: the contents of m'_1 is moved to neuron m_1 and at the end a spike is sent to the output neuron.

Note that in the same way as neuron c_4 has sent spikes to m_2 and m_3 when initiating the module *Move* m_1 *to* m'_1, now m_4 sends spikes to neurons m'_2, m'_3, thus initiating the module *Move* m'_1 *to* m_1. Similarly, on completion of the task of this module, its neuron m'_4 sends spikes to neurons m_2, m_3, triggering again the module *Move* m_1 *to* m'_1.

The computation in Π' never stops, and between any two consecutive spikes we have $n + 1$ steps: in n steps we use the rules of either m_1 or of m'_1, hence both m_2 and m_3, or m'_2 and m'_3, respectively, are fired (thus $2n$ spikes are moved), and one further step is necessary for sending the last spike from m_3 or m'_3, to m'_1 or m_1, respectively.

It should be clear from the above explanations that each computation γ of Π, with $st(\Pi) = \langle t, t + n \rangle$, is "prolonged" to an infinite computation in Π', spiking repeatedly, and, conversely, each computation in Π' corresponds to a computation in Π in the way stated in the lemma. □

This lemma can be used to obtain again universality results for the cases we consider as consequences of Theorem 1 (the case of $Spik^a_\omega P_*(\ldots)$ is covered again):

Theorem 5. *$Spik^\beta_\alpha P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 3$, and for $\alpha \in \{\omega\} \cup \{k \mid k \geq 3\}, \beta = a$ or β is omitted.*

Proof. Let $Q \in NRE$ be an arbitrary set and let $Q' = \{n-1 \mid n \in Q\}$; clearly, also $Q' \in NRE$. According to Theorem 1 there is an SN P system Π such that $N^h_{\underline{2}}(\Pi) = Q'$. Applying Lemma 4 to Π we get a system Π' as in the statement of Lemma 4. Clearly, $N_\omega(\Pi') = N_k(\Pi') = \{n+1 \mid n \in N^h_{\underline{2}}(\Pi)\} = Q - \{1\}$ for all $k \geq 2$.

If $1 \notin Q$, then we have $Q = Q - \{1\} \in Spik^\beta_\alpha P_*(rule_k, cons_p, forg_q)$ with k, p, q and α, β as in the theorem, and we are done.

If $1 \in Q$, then we also consider the system Π_1 from Figure 1, for which we have $N_\omega(\Pi_1) = N_k(\Pi_1) = \{1\}$. Now, we just observe that the families of the form $Spik^\beta_\alpha P_*(rule_k, cons_p, forg_q)$ are closed under union: the proof of the corresponding result from [5] (carried out there for $Spik^h_{\underline{2}} P_*(rule_k, cons_p, forg_q)$) is valid also in our cases. Then, we perform the union construction from [5] for Π' and Π_1, obtaining a system Π'' which computes exactly Q, both as $N_\omega(\Pi'')$ and as $N_k(\Pi'')$, and in both cases also for the alternating mode.

Thus, the theorem holds. □

We can now combine the constructions from the proofs of Theorem 3 and of Lemma 4: a "flooding" module, like the one composed of neurons d_1, d_2, d_3, d_4 in Figure 5, is added to the construction from Figure 6, sending three spikes to neurons m_4, m'_4, and *out*. In this way, both "triggering" neurons m_4 and m'_4 get blocked after k spikes, hence the system halts. Consequently, we also obtain the following result, which completes the result from Theorem 3:

Theorem 6. *$Spik^h_\alpha P_*(rule_h, cons_p, forg_q) = NRE$ for all $h \geq 2, p \geq \max(3, k)$, $q \geq 3$, and $\alpha \in \{k, \underline{k} \mid k \geq 2\}$.*

Somewhat conversely to Lemma 4, the following auxiliary result ensures the passage from infinite spike trains to trains of a specified length.

Lemma 7. *Given an SN P system Π we can construct an SN P system Π_k such that:*

(1) for each computation γ of Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_j \rangle$, $j \leq k$, there is a computation γ' of Π' such that $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_j + 2 \rangle$;

(2) for each computation γ of Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_k, \ldots \rangle$, there is a computation γ' of Π' such that $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_k + 2 \rangle$;

(3) each computation γ' of Π' either (i) never spikes, or (ii) has $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_j + 2 \rangle$ for some computation γ in Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_j \rangle$, $j \leq k$, or (iii) has $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_k + 2 \rangle$ for some computation γ in Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_k, \ldots \rangle$.

Proof. For a given $k \geq 1$ and an SN P system Π, we construct the SN P system Π' in the way indicated in Figure 7. The spikes of i_0, the output neuron of Π, are sent to both new neurons c_1 and c_2. While c_1 fires and spikes immediately, thus just delaying by two steps the moments when Π spikes, neuron c_2 accumulates the spikes until it collects $k+1$ spikes. Then c_2 starts to fire, hence in the new output neuron, *out*, we get two spikes at each moment that Π spikes, which means that the neuron *out* forgets all spikes beginning with the $(k+1)$th spike. Thus, if the computation in Π had a spike train of less than k spikes, then all these spikes are sent out, but if there are more than k spikes, then they are truncated to the first k ones. □

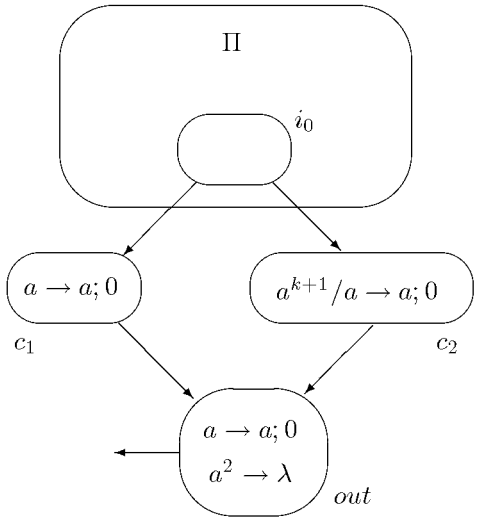

Fig. 7. The idea of the proof of Lemma 7.

Theorem 8. *$Spik_{\underline{k}}^{\beta}P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 3$, and for $\beta = a$ or β is omitted.*

Proof. This is a direct consequence of Lemma 4, Lemma 7, and of Theorem 1. Indeed, take a system Π such that $N_{\underline{2}}^{h}(\Pi)$ is a given set from NRE. We apply Lemma 4 to this system as well as the possible completion by union as in the proof of Theorem 5 obtaining a system Π' whose computations just repeat spiking at identical intervals. Now, to Π' we apply the construction from Lemma 7, obtaining a system Π'' with all spike trains truncated at exactly k spikes (this is the case, because the spike trains of Π' are all infinite). Then, it is clear that in both the arbitrary and the alternate modes we compute the same set of numbers. □

The construction from the proof of Lemma 7 does not stop the functioning of Π', but only the emission of spikes, hence we do not get the universality also for the halting cases (alternating or not). In turn, the construction from the proof of Lemma 4 simply prolongs the computation indefinitely. It is for this reason that in the halting case we had to use a different technique, viz., the one from the proofs of Theorems 3 and 6.

In some sense, the above proofs for the ω cases based on Lemma 4 are not "fair", because although the computations are infinite, the intervals between consecutive spikes are identical, each computation contributes only one number to the computed set. This motivates the following notions of fairness (we call this "coherence"):

- A system Π is said to be *strongly ω-coherent* if for all computations γ_1, γ_2 in Π we have $N(\gamma_1) = N(\gamma_2)$, where for any computation γ with $st(\gamma) = \langle t_1, t_2, \ldots \rangle$, $N(\gamma) = \{t_i - t_{i-1} \mid i \geq 2\}$. (Note that this does not mean that the two computations have the same spike trains, but only that the sets of intervals between consecutive spikes are the same.)
- A system Π is said to be *weakly ω-coherent* if there is a computation γ in Π such that $N(\gamma) = N_{all}(\Pi)$ (that is, there is a computation which provides all numbers which all other computations can provide).

Of course, the strong coherence implies the weak coherence: in a strongly ω-coherent system all computations fulfill the property of weak coherence.

The proofs above do not take care of these properties; extending the results in Table 2 to weakly and strongly coherent systems remains an *open problem.*

7. Spike Trains in the Case of Bounded Systems

We will investigate now only sets of numbers which can be computed by computations with a bounded number of spikes in any neuron. Thus, we will consider families $Spik^{\beta}_{\alpha}P_m(rule_k, cons_q, forg_p, bound_s)$, which correspond to families $Spik^{\beta}_{\alpha}P_m(rule_k, cons_q, forg_p)$, with the additional restriction that for an SN P system Π we consider sets of numbers of the form $N_{\alpha}(\Pi, s)$, resulting from computations where each neuron can contain at most s spikes (if a computation reaches a configuration where a neuron has more that s spikes, then this computation aborts and provides no result) – such computations are called s-bounded.

The following characterization of semilinear sets of numbers was proved in [5]:

Theorem 9. *$NREG = Spik^{\beta}_{2}P_*(rule_k, cons_q, forg_p, bound_s)$, for all $k \geq 3$, $q \geq 3$, $p \geq 3$, and $s \geq 3$, with $\beta = h$ or β is omitted.*

The proof of the inclusion $Spik^{\beta}_{2}P_*(rule_k, cons_q, forg_p, bound_s) \subseteq NREG$ in [5], can be extended in a direct way to a proof for the inclusion $Spik^{\beta}_{\alpha}P_*(rule_*, cons_*, forg_*, bound_*) \subseteq NREG$ for all α and β as considered here.

Unfortunately, we cannot use the technique from the previous section in order to extend the proof of the converse inclusion, $NREG \subseteq Spik^{\beta}_{2}P_*(rule_k, cons_q,$

$forg_p, bound_s$), from the case $\alpha = \underline{2}, \beta = h$ to other cases, because the proof of Lemma 4 introduces an unbounded number of spikes in the neurons. However, the steps used in [5] for proving this inclusion (for $\alpha = 2$ and $\beta = h$) can be modified, for cases $k, \underline{k}$, and ω, also for the alternating definition of the computed set, so a result as that in Theorem 9 is true also for other cases than those covered by the theorem.

Let us consider now each of these steps.

By Lemma 8.2 from [5], singleton sets $\{n\}, n \geq 1$, are in $Spik_{\underline{2}}^{h} P_1(rule_2, cons_1, forg_0, bound_1)$. Now, Lemma 4 applies, growing the number of spikes to $2n + 1$, hence to a bounded amount. Therefore, singleton sets are in the family $Spik_{\alpha}^{\beta} P_{14}(rule_2, cons_3, forg_2, bound_{2n+1})$.

Lemma 8.3 from [5] proves that each arithmetical progression $\{ni \mid i \geq 1\}$ with $n \geq 3$ is in $Spik_{\underline{2}}^{h} P_{n+2}(rule_3, cons_3, forg_2, bound_3)$. The computation of the SN P system Π used in the proof in [5] halts after the second spiking, but this can be avoided by replacing the rule $a^2 \to \lambda$ from neuron 1 of Π with the rule $a^2 \to a; 0$. In this way, after any spike the system is "re-initialized" and will spike again after a number of steps of the form ni for some $i \geq 1$. Thus, each arithmetical progression of the above form belongs to all our families $Spik_{\alpha}^{\beta} P_m(rule_k, cons_q, forg_p, bound_s)$ with $\beta \notin \{h, ha\}$.

However, the passage from "pure" arithmetical progressions, of the form $\{ni \mid i \geq 1\}$, to arbitrary progressions, of the form $\{r + ni \mid i \geq 1\}$, for some $r \geq 1$, is based in [5] on a lemma saying that to the elements of a set $Q \in Spik_{\underline{2}}^{h} P_m(rule_k, cons_p, forg_q, bound_s)$ we can add a constant r, thus obtaining the set $\{j + r \mid j \in Q\}$ which belongs to the same family. We do not see how to extend this lemma also to systems with infinite spike trains – therefore we will prove directly that all arithmetical progressions are in our new families.

This has been already shown for a particular case ($r = 1$, not covered by the next lemma), in the example from Figure 3. The construction from this example can be generalized in order to obtain the following result.

Lemma 10. *Each arithmetical progression of the form* $\{r + 2i \mid i \geq 1\}$, $r \geq 2$, *is in* $Spik_{\alpha}^{\beta} P_{r+4}(rule_2, cons_2, forg_3, bound_3)$, *for all* $\alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\}$ *and either* $\beta = a$ *or* β *omitted.*

Proof. For a given r as in the statement of the lemma, we consider the SN P system from Figure 8.

This system function in a way similar to the SN P system from Figure 3, with one important difference: neuron 1 is "loaded" with two spikes only r steps after the spiking of neuron 6. Therefore, the distance between any two consecutive spikes which exit the system is of the form $r + 2i, i \geq 1$. The computation never stops. □

A similar assertion is valid for arithmetical progressions with the step greater than 2.

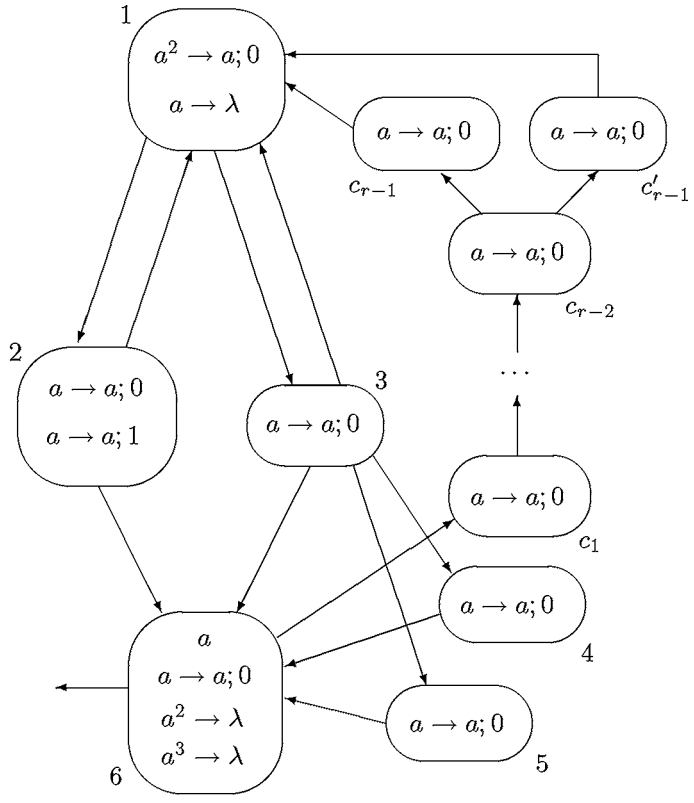

Fig. 8. An SN P system computing $\{r + 2i \mid i \geq 1\}$.

Lemma 11. *Each arithmetical progression of the form* $\{r + ni \mid i \geq 1\}$, $r \geq 1, n \geq 3$, *is in* $Spik^{\beta}_{\alpha}P_{n+r+2}(rule_3, cons_3, forg_4, bound_3)$, *for all* $\alpha \in \{\omega, all\} \cup \{k \mid k \geq 2\}$ *and* $\beta = a$ *or* β *is omitted.*

Proof. Let n, r be as in the statement of the lemma, and consider the SN P system Π from Figure 9.

The functioning of Π is somewhat similar to the functioning of the SN P system from Figure 8, with the output neuron "loading" neuron 1 after r time units, and with the cycle through neurons $1, 2, \ldots, n-2, n-1, n$ repeated an arbitrary number of times, ensuring in this way that the distance between any two consecutive spikes is of the form $r + ni$, for some $i \geq 1$. □

Lemma 8.4 (for each SN P system Π there is an equivalent SN P system Π' containing initially only one spike inside one of its neurons) and Lemma 8.5 (all families of numbers computed by systems with at least two rules, two consumed or used spikes, as well as with at least two spikes contained in the neurons, are closed

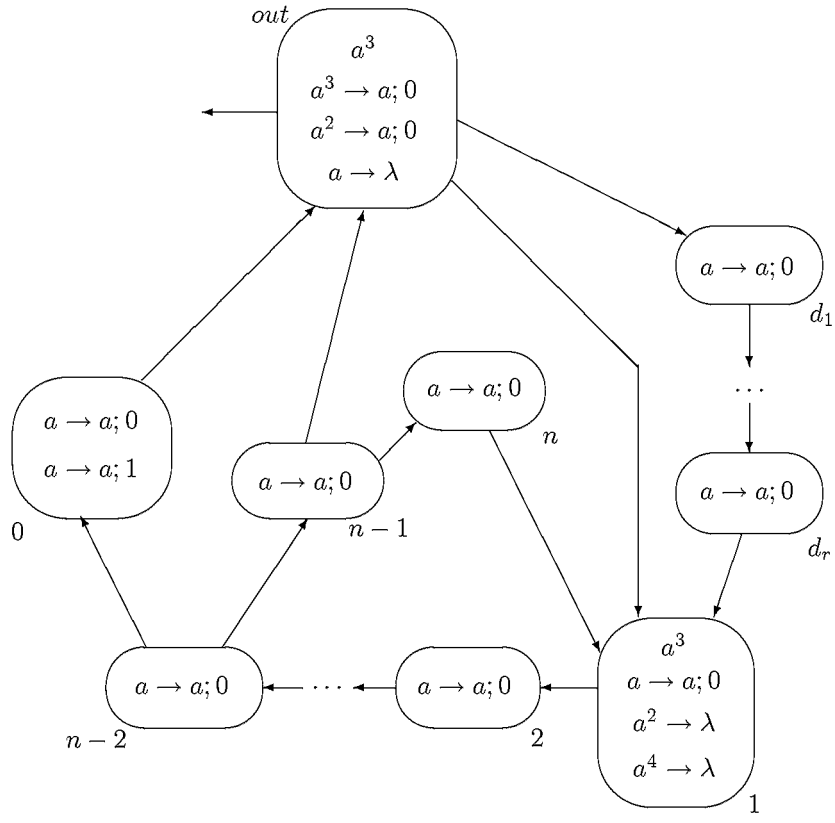

Fig. 9. An SN P system computing $\{r + ni \mid i \geq 1\}$, for $n \geq 3$.

under union) from [5] are true also for our cases (with the same proofs as in [5]).

Using this "union lemma", we ensure that all semilinear sets belong to families of the form $Spik^{\beta}_{\alpha}P_*(rule_k, cons_p, forg_q, bound_*)$ with small values of parameters k, p, q, but not for the case when we consider computations which halt after spiking. For instance, Lemma 7 ensures only that the system does not send out more than k spikes, but the computation can continue forever. However, this can be fixed in the case of bounded computations for any initial SN P system, not only for a particular one, as was the case in the proof of Theorem 6:

Lemma 12. *Given a system* Π *and a threshold* s *on the number of spikes in any neuron, we can construct a system* Π_k *such that:*

(1) for each s*-bounded computation* γ *of* Π *with* $st(\gamma) = \langle t_1, t_2, \ldots, t_j \rangle$, $j \leq k$, *there is a halting* $2s$*-bounded computation* γ' *of* Π' *such that* $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_j + 2 \rangle$;

(2) for each s-bounded computation γ of Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_k, \ldots \rangle$, there is a halting $2s$-bounded computation γ' of Π' such that $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_k + 2 \rangle$;

(3) each computation γ' of Π' either (i) never spikes, or (ii) $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_j + 2 \rangle$ for some computation γ in Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_j \rangle$, $j \leq k$, or (iii) $st(\gamma') = \langle t_1 + 2, t_2 + 2, \ldots, t_k + 2 \rangle$ for some computation γ in Π with $st(\gamma) = \langle t_1, t_2, \ldots, t_k, \ldots \rangle$.

Proof. For a given $k \geq 1$ and an SN P system Π we construct the system Π' as follows (without loss of generality, we may assume that all rules of Π are of the form $a^j \to x$, for some $j \leq s$, because only such rules can be useful in computations – hence we can discard all rules dealing with more than s spikes, also keeping from the languages of regular expressions only the strings of length at most s).

- We add the neurons with the labels *out* (the output neuron of Π') and $1, 2, \ldots, s, s+1$, where all of them are empty in the beginning.
- Neuron *out* contains the rule $a \to a; 0$, and each neuron $i \in \{1, 2, \ldots, s+1\}$ contains the rule $a^k/a \to a; 0$.
- We add the following synapses:
  (i_0, out) and $(i_0, i), 1 \leq i \leq s+1$,
  (i, l) for all $1 \leq i \leq s+1$, for $l = out$, and for all l which are labels of neurons in Π.
- To each neuron of Π as well as to the neuron *out* we add the rules $a^{s+i} \to \lambda$, for all $1 \leq i \leq s+1$.

The so constructed system Π' functions exactly as Π, except that after spiking k times all neurons of Π as well as the neuron *out* are "flooded" by $s+1$ spikes, and, irrespectively of the number of spikes contained in each neuron (from 0 to at most s), they have to forget all of them. Note that the rules $a^{s+i} \to \lambda$ cannot be used before spiking k times, and that no initial rule of Π can be used in the presence of more than $s+1$ spikes. Clearly, the system Π' halts after k spikes. □

Using this lemma, results about the sets $N_k(\Pi)$ can be transferred to the sets $N_{\underline{k}}(\Pi)$ – in case that computations are bounded.

Combining all the results from this section, we can state the following characterization of semilinear sets of numbers.

Theorem 13. *$NREG = Spik^{\beta}_{\alpha} P_*(rule_*, cons_q, forg_p, bound_*)$, for all $q \geq 3, p \geq 3$, and for all $\alpha \in \{\omega, all\} \cup \{k, \underline{k} \mid k \geq 2\}$, and $\beta \in \{h, a, ha\}$ or β is omitted.*

This time we cannot bound the number of rules, because of the proof of Lemma 12, neither give a precise upper bound on the number of spikes present in neurons, because of the way that singleton sets are computed.

8. Discussion

In this paper we have continued to investigate SN P systems introduced in [5].

Although we have settled here all questions about the sets of numbers that we have considered (universality as in Table 2, characterizations of semilinear sets of numbers as in Theorem 13), we believe that this is only the beginning of a systematic study of SN P systems. A lot of questions must be answered, even if we consider only the issue of computing sets of numbers by SN P systems.

To begin with, one can consider various alternative definitions of (various features of) SN P systems. Here is an example of a possible change in the basic definition which can be easily handled. Instead of (or in combination with) considering a time interval between firing and spiking (hence a delay in emitting the spike), it is also natural to consider a delay in transmitting a spike along a synapse. This can be easily formalized by associating natural numbers (delays) to synapses, and assuming that a spike reaches the destination with a given delay. However, it is easily seen that this feature can be handled by SN P systems. A way to do it is to consider intermediate neurons: if it takes k time units for a spike to get from neuron σ to neuron σ', then we add $k-1$ intermediate neurons, each one with the single rule $a \to a; 0$, thus only "forwarding" the spike ahead, which takes one time unit.

Not so clear is a way to address some other issues. For example, one can require various kinds of "structuring" the neurons/synapses graph. What is the power of SN P systems with this graph being, e.g., acyclic, or being "almost a tree" (in the sense that the neurons are arranged in the nodes of a tree, but we also allow additional synapses between neurons at the same depth level of the tree). Then, what is the effect of imposing bounds on the in-degree and/or the out-degree of the graph? In many of the examples and the proofs from this paper we deal with systems that have a limited in- or out-degree; still the problem whether we can have these degrees bounded by 2 (without losing the computing power) remains open.

A number of technical problems concerning the current model remain still open. For example: Can the forgetting rules be avoided? Can the delay in spiking be avoided (i.e., can we consider only rules with delay 0)? Note that the forgetting rules provide ways of "cleaning" the neurons without spiking, while the delay between firing and spiking does not only keeps a spike "hidden", but it also keeps a neuron closed (so that all spikes sent to this neuron "disappear"). Then, what restrictions can be imposed on the regular expressions used in firing rules, without restricting the computing power?

A number of modifications of the SN P model, motivated either by computational or by biological considerations, seem to be interesting to consider. For example: producing several spikes at the same time (hence using rules of the form $E/a^c \to a^q; d$), considering also "anti-spikes" (besides a, to have also $\bar{a}$, which, sent to any neuron, will "annihilate" one local a), allowing also self-synapses (synapses from a neuron to itself), providing ways to change the synapses graph during computations.

Finally, it is natural to consider infinite (sequences of symbols 0 and 1 associated with) spike trains, which we do in a forthcoming paper ([10]).

Note. A careful reading of the paper by two anonymous referees is gratefully acknowledged.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.

[2] J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity*. Springer-Verlag, Berlin, 1995.

[3] M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-related outputs of computations in P systems. *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 107–122.

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

[5] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. Submitted, 2005 (also available at [12]).

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

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

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

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

[10] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.

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

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