Spiking Neural P Systems

Mihai Ionescu
Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
armandmihai.ionescu@urv.net

Gheorghe Păun
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania, and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

Takashi Yokomori
Department of Mathematics, School of Education
Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku
Tokyo 169-8050, Japan
yokomori@waseda.jp

Abstract

This paper is an attempt to incorporate the idea of spiking neurons into the area of membrane computing, and to this aim we introduce a class of neural-like P systems which we call *spiking neural P systems* (in short, SN P systems). In these devices, the time (when the neurons fire and/or spike) plays an essential role. For instance, the result of a computation is the time between the moments when a specified neuron spikes. Seen as number computing devices, SN P systems are shown to be computationally complete (both in the generating and accepting modes, in the latter case also when restricting to deterministic systems). If the number of spikes present in the system is bounded, then the power of SN P systems falls drastically, and we get a characterization of semilinear sets. A series of research topics and open problems are formulated.

1 Introduction

Membrane computing takes as starting point the generic alive cell, considered alone (and in this case the internal compartmentalization by means of hierarchically arranged membranes plays a central role, [10]), or organized in tissues, organs, or other types of cell populations. In the second case, the cells are considered mono-membranar, placed in the nodes of a graph. This extension was first considered in [12], and then further elaborated in [8] and [11] (and in many subsequent papers for which we refer the reader to the bibliography available at [16]). In particular, in [8] also so-called neural-like P systems were introduced, where some ideas from neurobiology are incorporated (synapses, with the replication of impulses in the case of multiple

synapses, linking a neuron to several neighboring neurons, state of a neuron, as a model of the excitation protocol of neurons, etc.). However, an essential aspect is not captured in these systems, namely the fact that most of the neural impulses are almost identical, electrical signals of a given voltage, with a crucial role played by the time when these signals are issued, hence by the intervals between signals. This is an intriguing aspect, in a sharp distinction with many models of computer science: the information is not encoded in a sequence of different "symbols", but in the *sequence of moments* when a unique "symbol", the neuron spike, occurs. Otherwise formulated, in this framework time is (also) a data support; it is not (only) a computing resource as in usual complexity theory, but a way to encode information.

Encoding information in the duration of events, or in the interval of time elapsed between events, and related topics about time in classic computability or in more recent areas, e.g., in neural computing, were discussed in several places; we only mention here a few titles – [4], [5], [14], [15] – as well as a few titles related to spiking neural nets – [2], [6], [7].

The present paper takes this challenge, of using time as data support, and tries to adapt neural P systems to the spiking neurons realm.

Our proposal is the following one (rather simple in principle): let us use only one object, the symbol denoting a spike, and one-membrane cells (called *neurons*) which can hold any number of spikes; each neuron fires in specified conditions (after collecting a specified number of spikes, which are accumulated, added one after another) and then sends one spike along its axon; this spike passes to all neurons connected by a *synapse* to the spiking neuron (hence it is replicated into as many copies as many target neurons exist); between the moment when a neuron fires and the moment when it spikes, each neuron needs a time interval, and this time interval is the essential ingredient of the system functioning (the basic information carrier – with the mentioning that also the number of spikes accumulated in each moment in the neurons provides an important information for controlling the functioning of the system); one of the neurons is considered the output one, and its spikes provide the output of the computation.

This basic idea can be implemented in various ways in a formal model. The variant we choose is rather restrictive: the rules for spiking should take into account *all* spikes present in a neuron not only part of them; not all spikes present in a neuron are consumed in this way; after getting fired and before sending the spike to its synapses, the neuron is idle (biology calls this the refractory period) and cannot receive spikes; a computation is successful only if the output neuron spikes *exactly twice* during the computation; the result of a computation is the number of steps elapsed between the two spikes of the output neuron. There are also rules used for "forgetting" some spikes, rules which just remove a specified number of spikes from a neuron.

Note the important fact that in order for a computation to be successful we do not request that the computation halts, but only to have exactly two spikes sent to the environment.

Even in this restrictive framework, our devices turn out to be Turing complete, able to compute all Turing computable sets of natural numbers, both in the generative mode (as sketched above, a number is computed if it represents the interval between the two consecutive spikes of the output neuron) and in the accepting mode (a number is introduced in the system in the form of the interval of time between the two spikes entering a designated neuron, and this number is accepted if the computation halts). In the former case, SN P systems with only one neuron behaving non-deterministically, choosing between two firing rules to apply, are sufficient. In the latter case, deterministic systems are sufficient.

In the proofs of these results, we use SN P systems which can accumulate arbitrarily many spikes inside. If we want to be "more realistic" and we impose a bound on the number of spikes present in any neuron along a computation (if a neuron gets more spikes, then the computation aborts), then this restriction diminishes the power of our devices: we get in this case a

characterization of semilinear sets of numbers.

By SN P systems with at most two neurons we also get a characterization of finite sets of numbers.

Along the paper, many variants which deserve to be explored are mentioned, as well as more precise open problems. In general, the idea of spiking (neurons) opens a new research vista in membrane computing, and we believe that further investigations are worth carrying out in this area.

In the next section we provide some information about the neuron behavior, then we give the computability prerequisites used in the subsequent sections of the paper. Section 4 introduces the spiking neural P systems. In Section 5 we illustrates the definition with four examples, and we continue (Section 6) with a characterization of finite sets of numbers. In Section 7 we prove that SN P systems are computationally complete (via simulating register machines), while in Section 9 we give the characterization of semilinear sets of numbers by means of SN P systems with a bounded number of spikes in the neurons. We end (Section 10) with a few comments and suggestions for further investigations.

2 Spiking Neurons

We recall here from [1], [6], [7] some notions about the neural cell, mainly focusing on the electric pulses a neuron is transmitting through its synapses; such a pulse is usually called *a spike*, or *action potential*.

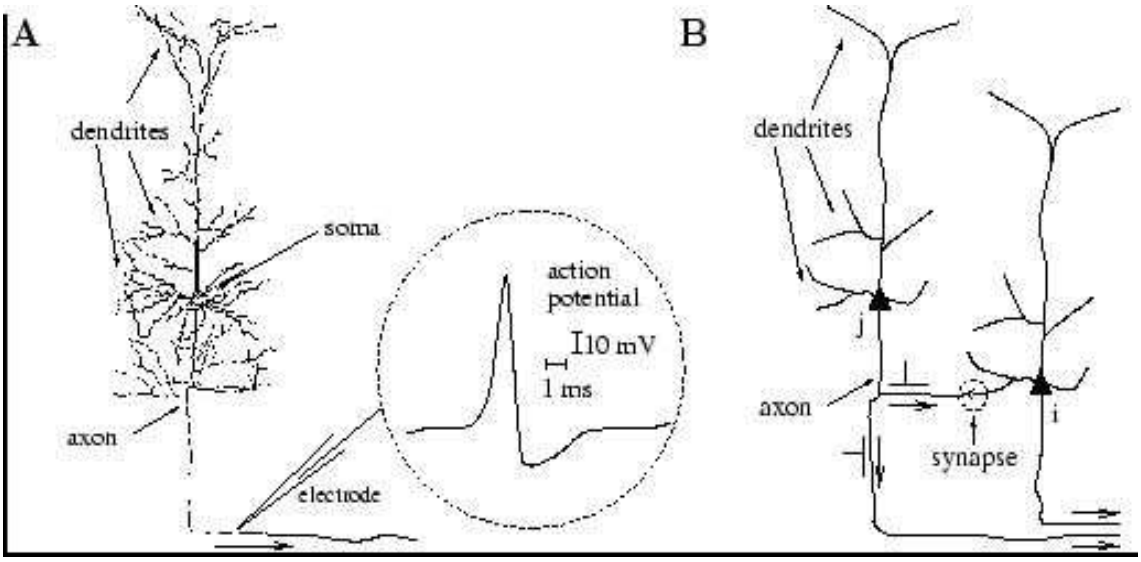

Figure 1: A neuron and its spiking

In Figure 1 (a drawing by Ramón y Cajal, one of the pioneers of neuroscience around 1900, reproduced from [2]), an example of a neural action potential is schematically given altogether with the main parts of a neuron – the cell itself (soma), the axon, the dendrites (a filamentous bush around the soma, where the synapses are established with the endbulbs of the axons of other neurons).

The neuronal signals consist of short electrical pulses (that can be observed as suggested in the figure by placing a fine electrode close to the soma or on the axon of a neuron) having an amplitude of about 100 mV and typically a duration of 1-2 ms. The form of the pulse does not change as the action potential propagates along the axon. A sequence of such impulses which occur at regular or irregular intervals is called a *spike train*. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, *it is the number and the timing of spikes what matter*.

So, the size and the shape of a spike is independent of the input of the neuron, but the *time* when a neuron fires depends on its input.

Action potentials in a spike train are usually well separated. Even with very strong input, it is impossible to excite a second spike during or immediately after a first one. The minimal distance between two spikes defines the refractory period of the neuron.

The contact of the axon of a neuron with the dendrites of another neuron is called a synapse. The most common type of synapse in the vertebrate brain is a chemical synapse. When an action potential arrives at a synapse, it triggers a complex chain of bio-chemical processing steps that lead to a release of neurotransmitter from the presynaptic neuron into the postsynaptic neuron. As soon as transmitter molecules have reached the postsynaptic side, they will be detected by specialized receptors in the postsynaptic cell membrane, and through specific channels, the ions from the extracellular fluid flow into the target cell. The ion influx, in turn, leads to a change of the membrane potential at the postsynaptic site so that, in the end, the chemical signal is translated into an electrical response. The voltage response of the postsynaptic neuron to a presynaptic action potential is called the postsynaptic potential.

In Section 4, we will capture some of these ideas in the framework of neural-like P systems as existing in the membrane computing literature, adapting the definition to the case of spiking.

3 Some Technical Prerequisites

We assume the reader to have some familiarity with language and automata theory, as well as with membrane computing, so that we recall here only a few definitions and we establish the notation we use; for further details we refer to [13] and [11], respectively (with the website of membrane computing, [16], providing recent information about the field).

For an alphabet V, V^* is the free monoid generated by V with respect to the concatenation operation and the identity λ (the empty string); the set of all nonempty strings over V, that is, $V^* - \{\lambda\}$, is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The length of a string $x \in V^*$ is denoted by $|x|$. The family of recursively enumerable languages 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).

We also use below the family $NFIN$, of finite sets of natural numbers, and $SLIN_1$, the family of semilinear sets of natural numbers (the subscript indicates that we work with one-dimensional vectors, not with semilinear sets of vectors in general).

We do not give a definition of semilinear sets of numbers, but we mention that they are the length sets of regular languages. In turn, regular languages are defined (among many other possibilities) by means of *regular expressions* – which will be essentially used also in our main definition in the next section. In short, such an expression over a given alphabet V is constructed starting from λ and the symbols of V and using the operations of union, concatenation, and Kleene +, using parentheses when necessary for specifying the order of operations. Specifically, (i) λ and each $a \in V$ are regular expressions, (ii) if E_1, E_2 are regular expressions over V, then also $(E_1) \cup (E_2), (E_1)(E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a

regular expression over V. With each expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses are omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ is written in the form $(E)^*$.

A language $L \subseteq V^*$ is said to be regular if there is a regular expression E over V such that $L(E) = L$.

In the universality proof from Section 7 we use the characterization of NRE by means of register machines, hence we introduce here also this notion – first, in the non-deterministic, generative form.

A *register machine* is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$ (add 1 to register r and then go to one of the instructions with labels l_2, l_3),
- $l_1 : (\texttt{SUB}(r), l_2, l_3)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known (see, e.g., [9]) that register machines (even with a small number of registers, but this detail is not relevant here) compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents.

A register machine can also work in the *accepting* mode: a number n is introduced in the first register (all other registers are empty) and we start computing with the instruction with label l_0; if the computation eventually halts, then the number n is accepted.

Register machines are universal also in the accepting mode; moreover, this is true even for deterministic machines, having ADD rules of the form $l_1 : (\texttt{ADD}, l_2, l_3)$ with $l_2 = l_3$: after adding 1 to register r we pass precisely to one instruction, without any choice (in such a case, the instruction is written in the form $l_1 : (\texttt{ADD}, l_2)$).

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

We also recall here the initial definition of a neural-like P system as considered in [8], [11] so that we can use it (with some modifications) in the next sections. The basic idea is to consider *cells* related by *synapses* and behaving according to their *states*; the states can model the firing of neurons, depending on the inputs, on the time of the previous firing, etc.

Formally, a *neural-like P system*, of degree $m \geq 1$, is a construct

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

where:

1. O is a finite non-empty alphabet (of *objects*, usually called *impulses*);
2. $\sigma_1, \ldots, \sigma_m$ are *cells* (also called *neurons*), of the form

$$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq m,$$

where:

a) Q_i is a finite set (of *states*);
b) $s_{i,0} \in Q_i$ is the *initial state*;
c) $w_{i,0} \in O^*$ is the *initial multiset* of impulses of the cell;
d) R_i is a finite set of *rules* of the form $sw \to s'xy_{go}z_{out}$, where $s, s' \in Q_i, w, x \in O^*$, $y_{go} \in (O \times \{go\})^*$, and $z_{out} \in (O \times \{out\})^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, 2, \ldots, m\}$ different from i_0;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (synapses among cells);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output cell*.

The standard rules used in this model are of the form $sw \to s'w'$, where s, s' are states and w, w' are multisets of impulses. The mark "go" assigned to some elements of w' means that these impulses have to leave immediately the cell and pass to the cells to which we have direct links through synapses. The communication among the cells of the system can be done in a replicative manner (the same object is sent to all adjacent cells), or in a non-replicative manner (the impulses are sent to only one neighboring cell, or can be distributed non-deterministically to the cells to which we have synapses). The objects marked with "out" (they can appear only in the cell i_0) leave the system. The computation is successful only if it halts, reaches a configuration where no rule can be applied.

The sequence of objects (note that they are symbols from an alphabet) sent to the environment from the output cell is the string computed by a halting computation, hence the set of all strings of this type is the language computed/generated by the system.

We will modify below several ingredients of a neural-like P system as above, bringing the model closer to the way the neurons communicate by means of spikes.

We close this section by mentioning the following **convention**: when evaluating or comparing the power of two number generating/accepting devices, the number 0 is ignored; 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.

4 Spiking Neural P Systems

We pass now from the previous definition of a neural-like P system to a model which makes explicit the restriction to work only with spikes. This means that we have to use only one impulse, hence symbol, identifying a generic electrical neural impulse, a "quanta" of electricity

used in neural interaction. We will also remove the states of neurons; the firing will be controlled by the number of spikes present in a neuron, and by the time elapsed since the previous spiking, in a very simple way; we will also relax the definition of successful computations, removing the halting condition (but adding a sort of simplicity condition: the output neuron can spike only twice).

Specifically, we consider a *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, in 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 by the cell;

   b) R_i is a finite set of *rules* of the following two forms:

      (1) $E/a^r \to a; t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$;

      (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \to a; t$ of type (1) from R_i;

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

The rules of type (1) are *firing* (we also say *spiking*) *rules*: provided that the contents of the neuron (the number of spikes present in it) is described by the regular expression E (we return immediately to this aspect), r spikes are consumed (this corresponds to the result of the quotient of language $L(E)$ with respect to a^r, thus motivating the notation E/a^r from the firing rules), the neuron is fired, and it produces a spike which will be sent to other neurons after t time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized).

We have here two important actions which can take place in a step: getting fired and spiking.

A neuron gets fired when using a rule $E/a^r \to a; t$, and this is possible only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$. This means that the regular expression E "covers" exactly the contents of the neuron. (For instance, a rule $a(aa)^+/a^3 \to a; 1$ can be applied to a neuron which contains seven copies of a, because $a^7 \in L(a(aa)^+) = \{a^{2n+1} \mid n \geq 1\}$, and then only four spikes remain in the neuron; now, the rule cannot be applied again – in general, the rule $a(aa)^+/a^3 \to a; 1$ cannot be applied if the neuron contains any even number of spikes.)

This is an important detail concerning the use of spiking rules, and we stress it again, especially because it contrasts the "standard" way of using rules in P systems, by rewriting parts of a multiset, in the maximally parallel mode both locally (in each compartment) and globally (at the level of the system). Here, at the level of each neuron we work in a sequential mode, with at most one rule used in each step, covering all spikes present in the neuron. Still, we

have a maximal parallelism at the level of the system, in the sense that in each step *all* neurons which can evolve (use a rule) have to do it. We will come back to this aspect.

Now, about spiking. The use of a rule $E/a^r \to a;t$ in a step q means firing in step q and spiking in step $q+t$. That is, if $t=0$, then the spike is produced immediately, in the same step when the rule is used. If $t=1$, then the spike will leave the neuron in the next step, and so on. In the interval between using the rule and releasing the spike, the neuron is assumed *closed* (in the refractory period), hence it cannot receive further spikes, and, of course, cannot fire again. This means that if $t \geq 1$ and another neuron emits a spike in any moment $q, q+1, \ldots, q+t-1$, then its spike will not pass to the neuron which has used the rule $E/a^r \to a;t$ in step q. In the moment when the spike is emitted, the neuron can receive new spikes (it is now free of internal electricity and can receive new electrical impulses). This means that if $t=0$, then no restriction is imposed, the neuron can receive spikes in the same step when using the rule. Similarly, the neuron can receive spikes in moment t, in the case $t \geq 1$.

If a neuron σ_i spikes, its spike is replicated in such a way that one spike is sent to *all* neurons σ_j such that $(i,j) \in syn$, *and* σ_j is open at that moment. If a neuron σ_i fires and either it has no outgoing synapse, or all neurons σ_j such that $(i,j) \in syn$ are closed, then the spike of neuron σ_i is lost; the firing is allowed, it takes place, but it produces no spike.

The rules of type (2) are *forgetting rules*: s spikes are simply removed ("forgotten") when applying $a^s \to \lambda$. Like in the case of spiking rules, the left hand side of a forgetting rule must "cover" the contents of the neuron, that is, $a^s \to \lambda$ is applied only if the neuron contains exactly s spikes.

As defined above, the neurons can contain several rules, without restrictions about their left hand sides. More precisely, it is allowed to have two spiking rules $E_1/a^{r_1} \to a;t_1$, $E_2/a^{r_2} \to a;t_2$ with $L(E_1) \cap L(E_2) \neq \emptyset$ (but not forgetting rules $a^s \to \lambda$ with $a^s \in L(E_i), i=1,2$). This leads to a non-deterministic way of using the rules. If we use the spiking neural P systems in a generative mode (starting from the initial configuration, we evolve non-deterministically, and collect all results of all successful computations – we define immediately the used terms), then we cannot avoid the non-determinism (deterministic systems will compute only singleton sets). In the accepting mode of using our systems, as considered in Section 8, the non-determinism is no longer necessary.

However, we have imposed a *minimal determinism-like restriction* (we can also consider this as a *coherence* condition: either firing or forgetting, without being possible to chose among these two actions): no forgetting rule can be interchanged with a spiking rule. Thus, only in the case of spiking we allow branchings.

As suggested above, the rules are used in the non-deterministic manner, in a maximally parallel way at the level of the system: in each step, all neurons which can use a rule, of any type, spiking or forgetting, have to evolve, using a rule.

A spike emitted by a neuron i will pass immediately to all neurons j such that $(i,j) \in syn$ and are open, that is, the transmission of a spike takes no time, the spikes are available in the receiving neurons already in the next step.

The distribution of spikes in neurons and the states of neurons corresponding to the spiking intervals specified by the last rules used in each neuron (the open-close status and the time since the neurons were closed, depending on the rule used) define the configuration of the system. The initial configuration is defined by the number of initial spikes, $n_1, \ldots, n_m$, with all neurons being open (no rule was used before).

Using the rules in this way, we pass from a configuration of the system to another configuration; such a step is called a *transition*; for two configuration C_1, C_2 of Π we denote by $C_1 \Longrightarrow C_2$ the fact that there is a direct transition from C_1 to C_2 in Π. The reflexive and transitive clo-

sure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$. A sequence of transitions, starting in the initial configuration, is called a *computation*.

With a computation we can associate several results. One possibility is the standard one in membrane computing: to consider only halting computations (reaching a configuration where no rule can be used) and to count the number of spikes present in the output neuron in the halting configuration, or sent to the environment by the output neuron during a halting computation. However, this is not in the style of spiking neurons, that is why we will here consider outputs related to the time when certain events take place. One idea is to consider the moments when the output neuron, that with label i_0, spikes (*not* when it fires), and already we have two possibilities: if neuron i_0 spikes at times $t_1, t_2, \ldots$, then (i) either the set of numbers $t_1, t_2, \ldots$ can be considered as computed by Π, or (ii) the set of intervals between spikes, $t_s - t_{s-1}, s \geq 2$, can be the set computed by Π. Another possibility is to take a sequence of symbols/bits 0 and 1 as the result of a computation, with 0 associated with a moment when the output neuron does not spike and 1 associated with a spiking step. Finite and also infinite sequences of bits can be obtained in this way.

All these possibilities look very attractive, but their study remains as a research topic[1]. In what follows, we consider a further possibility, rather relaxed: we do not care whether or not the computation halts, but we only request that the output neuron spikes exactly twice during the computation. Then, the number of steps elapsed between the two spikes is the number computed by the system along that computation.

We denote by $N_2(\Pi)$ the set of numbers computed in this way by a system Π, with the subscript 2 reminding of the way the result of a computation is defined, and by $Spik_2P_m(rule_k, cons_p, forg_q)$ the family of all sets $N_2(\Pi)$ computed as above by spiking neural P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^r \to a; t$ having $r \leq p$, and all forgetting rules $a^s \to \lambda$ having $s \leq q$. When one of the parameters m, k, p, q is not bounded, then it is replaced with $*$.

5 Examples

We illustrate here the previous definitions with several examples, most of them also useful later.

The first example concerns the system Π_1 given in a graphical form in Figure 2 – and in this way we also introduce a standard way to pictorially represent a configuration of an SN P system, in particular, the initial configuration. Specifically, each neuron is represented by a "membrane" (a circle or an oval), 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 arrows; besides the fact that the output neuron will be identified by its label, i_0, it is also suggestive to draw a short arrow which exits from it, pointing to the environment.

Using this example, we also introduce the following **convention:** if a spiking rule is of the form $E/a^r \to a; t$, with $L(E) = \{a^r\}$ (this means that such a rule is applied when the neuron contains exactly r spikes – and all these spikes are consumed), then we will write this rule in the simpler form $a^r \to a; t$.

In the system Π_1 we have three neurons, with labels 1, 2, 3; neuron 3 is the output one. In the initial configuration we have spikes in neurons 1 and 3, and these neurons fire already in the

[1]The case of infinite binary sequences is investigated in the forthcoming paper "Infinite spike trains in spiking neural P systems", by Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg.

first step. The spike of neuron 3 exits the system, so the number of steps from now until the next spiking of neuron 3 is the number computed by the system. After firing, neuron 3 remains empty, so it cannot spike again before receiving a new spike. In turn, neuron 2 cannot fire until collecting exactly k spikes.

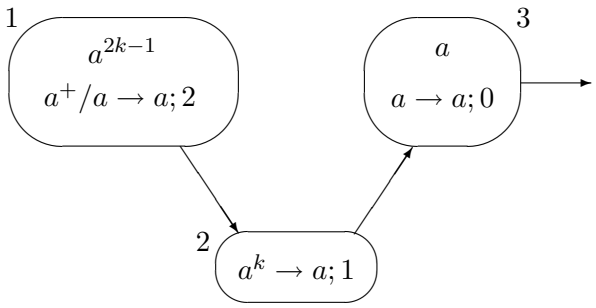

Figure 2: A simple example of an SN P system

After firing, neuron 1 will be closed/blocked for the next two steps; in the third step it will release its spike, sending it to neuron 2, and in step 3 will fire again. Thus, neuron 1 fires in every third step, consuming one of the spikes: any number $n \geq 1$ of spikes is "covered" by the regular expression a^+. In the step $3k$, neuron 2 will receive the kth spike emitted by neuron 1, hence in the next moment, $3k + 1$, it will fire. The delay between firing and spiking is of one time unit for neuron 2, hence its spike will reach neuron 3 in step $3k + 2$, meaning that neuron 3 spikes again in step $3k + 3$. Therefore, the computed number is $(3k + 3) - 1 = 3k + 2$.

The computation continues until consuming (and thus moving to neuron 2) all spikes from neuron 1, hence further $3(k - 1)$ steps. However, neurons 2 and 3 will never fire again. If we have at least one more spike in neuron 1, then neuron 2 accumulates again k spike, will fire for the second time, and the spike sent to neuron 3 will allow this neuron to spike for the third time. Such an event would make the computation unacceptable, we will get then no result.

The next example is presented in Figure 3 – we denote this SN P system by Π_2.

In the beginning, only neurons 1, 2, 3, and 7 (which is the output neuron) contain spikes, hence they fire in the first step – and spike immediately. In particular, the output neuron spikes, hence we have to count the number of steps until the next spike, to define the result of the computation.

Note that in the first step we cannot use the forgetting rule $a \to \lambda$ in neurons 1, 2, 3, because we have more than one spike present in each neuron.

The spikes of neurons 1, 2, 3 will pass to neurons 4, 5, 6. In step 2, neurons 1, 2, 3 contain no spike inside, hence will not fire, but neurons 4, 5, 6 fire. Neurons 5, 6 have only one rule, but neuron 4 behaves non-deterministically, choosing between the rules $a \to a; 0$ and $a \to a; 1$. Assume that for $m \geq 0$ steps we use here the first rule. This means that three spikes are sent to neuron 7, while each of neurons 1, 2, 3 receives two spikes. In step 3, neurons 4, 5, 6 cannot fire, but all neurons 1, 2, 3 fire again. After receiving the three spikes, neuron 7 uses its forgetting rule and gets empty again. These steps can be repeated arbitrarily many times.

In order to fire again neuron 7, we have to use sometimes the rule $a \to a; 1$ of neuron 4. Assume that this happens in step t (it is easy to see that $t = 2m + 2$). This means that at step t only neurons 5, 6 emit their spikes. Each of neurons 1, 2, 3 receives only one spike – and forgets it in the next step, $t + 1$. Neuron 7 receives two spikes, and fires again, thus sending the second spike to the environment. This happens in moment $t + 1 = 2m + 2 + 1$, hence the

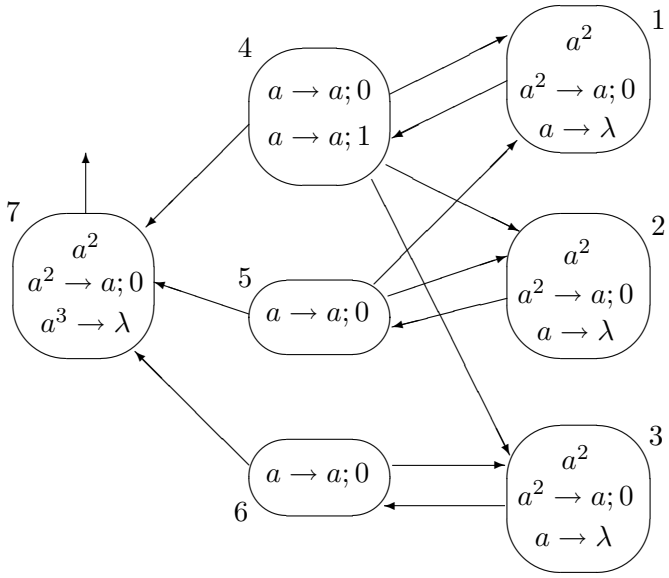

Figure 3: An SN P system generating all even natural numbers

computed number is $2m+2$ for some $m \geq 0$. The spike of neuron 4 (the one "prepared-but-not-yet-emitted" there by using the rule $a \to a; 1$ in step t) will reach neurons 1, 2, 3, and 7 in step $t+1$, hence it can be used only in step $t+2$; in step $t+2$ neurons 1, 2, 3 forget their spikes and the computation halts. The spike from neuron 7 remains unused, there is no rule for it. Note that we cannot avoid using the forgetting rules $a \to \lambda$ from neurons 1, 2, 3: without such rules, the spikes of neurons 5, 6 from step t will wait unused in neurons 1, 2, 3 and, when the spike of neuron 4 will arrive, we will have two spikes, hence the rules $a^2 \to a; 0$ from neurons 1, 2, 3 would be enabled again and the system will spike again.

Table 1 presents the computation of number 4 by the system Π_2; in each step, for each neuron we indicate in the first line the used rule and, in the second line, the spikes present in the neuron, with a dash used when no rule is applied and/or no spike is present; an exclamation mark indicates the spikes of the output neuron; the newly received spikes have a subscript which indicates their originating neuron.

We formally conclude that:

$$N_2(\Pi_2) = \{2n \mid n \geq 1\} \in Spik_2P_7(rule_2, cons_2, forg_3).$$

The next example is given both in a pictorial way, in Figure 4, and formally:

$$\begin{aligned}
\Pi_3 &= (\{a\}, \sigma_1, \sigma_2, \sigma_s, syn, 3), \text{ with} \\
\sigma_1 &= (2, \{a^2/a \to a; 0, \quad a \to \lambda\}), \\
\sigma_2 &= (1, \{a \to a; 0, \quad a \to a; 1\}), \\
\sigma_3 &= (3, \{a^3 \to a; 0, \quad a \to a; 1, \quad a^2 \to \lambda\}), \\
syn &= \{(1,2),\ (2,1),\ (1,3),\ (2,3)\}.
\end{aligned}$$

This system works as follows. All neurons can fire in the first step, with neuron 2 choosing non-deterministically between its two rules. Note that neuron 1 can fire only if it contains two spikes; one spike is consumed, the other remains available for the next step.

Table 1: A computation in the system from Figure 3

Step / Neuron	0	1	2	3	4	5	6
1		$a^2 \to a; 0$	—	$a^2 \to a; 0$	—	$a \to \lambda$	$a \to \lambda$
	aa	—	$a_4 a_5$	—	a_5	a_4	—
2		$a^2 \to a; 0$	—	$a^2 \to a; 0$	—	$a \to \lambda$	$a \to \lambda$
	aa	—	$a_4 a_5$	—	a_5	a_4	—
3		$a^2 \to a; 0$	—	$a^2 \to a; 0$	—	$a \to \lambda$	$a \to \lambda$
	aa	—	$a_4 a_6$	—	a_6	a_4	—
4		—	$a \to a; 0$	—	$a \to a; 1$	—	—
	—	a_1	—	a_1	—	—	—
5		—	$a \to a; 0$	—	$a \to a; 0$	—	—
	—	a_2	—	a_2	—	—	—
6		—	$a \to a; 0$	—	$a \to a; 0$	—	—
	—	a_3	—	a_3	—	—	—
7		$a^2 \to a; 0$!	—	$a^3 \to \lambda$	—	$a^2 \to a; 0$!	—
	aa	—	$a_4 a_5 a_6$	—	$a_5 a_6$	a_4	a

Both neurons 1 and 2 send a spike to the output neuron, 3; these two spikes are forgotten in the next step. Neurons 1 and 2 also exchange their spikes; thus, as long as neuron 2 uses the rule $a \to a; 0$, the first neuron receives one spike, thus completing the needed two spikes for firing again.

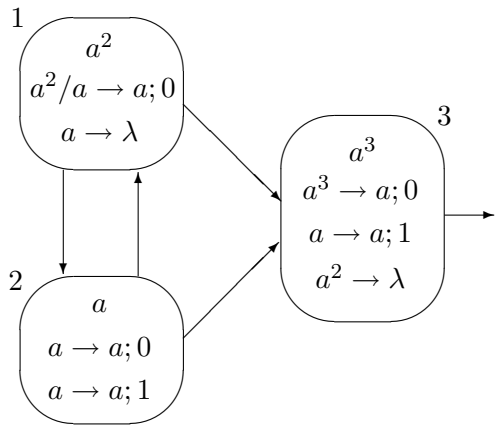

Figure 4: An SN P system generating all natural numbers greater than 1

However, at any moment, starting with the first step of the computation, neuron 2 can choose to use the rule $a \to a; 1$. On the one hand, this means that the spike of neuron 1 cannot enter neuron 2, it only goes to neuron 3; in this way, neuron 2 will never work again because it remains empty. On the other hand, in the next step neuron 1 has to use its forgetting rule $a \to \lambda$, while neuron 3 fires, using the rule $a \to a; 1$. Simultaneously, neuron 2 emits its spike, but it cannot enter neuron 3 (it is closed this moment); the spike enters neuron 1, but it is forgotten in the next step. In this way, no spike remains in the system. The computation ends

with the expelling of the spike from neuron 3. Because of the waiting moment imposed by the rule $a \to a; 1$ from neuron 3, the two spikes of this neuron cannot be consecutive, but at least two steps must exist in between.

Thus, we conclude that (remember that number 0 is ignored)

$$N_2(\Pi_3) = \mathbf{N} - \{1\} \in Spik_2P_3(rule_3, cons_3, forg_2).$$

At the price of using one more neuron, we can compute all natural numbers. Such a system is given in Figure 5 (it is denoted by Π_4 and its output neuron is 4). The task to check that $N_2(\Pi_4) = \mathbf{N}$ (again, 0 is ignored) is left to the reader.

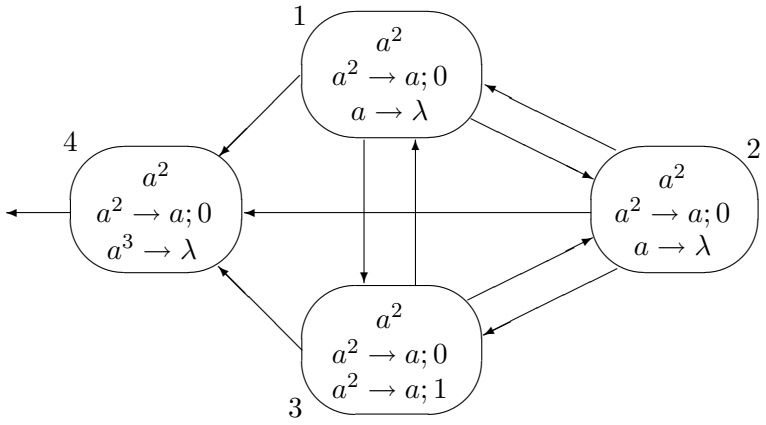

Figure 5: An SN P system generating all natural numbers

The last three examples will be useful in the next sections. Moreover, many of the proofs which follow are based on explicit constructions, hence further examples can be found in the sequel.

6 A Characterization of NFIN

In the continuation of the examples before, where an SN P system with three neurons was presented which computes an infinite set of numbers, let us examine the power of systems with one or two neurons only.

We start with the observation that by using any type of rules, spiking or forgetting rules, in a neuron σ_i we diminish the number of spikes from σ_i. In general, the number of spikes can increase in the whole system, because the spikes are replicated in the case of multiple synapses starting from a given neuron. However, if we have only one neuron in the system, then no replication is possible, hence each computation lasts at most as many steps as the number of spikes initially present in the neuron.

Consequently, SN P systems with only one neuron can only compute finite sets of numbers. Interesting enough, *any finite set can be computed by a one-neuron system.* Indeed, let us take a finite set of numbers, $F = \{n_1, n_2, \ldots, n_k\}$, and construct the system Π_F with only one neuron, containing initially two spikes, and the following rules:

$$a^2a \to a; 0,$$
$$a \to a; n_i, \text{ for each } i = 1, 2, \ldots, k.$$

The system spikes in the first step by means of the rule $a^2/a \to a; 0$, then, because one spike remains inside, it fires again the next step, using any of the rules $a \to a; n_i$. This means that the next spike is sent out in step $n_i + 1$, hence $N_2(\Pi_F) = F$.

What about systems consisting of two neurons? One of them should be the output one, and the output neuron can spike only twice. This means that this neuron can increase the number of spikes in the system at most with two. The other neuron does not have a synapse to itself, hence, like in the case of single-neurons systems, it cannot use more rules than the number of spikes initially present in it, maybe plus two, those possibly received from the output neuron. This means that all computations are of a bounded length, hence the set of numbers computed by the system is again finite.

We synthesize these observations in the form of a theorem, mainly in view of the third example from the previous section: this is the best result of this type (in what concerns the number of neurons):

Theorem 6.1 $NFIN = Spik_2P_1(rule_*, cons_1, forg_0) = Spik_2P_2(rule_*, cons_*, forg_*) = Spik_2P_2(rule_*, cons_*, forg_*)$.

From these very small systems, let us now jump to the most general case, without any restriction on the number of neurons, or on other parameters.

7 Computational Completeness

The next inclusions follow directly from the definitions, with the inclusion in NRE provable in a straightforward manner (or, we can invoke for it the Turing-Church thesis):

Lemma 7.1 $Spik_2P_m(rule_k, cons_p, forg_q) \subseteq Spik_2P_{m'}(rule_{k'}, cons_{p'}, forg_{q'}) \subseteq Spik_2P_*(rule_*, cons_*, forg_*) \subseteq NRE$, *for all* $m' \geq m \geq 1$, $k' \geq k \geq 1$, $p' \geq p \geq 1$, $q' \geq q \geq 0$.

Surprisingly enough, taking into account the restrictive form of our systems, the spiking neural P systems prove to be computationally universal:

Theorem 7.1 $Spik_2P_*(rule_k, cons_p, forg_q) = NRE$ *for all* $k \geq 2, p \geq 3, q \geq 3$.

Proof. In view of Lemma 7.1, we only have to prove the inclusion $NRE \subseteq Spik_2P_*(rule_2, cons_3, forg_3)$. To this aim, we use the characterization of NRE by means of register machines.

Let $M = (m, H, l_0, l_h, I)$ be a register machine, having the properties specified in Section 3: the result of a computation is the number from register 1 and this register is never decremented during the computation.

We construct a spiking neural P system Π as follows.

Instead of specifying all technical (and difficult to follow) details of the construction, we present the three main *types of modules* of the system Π, with the neurons, their rules, and their synapses represented graphically. All neurons are initially empty, with the single exception of the neuron with label l_0 (the label of the initial instruction of M), which contains exactly two spikes (two copies of a).

What we want to do is to have Π constructed in such a way (1) to simulate the register machine M, and (2) to have its output neuron spiking only twice, at an interval of time which corresponds to a number computed by M.

In turn, simulating M means to simulate the ADD instructions and the SUB instructions. Thus, we will have a type of modules associated with ADD instructions, one associated with

SUB instructions, and one dealing with the spiking of the output neuron (a FIN module). The modules of the three types are given in Figures 6, 7, 8, respectively. The neurons appearing in these figures have labels l_1, l_2, l_3, as in the instructions from I, labels $1, 2, \ldots, m$ associated with the registers of M, l_1', l_1'', l_1''' associated with the label l_1 identifying ADD and SUB instructions from I, b_r (from "before r", because this neuron is used by all ADD instructions when sending a spike to neuron r), $f_1, f_2, f_3, f_4, f_5, f_6$, used by the FIN module, as well as *out*, labeling the output neuron.

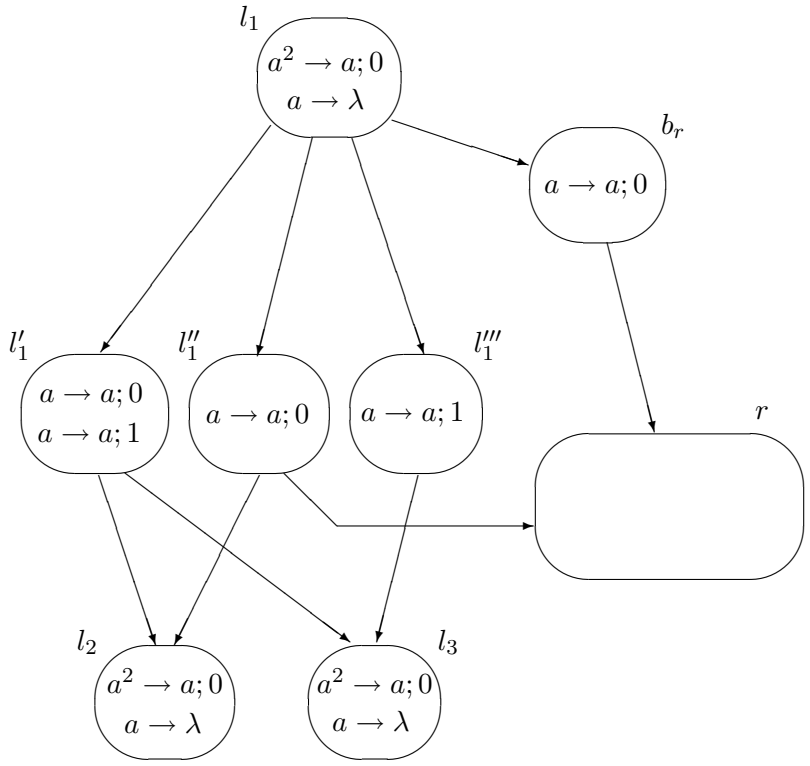

Figure 6: Module ADD (simulating $l_1 : (\texttt{ADD}(r), l_2, l_3)$)

Before describing the work of these modules, let us remember that the labels are injectively associated with the instructions of M, hence each label precisely identifies one instruction, either an ADD or a SUB one, with the halting label having a special situation – it will be dealt with by the FIN module.

Simulating an ADD instruction $l_1 : (\texttt{ADD}(r), l_2, l_3)$ – module ADD (Figure 6).

The initial instruction, that labeled with l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction $l_1 : (\texttt{ADD}(r), l_2, l_3)$, with two spikes present in neuron l_1 (like in the initial configuration) and no spike in any other neuron, except those neurons associated with the registers. Having two spikes inside, neuron l_1 gets fired. Its spike will simultaneously go to four neurons, l_1', l_1'', l_1''', and b_r.

In the next step, both neurons l_1'' and b_r will send a spike to neuron r, the one which corresponds to register r of M. In this way, the contents of neuron r increases by two. In each moment, if register r has value n, then neuron r will contain $2n$ spikes. In Figure 6, neuron r contains no rules, but, as we will see immediately, the neurons associated with registers have two rules each, used when simulating the SUB instructions, but both these rules need an odd

number of spikes to be applied (this is true also for the module FIN, which only deals with the neuron associated with register 1). Therefore, during the simulation of an ADD instruction, neuron r just increases by 2 its contents, and never fires.

Now, the problem is to pass non-deterministically to one of the instructions with labels l_2 and l_3, that is, in our system we have to ensure the firing of neurons l_2 or l_3, non-deterministically choosing one of them. To this aim, we use the non-determinism of the rules in neuron l_1'. One of them will be used, thus consuming the unique spike existing here. If we use the rule $a \to a; 0$, then both neurons l_2, l_3 receive a spike from l_1'. For l_3 this is the unique spike it receives now (note that neuron l_1''' fires now but its spike will leave one step later), hence in the next step the forgetting rule of neuron l_3 should be used. Instead, neuron l_2 receives two spikes, one from neuron l_1' and one from neuron l_1'', hence in the next step it is fired.

However, if instead of rule $a \to a; 0$ we use the rule $a \to a; 1$ of neuron l_1' (note that the only difference is the time of spiking), then it is l_2 which receives only one spike, and immediately it "forgets" it, while in the next step neuron l_3 receives two spikes, and fires.

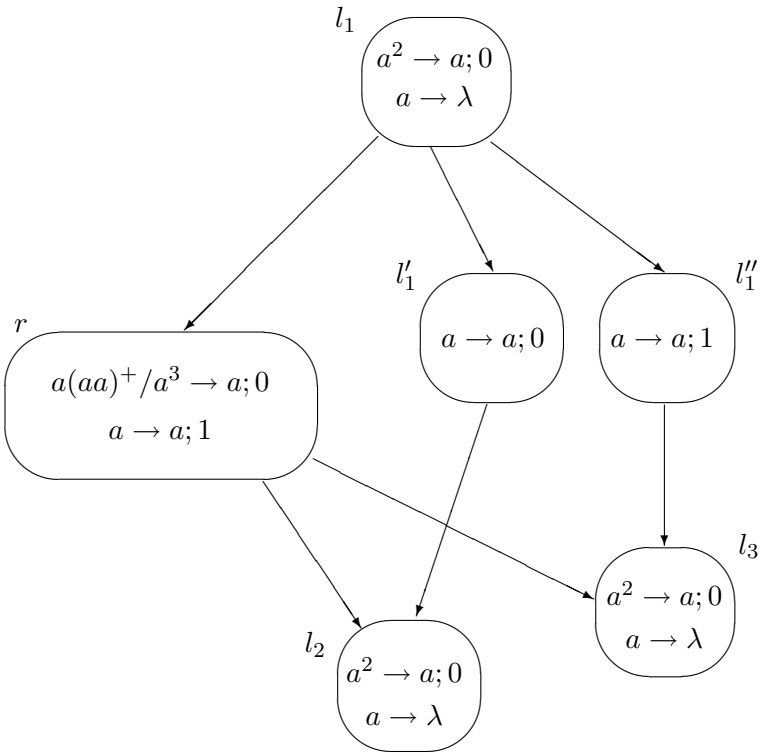

Figure 7: Module SUB (simulating $l_1 : (\texttt{SUB}(r), l_2, l_3)$)

Therefore, from firing neuron l_1, we pass to firing non-deterministically one of neurons l_2, l_3, while also increasing by 2 the number of spikes from neuron r.

Simulating a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3)$ – module SUB (Figure 7).

Let us examine now Figure 7, starting from the situation of having two spikes in neuron l_1 and no spike in other neurons, except neuron r, which holds an even number of spikes (half of this number is the value of the corresponding register r). The spike of neuron l_1 goes immediately to three neurons, l_1', l_1'', and r. Neuron l_1' will send in the next step a spike to neuron l_2, while neuron l_1'' will send a spike to neuron l_3 one step later.

A similar situation appears in neuron r: because it contains now an odd number of spikes, it gets fired, and it will either spike immediately, if the first rule is used ($(a(aa)^+/a^3 \to a; 0)$, or one step later, if the second rule is used ($a \to a; 1$). In turn, the first rule is used if and only if neuron r contains at least three spikes; one spike has just came from neuron l_1, hence at least two existed in the neuron, which means that the value of register r was at least one. The two spikes which reach neuron l_2 make this neuron fire, as requested by simulating the SUB instruction. The spike from neuron l_3 will be removed by the local forgetting rule. If in neuron r there is only one spike (this corresponds to the case when register r is empty), then the second rule is used, hence the neuron spikes at the same time with l_1'', in the next step. This means that neuron l_2 receives only one spike (and removes it), while neuron l_3 receives two, and fires.

The simulation of the SUB instruction is correct, we started from l_1 and we ended in l_2 if the register was non-empty and decreased by one, and in l_3 if the register was empty.

Note that there is no interference between the neurons used in the ADD and the SUB instructions, other than correctly firing the neurons l_2, l_3 which may label instructions of the other kind. In particular, the ADD instructions do not use any rule for handling the spikes of neurons $1, 2, \ldots, m$. The only neurons used by several rules are those which correspond to registers and neuron b_r, used as a sort of interface before sending a spike to neuron r; then, each neuron r associated with a register which is subject of a SUB instruction sends a spike to several, possibly to all, neurons with labels $l \in H$ – but only one of these neurons also receives at the same time a spike from the corresponding neurons l_1', l_2'', hence only the correct neuron fires, all others forget immediately the unique spike.

Ending a computation – module FIN (Figure 8).

Assume now that the computation in M halts, which means that the halting instruction is reached. For Π this means that the neuron l_h gets two spikes and fires. At that moment, neuron 1 contains $2n$ spikes, for n being the contents of register 1 of M. The spike of neuron l_h reaches immediately neurons 1, f_1, f_5. This means that neuron 1 contains now an odd number of spikes, and it can fire. It is important to remember that this neuron was never involved in a SUB instruction, hence it does not contain any rule as those from Figure 7.

Let t be the moment when neuron l_h fires.

At moment $t+1$, neurons f_1, f_5, and 1 fire and all of them spike immediately. The spiking of neuron 1 means subtracting one from the value of the associated register: by using the rule $a^3(aa)^+/a^2 \to a; 0$, two copies of a are consumed (hence the number of spikes remains odd, the rule can be used again next step). The spike of neuron 1 goes to four neurons, f_3, f_4, f_5, and f_6. This means that in step $t+1$ neuron f_6 receives three spikes (from neurons f_1, f_5, and 1), which are immediately forgotten.

In step $t+2$, neurons f_2, f_3, f_5, and 1 fire and spike; again neuron f_6 receives three spikes, which are forgotten immediately. The spike from neuron f_2 reaches the output neuron, *out*, which in step $t+3$ will fire and spike. This is the first spike of the output neuron. The number of steps from this spike to the next one is the number computed by the system.

Let us see how the neurons f_3, f_4 interplay: in each step, each of them receives a spike from neuron 1. We start with neuron f_3 fired first; it sends a spike to neuron f_6 and one to the companion neuron f_4. This means that in the next step neuron f_4 is fired (he has received one spike from 1 and one from f_3), while neuron f_3, having only the spike from neuron 1, removes it. In the next step the roles of neurons f_3, f_4 are interchanged. This means that in each step one of them fires and sends a spike to the other (and one to neuron f_6), while the other only forgets one spike.

Then, let us observe that neuron 1 sends in each moment two spikes towards neuron f_6, one reaching immediately the target, the other one with one step delay, because it passes through

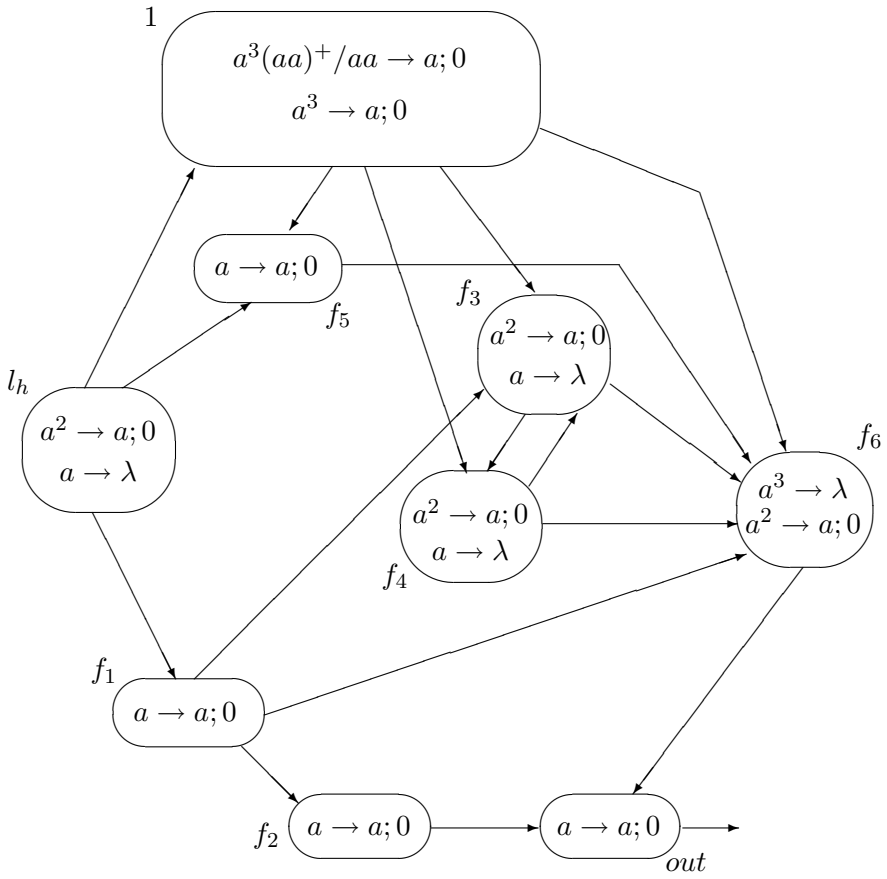

Figure 8: Module FIN (ending the computation)

neuron f_5. In turn, neuron f_5 never contains more than one spike, because the spike of neuron l_h reaches it in the first step and those from neuron 1 in the subsequent steps.

This means that the process of removing two spikes from neuron 1 continues, iteratively, without having neuron f_6 spiking, until using the rule $a^3 \to a;0$. This is the last time when neuron 1 fires, hence this is step $t+n$ (for $2n$ being the initial contents of neuron 1). In the next step, $t+n+1$, one of neurons f_3, f_4 still fires, because it has two spikes, and the same with neuron f_5. No other neuron fires in this step. This means that neuron f_6 receives only two spikes, and this makes it spike in the next step, $t+n+2$. This is the only spiking in this step, all other neurons are empty.

In step $t+n+3$ also the output neuron spikes, and this ends the computation, the system contains no spike.

The interval between the two spikes of neuron *out* is $(t+n+3)-(t+3) = n$, exactly the value of register 1 of M in the moment when its computation halts. Consequently, $N_2(\Pi) = N(M)$ and this completes the proof. □

The previous construction contains only a few neurons which do not spike immediately, but their role in the correct functioning of the system is essential. Also the forgetting rules are crucial in this proof. It is an interesting *open problem* which is the power of spiking neural P systems without forgetting rules; are they still universal? Another (standard) open problem is whether or not the parameters used in the theorem are optimal, or they can be improved. Of

course, systems with only one rule in each neuron are deterministic, hence for such systems we have to look for other types of results of computations, e.g., the infinite sequence of bits marking the spiking of the output neuron. However, in what concerns the number of neurons behaving non-deterministically, a spectacular (especially by its metaphoric interpretation) result can be obtained:

Corollary 7.1 *Each set from NRE can be computed by an SN P system* Π *having only one neuron with rules which can be used non-deterministically.*

Proof. It is enough to observe that we have non-deterministic neurons only in the module ADD – this is the case with neuron l_1'. Let us "unify" all these neurons for all ADD instructions, in the form of a neuron l_{ndet} containing the rules

$$a \to a; 0, \quad a \to a; 1,$$

with synapses (l_1, l_{ndet}) for all instructions $l_1 : (\texttt{ADD}(r), l_2, l_3)$ in the register machine we want to simulate, and (l_{ndet}, l) for all $l \in H$. When starting to simulate any instruction $l_1 : (\texttt{ADD}(r), l_2, l_3)$, neuron l_{ndet} receives a spike, fires, and either spikes immediately, or in the next step. The spike is sent to all neurons $l \in H$, but it meets another spike only in one neuron: l_2 if l_{ndet} spikes immediately and l_3 if it spikes in the next step. In all neurons different from these neurons, the spike is forgotten. The system obtained in this way is clearly equivalent with the one constructed in the proof of Theorem 7.1 and it contains only one non-deterministic neuron. □

This result can have a nice interpretation: it is sufficient for a "brain" (in the form of an SN P system) to have only one neuron which behaves non-deterministically in order to achieve "complete (Turing) creativity".

Now, in what concerns the number of spikes consumed for firing and the number of forgotten spikes, the problem remains whether or not the value 3 from the theorem can be decreased. Finally, it would be interesting to have universality results for systems with a bounded number of neurons, maybe paying in other parameters, such as the number of rules in neurons.

The previous construction can easily be modified – actually, simplified – in order to obtain an universality proof for the case where the result of a computation is defined as the number of spikes present in the output neuron in the last configuration of a halting computation (in this case we cannot avoid imposing the halting condition, because we have to make sure that no further spike will be added to the output): we just replace the module FIN with the module from Figure 9.

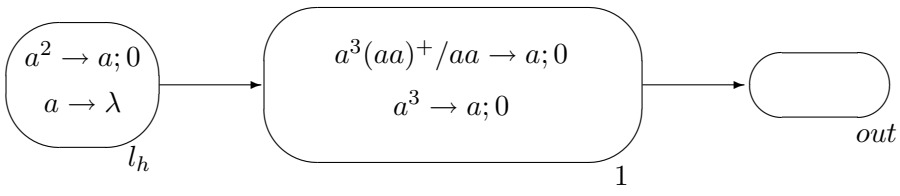

Figure 9: Module FIN in the case of counting the number of the spikes

When the halting instruction of the register machine M is reached, the neuron labeled l_h sends a spike to neuron 1, which has now $2n + 1$ spikes inside, and can start to send spikes to neuron *out*. Exactly as in the proof of Theorem 7.1, the spiking of neuron 1 corresponds to

decreasing by one the first register of M, hence the number of spikes accumulated in neuron *out* is equal to the value of register 1 in the end of a computation of M.

If we add the rule $a \to a; 0$ to neuron *out*, then in each step, after receiving a spike from neuron 1, neuron *out* spikes, hence in this way we obtain the result in the environment.

8 Spiking Neural P Systems Working in the Accepting Mode

An SN P system can be also used in the accepting mode. We consider here the following way of introducing the number to be accepted, again in the spirit of spiking neurons, with the time as the main data support: the special neuron i_0 is used now as an input neuron, which can receive spikes from the environment of the system (in the graphical representation an incoming arrow will indicate the input neuron); we assume that exactly two spikes are entering the system; the number n of steps elapsed between the two spikes is the one analyzed; if, after receiving the two spikes, the system halts (not necessarily in the moment of receiving the second spike), then the number n is accepted. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by a system Π, and by $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ the family of sets of this form corresponding to the family $Spik_2P_m(rule_k, cons_p, forg_q)$.

In the accepting mode we can impose the restriction that in each neuron, in each time unit at most one rule can be applied, hence that the system behaves deterministically. When considering only deterministic SN P systems, the notation $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ will get a letter "D" in front of it.

Of course, inclusions as those in Lemma 7.1 are valid both for deterministic and non-deterministic accepting SN P systems, and we also have the inclusion $DSpik_{acc}P_m(rule_k, cons_p, forg_q) \subseteq Spik_{acc}P_m(rule_k, cons_p, forg_q)$, for all m, k, p, q (numbers or $*$). A counterpart of Theorem 7.1 is also true:

Theorem 8.1 *$DSpik_{acc}P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 2$.*

Proof. The proof is a direct consequence of the proof of Theorem 7.1. Namely, we start from a deterministic register machine M and we construct the SN P system Π as in the proof of Theorem 7.1 – with changes which will be immediately mentioned –, as well as a further module, called INPUT, which takes care of initializing the work of Π. This time Π has no object inside, and the same is true with the new module.

The module INPUT is indicated in Figure 10. Its functioning is rather clear. Because the system contains no spike inside, no rule can be applied. When a spike enters neuron *in*, this neuron sends a spike to all neighbors c_1, c_2, c_3, c_4 (they are new neurons). Neurons c_1, c_2 do nothing, they just wait for a second spike. Neurons c_3, c_4 spike immediately, sending together two spikes to neuron 1 (it corresponds to the first register of M), as well as to each other. This means that in each step each of c_3, c_4 fires, hence in each step the contents of neuron 1 increases again with two spikes. If at some moment, at n steps after the first spike coming to the system, a second spike enters neuron *in* coming from the environment, then this neuron spikes again. For neurons c_1, c_2 this entails the firing, and thus neuron l_0 gets the necessary spikes to fire, while for neurons c_3, c_4 this means the end of work, because the spikes are erased by the rules $a^2 \to \lambda$. Therefore, on the one hand, the contents of neuron 1 remains $2n$, on the other hand, neuron l_0 triggers the simulation of a computation in M, starting with the instruction labeled with l_0, in recognizing the number n.

Note that this is consistent with the way the system Π from the proof of Theorem 7.1 works: the neuron l_0 starts by having two spikes inside, and the contents of register 1 is double the number to analyze.

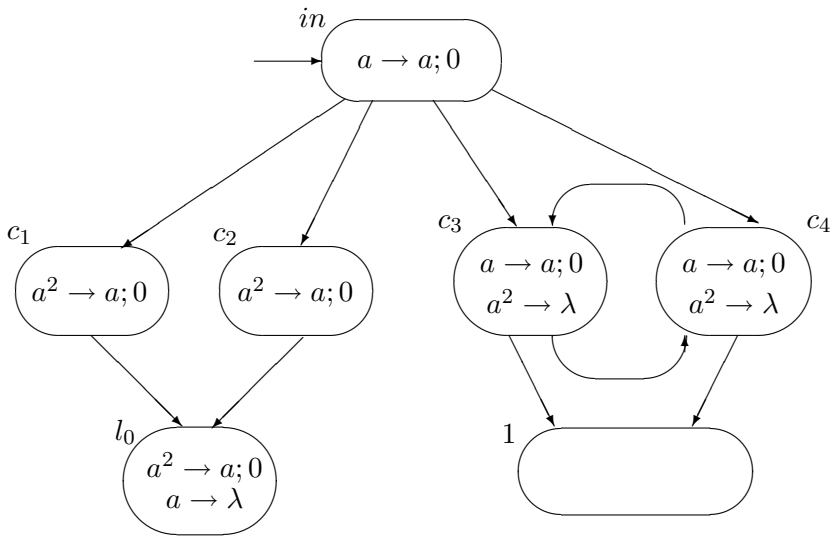

Figure 10: Module INPUT (initiating the computation)

Now, we start using modules ADD and SUB associated with the register machine M, with modules ADD constructed for instructions of the form $l_1 : (\texttt{ADD}(r), l_2)$. This means that the module ADD is now much simpler than in Figure 6, namely, it looks like in Figure 11.

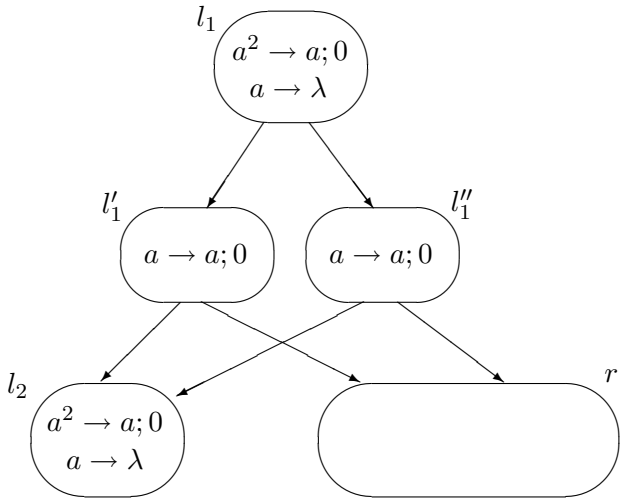

Figure 11: Module ADD in the deterministic case

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

The modules SUB remain unchanged, while the module FIN is simply removed, with the neuron l_h remaining in the system, with no rule inside. Thus, the computation will stop if and only if the computation in M stops. The observation that the only forgetting rule $a^s \to \lambda$ with $s = 3$ was present in module FIN, which is no longer used, completes the proof. □

It is worth noting that both in this section and in the previous one, we use only firing rules with immediate spiking and with spiking at the next step. Furthermore, the regular expressions

we have used (in the SUB and FIN modules) are only meant to check the parity of the number of spikes present in the neuron. Both these parameters – the maximal delay in spiking and the complexity of the regular expressions used in rules – can be considered as complexity parameters of our systems (and the results above show that universality can be obtained even for rather simple systems from these points of view).

9 A Characterization of Semilinear Sets of Numbers

Let us now try to be more "realistic", not allowing the neurons to hold arbitrarily many spikes at the same time. This reminds of the sigmoidal function relating the input excitations to the neuron exciting: after a given threshold, the additional spikes do not matter, they are simply ignored.

Not very surprising (this reminds other characterizations of semilinear sets of numbers of or vectors of numbers in terms of P systems – see, e.g., [3]), but interesting by the proof, the SN P systems which have a bound on the number of spikes they hold during a computation generate exactly semilinear sets of natural numbers.

Let us denote by $Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$ the family of sets of numbers $N_2(\Pi)$ computed by SN P systems Π with at most m cells, using at most k rules in each cell, consuming at most p and forgetting at most q spikes in each rule, and with at most s spikes present at any time in any neuron (if a computation reaches a configuration where a neuron accumulates more than s spikes, then it aborts, such a computation does not provide any result). As usual, we replace by $*$ the parameters which are not bounded ($bound_*$ will mean that we consider only SN P systems with a bound on the number of spikes present in any neuron, but this bound is not specified; when $bound_\alpha$ is simply missing from the notation, this will mean that the number of spikes in neurons can grow arbitrarily, beyond any limit – like in the previous sections).

In this section we prove the following theorem:

Theorem 9.1 *$SLIN_1 = Spik_2P_*(rule_k, cons_p, forg_q, bound_s)$, for all $k \geq 3, q \geq 3, p \geq 3$, and $s \geq 3$.*

We start with the inclusion which is simpler to prove:

Lemma 9.1 *$Spik_2P_*(rule_*, cons_*, forg_*, bound_*) \subseteq SLIN_1$.*

Proof. Take a system Π with a bound s on the number of spikes in each neuron. The number of neurons is given, their contents is bounded, the number of rules in neurons (hence the length of the refractory periods) is given, hence the number of configurations reached by Π is finite. Let $\mathcal{C}$ be their set, and let C_0 be the initial configuration of Π.

We construct the right-linear grammar $G = (N, \{a\}, (C_0, 0), P)$, where $N = \mathcal{C} \times \{0, 1, 2\}$, and P contains the following rules:

1. $(C, 0) \to (C', 0)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron does not spike;
2. $(C, 0) \to (C', 1)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron spikes;
3. $(C, 1) \to a(C', 1)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron does not spike;

4. $(C,1) \to a(C',2)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron spikes;

5. $(C,2) \to \lambda$, for $C \in \mathcal{C}$ if there is a halting computation $C \Longrightarrow^* C'$ in Π during which the output neuron never spikes, or there is an infinite computation starting in configuration C during which the output neuron of Π never spikes.

The way of controlling the derivation by the two components of the nonterminals in N ensures the fact that $N_2(\Pi)$ is the length set of the regular language $L(G)$, hence it is semilinear.

It is worth noting that the construction of grammar G is effective, because the conditions involved in defining the rules – including those from step 5 – can be decided algorithmically. □

The opposite inclusion is based on the observation that any semilinear set of numbers is the union of a finite set with a finite number of arithmetical progressions. Now, a finite set is the union of a finite number of singleton sets. Thus, it suffices to prove the closure under union and the fact that singleton sets and arithmetical progressions are in $Spik_2P_*(rule_3, cons_3, forg_3, bound_3)$, and we will do this in the following series of lemmas, whose conjunction – together with Lemma 9.1 – proves the theorem.

Lemma 9.2 *Each singleton $\{n\}, n \geq 1$, is in $Spik_2P_1(rule_2, cons_1, forg_0, bound_2)$.*

Proof. Take the system with only one neuron, containing initially two spikes, and two rules:

$$a^2/a \to a; 0, \qquad a \to a; n-1.$$

The first spike exits in step 1, the second one in step $2 + (n-1)$, hence the computed number is n. □

Lemma 9.3 *Each arithmetical progression $\{ni \mid i \geq 1\}$, $n \geq 3$, is in $Spik_2P_{n+2}(rule_3, cons_3, forg_2, bound_3)$.*

Proof. For given n as in the statement of the lemma, we consider the SN P system in Figure 12.

The neuron *out* spikes in step 1. The spike emitted by it goes along the path $1, 2, \ldots, n-2$ until getting doubled when passing from neuron $n-2$ to neurons $n-1$ and 0. Both these last neurons get fired. As long as neurons 0 and $n-1$ spike in different moments (because neuron 0 uses the rule $a \to a; 1$), no further spike exits the system (neuron *out* gets only one spike and forgets it immediately), and one passes along the cycle of neurons $1, 2, \ldots, n-1, n$ again and again. If neurons 0 and $n-1$ spike at the same time (neuron 0 uses the rule $a \to a; 0$), then the system spikes again – hence in a moment of the form $ni, i \geq 1$. The spike of neuron *out* arrives at the same time in neuron 1 with the spike of neuron n, and this halts the computation, because of the rule $a^2 \to \lambda$, which consumes the spikes present in the system. Consequently, the system computes the "pure" arithmetical progression $\{ni \mid i \geq 1\}$. □

However, we have to compute not only "pure" progressions, but also of the form $\{r + ni \mid i \geq 1\}$, for $n \geq 1$ and $r \geq 1$. This is ensured by the following general result:

Lemma 9.4 *If $Q \in Spik_2P_m(rule_k, cons_q, forg_p, bound_s)$ and $r \geq 1$, then $\{x + r \mid x \in Q\} \in Spik_2P_{m+1}(rule_k, cons_p, forg_q, bound_s)$, for all $m \geq 1, k \geq 2, p \geq 3, q \geq 0, s \geq 3$.*

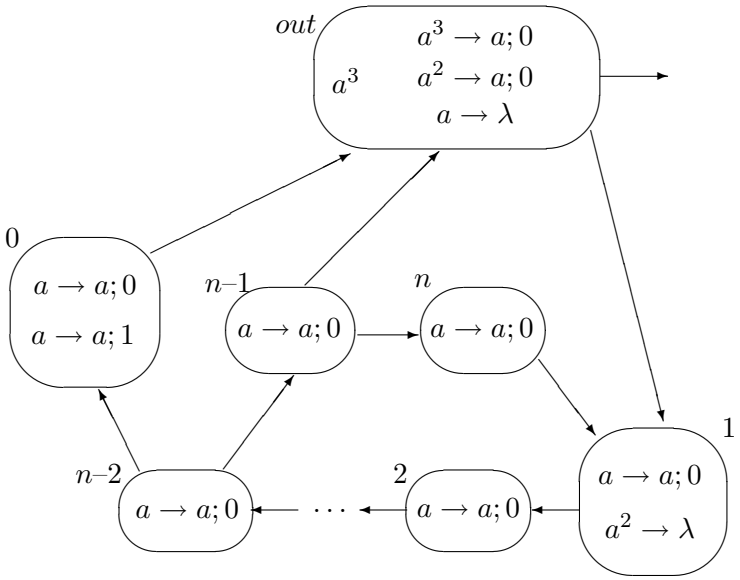

Figure 12: An SN P system generating an arithmetical progression

Proof. Having a system Π, generating a given set Q, we add to this system a further neuron, labeled with out', with a synapse from the output neuron of Π to this new neuron; the new neuron has initially a^2 inside, and the rules

$$a^3 \to a; 0, \qquad a \to a; r.$$

Let Π' be the system obtained in this way, with the output neuron out'. If the system Π spikes at moments t_1 and t_2, then Π' spikes at the moments $t_1 + 1$ and $t_2 + 1 + r$, hence if Π computes the number $t_2 - t_1$, then Π' computes the number $(t_2 + 1 + r) - (t_1 + 1) = (t_2 - t_1) + r$. □

We still have to consider two particular progressions, the one with step 2 and the one with step 1. The former case was already covered by the second example from Section 5 (Figure 3). The latter is the set $\mathbf{N}$, which, from the fourth example from Section 5 (Figure 5) is know to be in $Spik_2P_4(rule_2, cons_2, forg_3, bound_2)$.

For proving the closure under union we need an auxiliary result. For an SN P system Π, let us denote by $spin(\Pi)$ the maximal number of spikes present in a neuron in the initial configuration of Π.

Lemma 9.5 *For every SN P system Π (not necessarily with a bound on the number of spikes present in its neurons), there is an equivalent system Π' such that $spin(\Pi') = 1$. The system Π' has $spin(\Pi) + 1$ additional neurons, all of them containing only one rule, of the form $a \to a; 0$.*

Proof. Take an arbitrary SN P system Π and construct a system Π' as follows. We consider further $spin(\Pi) + 1$ neurons, with labels $0, 1, 2, \ldots, spin(\Pi)$ (we assume that these labels are not used also in Π). Neuron 0 is the only one in the whole system which contains any spike, namely one (we remove all spikes from the neurons of system Π). The old neurons have the same rules as in Π, while each new neuron contains only the rule $a \to a; 0$. From neuron 0 start synapses to all neurons $1, 2, \ldots, spin(\Pi)$. From these neurons we establish as many synapses to the neurons

of Π as many spikes they have in the initial configuration of Π (precisely, if a neuron l of Π has n_l spikes in the initial configuration of Π, then we establish the synapses (i, l), for all $1 \leq i \leq n_l$). Thus, after two steps, all neurons of Π' which correspond to neurons of Π contain exactly as many spikes as they contained in the initial configuration of Π. No synapse goes back to the new neurons, hence from now on the system works exactly as Π, that is, $N_2(\Pi') = N_2(\Pi)$. □

We can now complete the proof of Theorem 9.1, by proving the "union lemma":

Lemma 9.6 *If $Q_1, Q_2 \in Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$, for some $m \geq 1, k \geq 2$, $p \geq 2$, $q \geq 1$, $s \geq 2$, then $Q_1 \cup Q_2 \in Spik_2P_{2m+6}(rule_k, cons_p, forg_q, bound_s)$.*

Proof. Take two SN P systems Π_1, Π_2 in the normal form given by Lemma 9.5. Let in_1, in_2 be the labels of neurons in Π_1, Π_2, respectively, containing initially one spike. We construct a system Π as suggested in Figure 13.

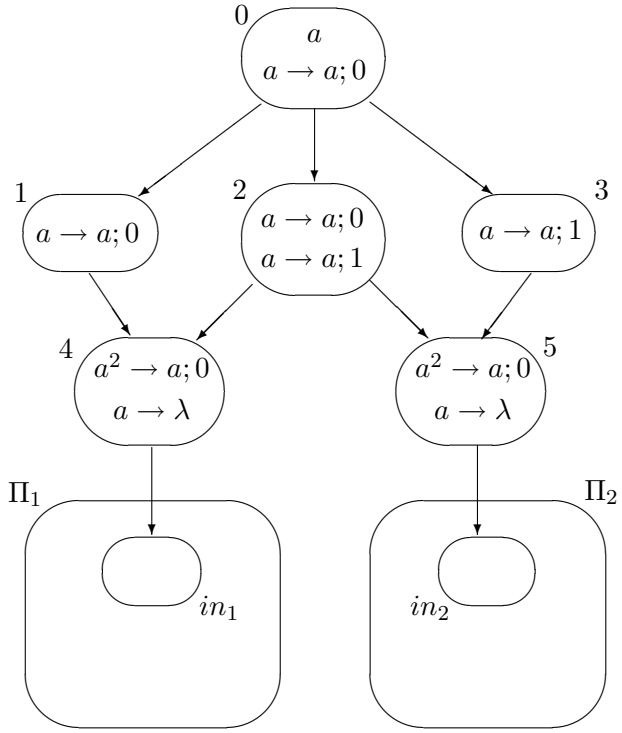

Figure 13: The idea of the union construction

In the system Π constructed in this way only neuron 0 contains a spike (those from neurons in_1, in_2 were removed. The non-deterministic functioning of neuron 2 will enable either the (sub)system Π_1, or the (sub)system Π_2, hence Π can compute whatever any of the two systems can compute. □

Note that also this lemma holds true for arbitrary systems, not only for the bounded ones.

Theorem 9.1 can be interpreted in the following way: even if we use time as a support for information, without using the classic workspace as a resource and without considering information also encoded in the number of objects used in the system, we cannot compute "too

much" – precisely, we cannot go beyond the semilinear sets (the length sets of regular languages).

From Theorem 9.1, all closure properties of the family $SLIN_1$ are transferred to the family $Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$; for instance, we get the closure under sum, intersection, complement, and the non-closure under product. Still more precise results also follow from Lemmas 9.6 and 9.4, concerning the closure under union and the sum with a given number. A similar situation is meet with respect to the non-closure under product:

Corollary 9.1 *There are sets Q_1, Q_2 in the family $Spik_2P_3(rule_3, cons_3, forg_2, bound_3)$ such that $Q_1Q_2 = \{nm \mid n \in Q_1, m \in Q_2\}$ is not in $Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$.*

Proof. We take $Q_1 = Q_2 = \{2, 3, \ldots\}$. From the third example in Section 5 we know that $Q_1 \in Spik_2P_3(rule_3, cons_3, forg_2, bound_3)$. Because Q_1Q_2 is the set of all composite numbers, and this set is not semilinear, from Theorem 9.1 it follows that $Q_1Q_2 \notin Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$. □

We end this section with the following **general remark**: all SN P systems considered in the constructions from the proofs in this section and from Section 7 always halt (maybe a few steps) after the second spiking. Thus, all results above are valid also in the restrictive case of considering successful only halting computations (which spike exactly twice).

10 Closing Remarks

Starting from the definition of neural-like P systems and following the idea of spiking neurons from neurobiology, we have proposed a class of spiking neural P systems for which we have proved the universality in the general case and a characterization of semilinear sets of numbers in the case when the number of spikes is bounded.

This work is the first one in this area, and many research directions remain to be explored (several of them already mentioned in the previous sections), starting with considering further ideas from neurobiology. Inhibiting/de-inhibiting mechanisms, dynamic structures of synapses, changing the neurons themselves during their "lives" (maybe "damaging" ones and letting other neurons to replace them), assigning a more important role to the environment, learning and adapting to new inputs from the environment, and so on and so forth, can be issues to investigate.

Then, returning to the theoretical framework, the similarity with Petri nets is visible (both models move "tokens" across a network), and equally the differences (there is no delay in spiking, forgetting rules, counting states in Petri nets, neither universality for the basic classes of nets); each domain has probably to import ideas and results from the other one.

We conclude the paper with a technical open problem. In Section 6 we have seen that two neurons can compute at most finite sets. How many neurons – with unbounded contents – suffice in order to compute a non-semilinear set of numbers?

Figure 14 gives a first answer, using 18 neurons. We start with 5 spikes in neuron 1, one in neuron 4, and two spikes in each neuron 16 and 17. Iteratively, the $2n + 1$ spikes from neuron 1 are moved to neuron 7, doubling all but the last one spike, hence getting $4n + 1$ spikes in neuron 7. The process is somewhat similar to that carried by the modules FIN and SUB from the proof of Theorem 7.1 ($2n$ is meant to represent the number n, which thus is doubled when passing from neuron 1 to neuron 7). From neuron 7, all spikes are then moved back to neuron 1. After each cycle, neurons 6 and 10 spike. As long as the rule $a \to a; 0$ is used in neuron 13, the spike of neuron 10 is forgotten on the way towards the output neuron. Similarly, the two

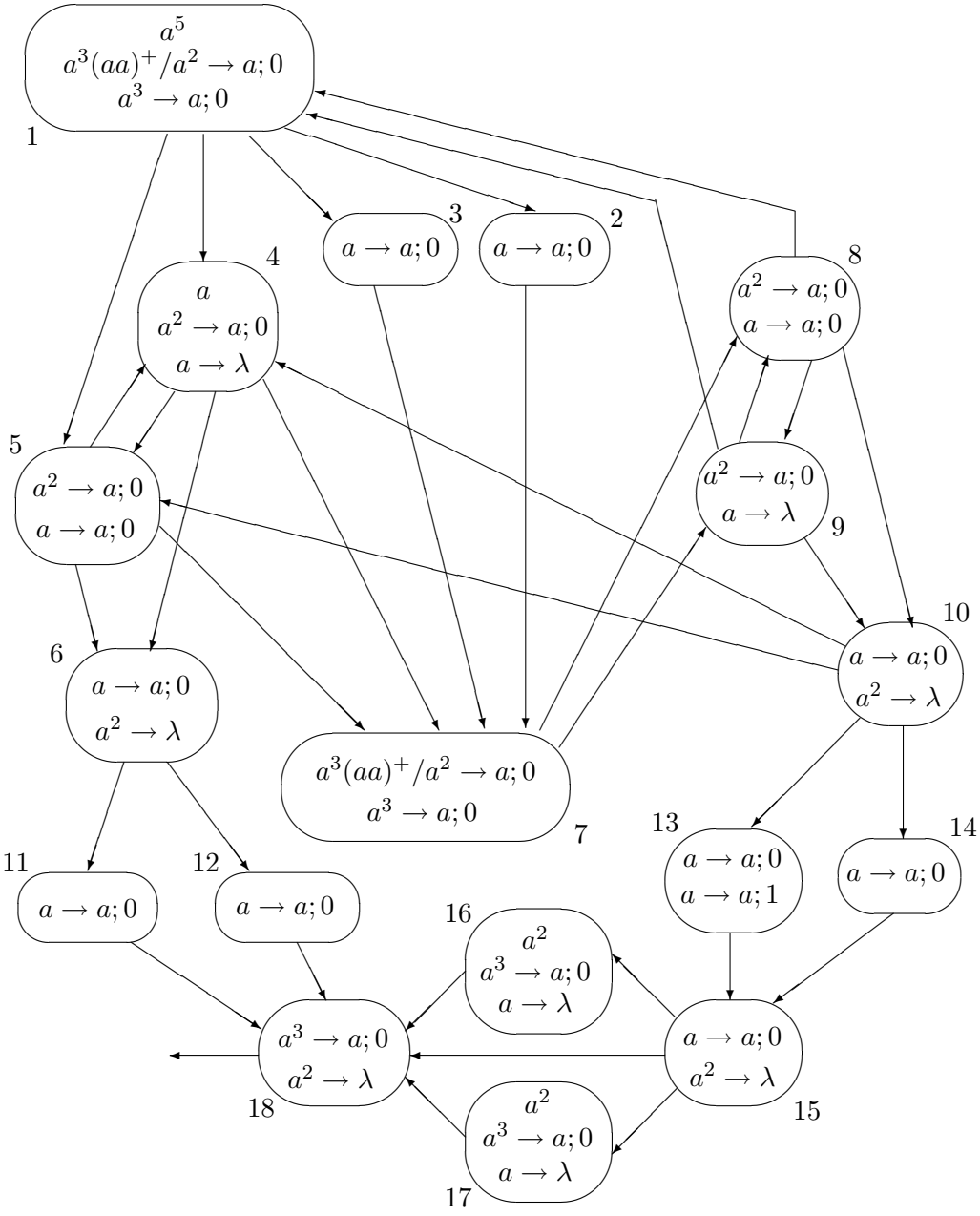

Figure 14: A "small" SN P system computing a non-semilinear set of numbers

spikes coming from neuron 6 to the output neuron are forgotten in neuron 18. When neuron 13 uses the rule $a \to a;1$, neuron *out* spikes, once when receiving the signal from neuron 10 and once when receiving the signal from neuron 6, and this means that in between we have m steps, for $2m+1$ being the contents of neuron 1 just moved to neuron 7. We leave the details to the reader and we only mention that $N_2(\Pi) = \{2^n \mid n \geq 2\}$, hence this non-semilinear set belongs to $Spik_2P_{18}(rule_3, cons_3, forg_2)$.

Can these parameters be (significantly) decreased? Which is the smallest number of neurons sufficient for computing a non-semilinear set of numbers?

Acknowledgements

The work of the first author was done during a stay in Waseda University, supported by the fellowship "Formación de Profesorado Universitario" from the Spanish Ministry of Education, Culture and Sport. Useful discussions with M.J. Pérez-Jiménez, G. Rozenberg, A. Rodríguez-Patón, P. Sosik are 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] W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.

[3] O.H. Ibarra, S. Woodworth, H.-C. Yen, Z. Dang: On symport/antiport P systems and semilinear sets. *Membrane Computing, International Workshop, WMC6, Vienna, Austria, July 2005, Selected and Invited Papers* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer-Verlag, Berlin, 2006, 255–273.

[4] Z. Kemp, A. Kowalczyk: Incorporating the temporal dimension in a GIS. Chapter 7 in *Innovations in GIS 1*, Taylor and Francis, London, 1994.

[5] V. Lum, P. Dadam, R. Erbe, J. Guenaver, P. Pistor, G. Walch, H. Werner, J. Woodfill: Designing dbms support for the temporal dimension. In *Proceedings of SIGMOD'84 Conference*, 1984, 115–130.

[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] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón: A new class of symbolic abstract neural nets: Tissue P systems. In *Proceedings of COCOON 2002*, Singapore, LNCS 2387, Springer-Verlag, Berlin, 2002, 290–299.

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

[10] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143 (also TUCS Report 208, November 1998, `www.tucs.fi`).

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

[12] Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.

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

[14] C. Teuscher: *Turing's Connectionism. An Investigation of Neural Network Architectures.* Springer-Verlag, London, 2002.

[15] S. Yu: The time dimension of computation models. *Technical Report 549*, Computer Sci. Dept., Univ. of Western Ontario, London-Ontario, Canada, 2000.

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