Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers

Artiom Alhazov[1,2], Rudolf Freund[3], Marion Oswald[3], and Marija Slavkovik[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md

[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.cat

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: {rudi,marion,marija}@emcc.at

Abstract. We consider extended variants of spiking neural P systems and show how these extensions of the original model allow for easy proofs of the computational completeness of extended spiking neural P systems and for the characterization of semilinear sets and regular languages by finite extended spiking neural P systems (defined by having only finite sets of rules assigned to the cells) with only a bounded number of neurons.

1 Introduction

Just recently, a new variant of P systems was introduced based on the biological background of neurons sending electrical impulses along axons to other neurons. This biological background had already led to several models in the area of neural computation, e.g., see [12], [13], and [8]. In the area of P systems, one basic model considers hierarchical membrane structures, whereas in another important model cells are placed in the nodes of a graph (which variant was first considered in [19]; tissue P systems then were further elaborated, for example, in [7] and [14]). Based on the structure of this model of tissue P systems, in [11] the new model of spiking neural P systems was introduced. The reader is referred to this seeding paper for the interesting details of the biological motivation for this kind of P systems; we will recall just a few of the most important features:

In spiking neural P systems, the contents of a cell (neuron) consists of a number of so-called *spikes*. The rules assigned to a cell allow us to send information to other neurons in the form of electrical impulses (also called spikes) which are summed up at the target cell; the application of the rules depends on the contents of the neuron and in the general case is described by regular sets. As inspired from biology, the cell sending out spikes may be "closed" for a specific time period corresponding to the refraction period of a neuron; during this refraction period, the neuron is closed for new input and cannot get excited ("fire") for spiking again.

The length of the axon may cause a time delay before a spike arrives at the target. Moreover, the spikes coming along different axons may cause effects of different magnitude. We shall include these features in our extended model of spiking neural P systems considered below. Some other features also motivated from biology will shortly be discussed in Section 5, e.g., the use of inhibiting neurons and spikes, respectively. From a mathematical point of view, the most important theoretical feature we shall include in our model of extended spiking neural P systems is that we allow the neurons to send spikes along the axons with different magnitudes at different moments of time.

In [11] the output of a spiking neural P system was considered to be the time between two spikes in a designated output cell. It was shown how spiking neural P systems in that way can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P system with a bounded number of spikes in the neurons. These results can also be obtained with even more restricted forms of spiking neural P systems, e.g., no time delay (refraction period) is needed, as it was shown in [10]. In [18], the behaviour of spiking neural P systems on infinite strings and the generation of infinite sequences of 0 and 1 (the case when the output neuron spikes) was investigated. Finally, in [1], the generation of strings (over the binary alphabet 0 and 1) by spiking neural P systems was investigated; due to the restrictions of the original model of spiking neural P systems, even specific finite languages cannot be generated, but on the other hand, regular languages can be represented as inverse-morphic images of languages generated by finite spiking neural P systems, and even recursively enumerable languages can be characterized as projections of inverse-morphic images of languages generated by spiking neural P systems. The problems occurring in the proofs are also caused by the quite restricted way the output is obtained from the output neuron as sequence of symbols 0 and 1. The strings of a regular or recursively enumerable language could be obtained directly by collecting the spikes sent by specific output neurons for each symbol.

In the extended model introduced in this paper, we shall use a specific output neuron for each symbol. Computational completeness can be obtained by simulating register machines as in the proofs elaborated in the papers mentioned above, yet in an easier way using only a bounded number of neurons. Moreover, regular languages can be characterized by finite extended spiking neural P systems; again, only a bounded number of neurons is really needed.

The rest of the paper is organized as follows: In the next section, we recall some preliminary notions and definitions, especially the definition and some well-known results for register machines. In section 3 we define our extended model of spiking neural P systems and explain how it works. The generative power of extended spiking neural P systems is investigated in section 4. Finally, in section 5 we give a short summary of the results obtained in this paper and discuss some further variants of extended spiking neural P systems, especially variants with inhibiting neurons or axons.

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [2] and [20]. We just list a few notions and notations: V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element. $\mathbb{N}_+$ denotes the set of positive integers (natural numbers), $\mathbb{N}$ is the set of non-negative integers, i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$, and $\mathbb{Z}$ denotes the set of integers. The interval of natural numbers between 1 and m is denoted by $[1..m]$. We call a set $M \subseteq \mathbb{Z}$ *regular* if and only if $M = M' \cup M''$ with $M' \subseteq \mathbb{N}$ and $M'' \subseteq -\mathbb{N}_+$ such that both M' and $-M''$ are regular (i.e., semilinear) subsets of $\mathbb{N}$. Observe that there is a one-to-one correspondence between a set $M \subseteq \mathbb{N}$ and the one-letter language $L(M) = \{a^n \mid n \in M\}$, hence, M is a regular (semilinear) set of non-negative integers if and only if $L(M)$ is a regular language. In an obvious way, the notion of regularity can be extended to subsets of $\mathbb{Z}^k$ for every $k \in \mathbb{N}_+$. By $FIN(M)$, $REG(M)$, and $RE(M)$, for any $M \subseteq \mathbb{Z}^k$, we denote the sets of subsets of M that are finite, regular, and recursively enumerable, respectively.

By REG $(REG(k))$ and RE $(RE(k))$ we denote the family of regular and recursively enumerable languages (over a k-letter alphabet), respectively. By $\Psi_T(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by $PsFL$ we denote the set of Parikh images of languages from a given family FL. In that sense, $PsRE(k)$ corresponds with the family of recursively enumerable sets of k-dimensional vectors of non-negative integers.

2.1 Register Machines

The proofs of the results establishing computational completeness in the area of P systems often are based on the simulation of register machines; we refer to [15] for original definitions, and to [4] for definitions like those we use in this paper:

An *n-register machine* is a construct $M = (n, P, l_0, l_h)$, where n is the number of registers, P is a finite set of instructions injectively labelled with elements from a given set $Lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (A(r), l_2, l_3)$ (ADD instruction)
  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) l_2 and l_3.
- $l_1 : (S(r), l_2, l_3)$ (SUB instruction)
  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3.

- $l_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

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

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [7]).

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

The results proved in [5] (based on the results established in [15]) and [4], [6] immediately lead to the following result:

Proposition 1. *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta + 2$ being empty, M non-deterministically computes and halts with n_i in registers i, $1 \leq i \leq \beta$, and registers $\beta+1$ and $\beta+2$ being empty if and only if $(n_1, ..., n_\beta) \in L$. Moreover, the registers 1 to β are never decremented.*

When considering the generation of languages, we can use the model of a *register machine with output tape*, which also uses a tape operation:

- $l_1 : (write(a), l_2)$
  Write symbol a on the output tape and go to instruction l_2.

We then also specify the output alphabet T in the description of the register machine with output tape, i.e., we write $M = (n, T, P, l_0, l_h)$.

The following result is folklore, too (e.g., see [15] and [3]):

Proposition 2. *Let $L \subseteq T^*$ be a recursively enumerable language. Then L can be generated by a register machine with output tape with 2 registers. Moreover, at the beginning and at the end of a successful computation generating a string $w \in L$ both registers are empty, and finally, only successful computations halt.*

3 Extended Spiking Neural P Systems

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [16]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [11].

An *extended spiking neural P system* (of degree $m \geq 1$) (in the following we shall simply speak of an *ESNP system*) is a construct

$$\Pi = (m, S, R)$$

where

- m is the number of *cells* (or *neurons*); the neurons are uniquely identified by a number between 1 and m (obviously, we could instead use an alphabet with m symbols to identify the neurons);
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron; for the sake of simplicity, we assume that at the beginning of a computation we have no pending packages along the axons between the neurons;
- R is a finite set of *rules* of the form $(i, E, k, d; r)$ such that $i \in [1..m]$ (specifying that this rule is assigned to cell i), $E \subseteq REG(\mathbb{N})$ is the *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, d is the *delay* (the "refraction time" when neuron i performs this rule), and r is a (possibly empty) sequence of *productions* of the form (l, w, t) where $l \in [1..m]$ (thus specifying the target cell), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron l, and t is the time needed before the information sent from neuron i arrives at neuron l (i.e., the *delay along the axon*).

A *configuration* of the ESNP system is described as follows:

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

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

- for each neuron i, we first check whether we find an "activated rule" $(i, E, k, d'; r)$ waiting to be executed; if $d' = 0$, then neuron i "spikes", i.e., for every production (l, w, t) occurring in the sequence r we put the corresponding package (l, w, t) on the axon from neuron i to neuron l, and after that, we eliminate this "activated rule" $(i, E, k, 0; r)$;
- for each neuron l, we now consider all packages (l, w, t') on axons leading to neuron l; provided the neuron is not closed, i.e., if it does not carry an activated rule $(i, E, k, d'; r)$ with $d' > 0$, we then sum up all weights w in such packages where $t' = 0$ and add this sum to the corresponding number of spikes in neuron l; in any case, the packages with $t' = 0$ are eliminated from the axons, whereas for all packages with $t' > 0$, we decrement t' by one;

– for each neuron i, we now again check whether we find an “activated rule” $(i, E, k, d'; r)$ (with $d' > 0$) or not; if we have not found an “activated rule”, we now may apply any rule $(i, E, k, d; r)$ from R for which the current number of spikes in the neuron is in E and then put a copy of this rule as “activated rule” for this neuron into the description of the current configuration; on the other hand, if there still has been an “activated rule” $(i, E, k, d'; r)$ in the neuron with $d' > 0$, then we replace d' by $d' - 1$ and keep $(i, E, k, d' - 1; r)$ as the “activated rule” in neuron i in the description of the configuration for the next step of the computation.

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A *computation* is a sequence of configurations starting with the initial configuration given by S. A computation is called *successful* if it halts, i.e., if no pending package can be found along any axon, no neuron contains an activated rule, and for no neuron, a rule can be activated.

In the original model introduced in [11], in the productions (l, w, t) of a rule $(i, E, k, d; \{(l, w, t)\})$, only $w = 1$ (for *spiking rules*) or $w = 0$ (for *forgetting rules*) as well as $t = 0$ was allowed (and for forgetting rules, the checking set E had to be finite and disjoint from all other sets E in rules assigned to neuron i). Moreover, reflexive axons, i.e., leading from neuron i to neuron i, were not allowed, hence, for (l, w, t) being a production in a rule $(i, E, k, d; r)$ for neuron i, $l \neq i$ was required. Yet the most important extension is that different rules for neuron i may affect different axons leaving from it whereas in the original model the structure of the axons (called synapses there) was fixed. Finally, we should like to mention that the sequence of substeps leading from one configuration to the next one together with the interpretation of the rules from R was taken in such a way that the original model can be interpreted in a consistent way within the extended model introduced in this paper. From a mathematical point of view, another interpretation in our opinion would have been more suitable: whenever a rule $(i, E, k, d; r)$ is activated, the packages induced by the productions (l, w, t) in the sequence r of a rule $(i, E, k, d; r)$ activated in a computation step are immediately put on the axon from neuron i to neuron l, whereas the delay d only indicates the refraction time for neuron i itself, i.e., the time period this neuron will be closed. Yet as in the proofs of computational completeness given below we shall not need any of the delay features, we shall not investigate this variant in more details anymore in the rest of the paper.

Depending on the purpose the ESNP system shall be used, some more features have to be specified: for generating k-dimensional vectors of non-negative integers, we have to designate k neurons as *output neurons*; the other neurons then will also be called *actor neurons.* Without loss of generality, in the following we shall assume the output neurons to be the first k neurons of the ESNP system. Moreover, for the sake of conciseness, we shall also assume that no rules are assigned to these output neurons (as in the original model they correspond to a sensor in the environment of the system and are not neurons of the system itself). There are several possibilities to define how the output values are computed; according to [11], we can take the distance between the first two spikes in an output neuron to define its value; in this paper, we shall prefer to take the number of spikes at the end of a successful computation in the neuron as the output value. For generating strings, we do not interpret the spike train of a single output neuron as done, for example, in [1], but instead consider the sequence of spikes in the output neurons each of them corresponding to a specific terminal symbol; if more than one output neuron spikes, we take any permutation of the corresponding symbols as the next substring of the string to be generated.

The delay t in productions (l, w, t) can be used to replace the delay in the neurons themselves in many of the constructions elaborated, for example, in [11], [17], and [1]; there often a subconstruction is implemented which ensures that a neuron l_3 gets a spike one time step later than a neuron l_2, both getting the impulse from a neuron l_1; to accomplish this task in the original model, two intermediate neurons are needed using the refraction period delay of one neuron, whereas we can get this effect directly from neuron l_1 by using the delay along the axons using the rule $(l_1, E, k, 0; (l_2, 1, 0), (l_3, 1, 1))$. In that way, only this feature allows for simpler proofs; on the other hand, taking into account the other extensions in ESNP systems as defined above, we shall not need any of the delay features for the proofs of computational completeness given below.

4 ESNP Systems as Generating Devices

We now consider extended spiking neural P systems as generating devices. As throughout this section we do not use delays in the rules and productions, we simply omit them to keep the description of the systems concise, i.e., for a production (i, j, t) we simply write (i, j); for example, instead of $(2, \{i\}, i, 0; (1, 1, 0), (2, j, 0))$ we write $(2, \{i\}, i; (1, 1), (2, j))$.

The following example gives a characterization of regular sets of non-negative integers:

Example 1. Any semilinear set of non-negative integers M can be generated by a finite ESNP system with only two neurons.

Let M be a semilinear set of non-negative integers and consider a regular grammar G generating the language $L(G) \subseteq \{a\}^*$ with $N(L(G)) = M$; without loss of generality we assume the regular grammar to be of the form $G = (N, \{a\}, A_1, P)$ with the set of non-terminal symbols N, $N = \{A_i \mid 1 \leq i \leq m\}$, the start symbol A_1, and P the set of regular productions of the form $B \to aC$ with $B, C \in N$ and $A \to \lambda$. We now construct the finite ESNP system $\Pi = (2, S, R)$ that generates an element of M by the number of spikes contained in the output neuron 1 at the end of a halting computation: we start with one spike in neuron 2 (representing the start symbol A_1 and no spike in the output neuron 1, i.e., $S = \{(1, 0), (2, 1)\}$. The production $A_i \to aA_j$ is simulated by the rule $(2, \{i\}, i; (1, 1), (2, j))$ and $A_i \to \lambda$ is simulated by the rule $(2, \{i\}, i;)$, i.e., in sum we obtain

$$\begin{aligned}
\Pi &= (2, S, R)\,, \\
S &= \{(1,0), (2,1)\}\,, \\
R &= \{(2, \{i\}, i; (1,1), (2,j)) \mid 1 \leq i, j \leq m, A_i \rightarrow aA_j \in P\} \\
&\quad \cup \{(2, \{i\}, i;) \mid 1 \leq i \leq m, A_i \rightarrow \lambda \in P\}\,.
\end{aligned}$$

Neuron 2 keeps track of the actual non-terminal symbol and stops the derivation as soon as it simulates a production $A_i \rightarrow \lambda$, because finally neuron 2 is empty. In order to guarantee that this is the only way how we can obtain a halting computation in Π, without loss of generality we assume G to be reduced, i.e., for every non-terminal symbol A from N there is a regular production with A on the left-hand side. These observations prove that we have $N(L(G)) = M$.

We should like to mention that we can also generate the numbers in M as the difference between the (first) two spikes arriving in the output neuron by the following ESNP system Π':

$$\begin{aligned}
\Pi' &= (2, S', R')\,, \\
S' &= \{(1,0), (2, m+1)\}\,, \\
R' &= \{(2, \{i\}, i; (2,j)) \mid 1 \leq i, j \leq m, A_i \rightarrow aA_j \in P\} \\
&\quad \cup \{(2, \{i\}, i; (1,1)) \mid 1 \leq i \leq m, A_i \rightarrow \lambda \in P\} \\
&\quad \cup \{(2, \{m+1\}, m+1; (1,1), (2,1))\}\,.
\end{aligned}$$

We should like to mention that one reason for the constructions given above to be that easy is the fact that we allow "reflexive" axons, i.e., we can keep track of the actual non-terminal symbol without delay. We could avoid this by adding an additional neuron 3 thus obtaining the following finite ESNP system:

$$\begin{aligned}
\Pi'' &= (3, S'', R'')\,, \\
S'' &= \{(1,0), (2,0), (3,1)\}\,, \\
R'' &= \{(2, \{i\}, i; (3,j), (1,1)), (3, \{i\}, i; (2,i)) \\
&\qquad \mid 1 \leq i, j \leq m, A_i \rightarrow aA_j \in P\} \\
&\quad \cup \{(2, \{i\}, i;) \mid 1 \leq i \leq m, A_i \rightarrow \lambda \in P\}\,.
\end{aligned}$$

Observe that the derivation in the corresponding grammar now is delayed by a factor of 2 in the computation in Π, because we need one step to propagate the information from neuron 3 to neuron 2 which then sends the spikes to the output neuron. Hence, an interpretation of the generated number as the difference between two spikes in the output neuron is not possible anymore.

Lemma 1. *For any ESNP system where during a computation only a bounded number of spikes occurs in the actor neurons, the generated language is regular.*

Proof (sketch). Let Π be an ESNP system where during a computation only a bounded number of spikes occurs in the actor neurons. Then the number of configurations differing in the actor neurons and the packages along the axons, but without considering the contents of the output neurons, is finite, hence, we can assign non-terminal symbols A_k to each of these configurations and take all right-regular productions $A_i \to wA_j$ such that w is a permutation of the string obtained by concatenating the symbols indicated by the number of added spikes when going from configuration i to configuration j. If we interpret just the number of spikes in the output neurons as the result of a successful computation, then we obtain the Parikh set of the language, which then obviously is regular, i.e., semilinear. $\square$

As we shall see later in this section, ESNP systems are computationally complete, hence, the question whether in an ESNP system during a computation only a bounded number of spikes occurs in the actor neurons is not decidable, but there is a simple syntactic feature that allows us to characterize regular sets, namely the finiteness of an ESNP system. The following theorem is a consequence of the preceding lemma and the example given above:

Theorem 1. *Any regular language L with $L \subseteq T^*$ for a terminal alphabet T with card $(T) = n$ can be generated by a finite ESNP system with $n+1$ neurons. On the other hand, every language generated by a finite ESNP system is regular.*

Proof. Let G be a regular grammar G generating the language $L(G) = L \subseteq T^*$, $T = \{a_k \mid 1 \leq k \leq n\}$; without loss of generality we assume the regular grammar to be of the form $G = (N, T, A_1, P)$ with the set of non-terminal symbols N, $N = \{A_i \mid 1 \leq i \leq m\}$, the start symbol A_1, and P the set of regular productions of the form $A_i \to a_k A_j$ with $A_i, A_j \in N$, $a_k \in T$, and $A_i \to \lambda$ with $A_i \in N$. We now construct the finite ESNP system $\Pi = (n+1, S, R)$ that generates an element of L by the sequence of spikes in the output neurons 1 to n corresponding with the desired string during a halting computation: we start with one spike in neuron $n+1$ (representing the start variable A_1 and no spike in the output neurons 1 to n:

$$\begin{aligned}
\Pi &= (n+1, S, R)\,, \\
S &= \{(1,0), ..., (n,0), (n+1,1)\}\,, \\
R &= \{(n+1, \{i\}, i; (k,1), (n+1,j))\,, \\
&\qquad \mid 1 \leq i,j \leq m, 1 \leq k \leq n, A_i \to a_k A_j \in P\} \\
&\quad \cup \{(n+1, \{i\}, i;) \mid 1 \leq i \leq m, A_i \to \lambda \in P\}\,.
\end{aligned}$$

Obviously, $L(G) = L(\Pi) = L$.

On the other hand, let $\Pi = (m, S, R)$ be a finite ESNP system. Then from Π we can construct an equivalent finite ESNP system $\Pi' = (m, S, R')$ such that Π' fulfills the requirements of Lemma 1: let x be the maximal number occurring in all the checking sets of rules from R and let y be (a number not less than) the maximal number of spikes that can be added in one computation step to any of the m neurons of Π (an upper bound for y is the maximal number of weights in the productions of the rules in R multiplied by the maximal delay in these productions $+1$ multiplied by the maximal number of axons ending in a neuron of Π); then define

$$R' = R \cup \{(i, \{x+1+k\}, k;) \mid 1 \leq k < 2y\}\,.$$

Hence, the maximal number of spikes in any of the neurons of Π' is $x + 2y$, therefore Lemma 1 can be applied (observe that the additional rules in R' cannot lead to additional infinite computations, because they only consume spikes, but let the contents of the neurons stay above x, hence, no other rules become applicable). □

Corollary 1. *Any semilinear set of n-dimensional vectors can be generated by a finite ESNP system with $n+1$ neurons. On the other hand, every set of n-dimensional vectors generated by a finite ESNP system is semilinear.*

Proof. The result directly follows from Theorem 1 by just taking the number of spikes in the output neurons, i.e., the corresponding Parikh set of the generated language (because $PsREG\,(n) = REG\,(\mathbb{N}^n)$). □

We now show that every recursively enumerable language over an n-letter alphabet can be generated by an ESNP system with a bounded number of neurons:

Theorem 2. *Any recursively enumerable language L with $L \subseteq T^*$ for a terminal alphabet T with $card\,(T) = n$ can be generated by an ESNP system with $n+3$ neurons.*

Proof. Let $M = (2, T, P, l_0, l_h)$ be a register machine with output tape generating L according to Proposition 2. Then we construct an ESNP system $\Pi = (n+3, S, R)$ where neurons 1 to n are the output neurons, neurons $n+1$ and $n+2$ represent the two registers of M and neuron $n+3$ is the actor neuron guiding the whole computation by carrying out all actions needed to simulate the instructions of P. The main idea of the construction of Π is that neurons $n+1$ and $n+2$ in each even computation step send one or two spikes, respectively, to neuron $n+3$ if they are non-empty, hence, in the odd steps neuron $n+3$ "knows" what to do next when simulating an instruction from Π, e.g., if the contents of one of these neurons is non-empty and has to be incremented, then two spikes are sent back, if it has to be decremented, no spike is sent back, and otherwise one spike is sent back to restore the original contents. As at the end of a successful computation both registers are empty, also the computation in Π stops because neurons $n+1$ and $n+2$ will not spike anymore. A tape operation $l_1 : (write\,(a_k)\,, l_2)$ is simulated by sending a spike to the output neuron representing symbol a_k. Now let $Lab\,(M) = [1..h]$ be the set of labels for the instructions in P; without loss of generality, we assume $l_0 = 1$ and $l_h = h$. Then

Π is constructed as follows:

$$
\begin{aligned}
\Pi &= (n+3, S, R)\,,\\
S &= \{(1,0)\,, ..., (n,0)\,, (n+1,1)\,, (n+2,1)\,, (n+3,4)\}\,,\\
R &= \{(n+3, \{4l_1\}\,, 4l_1; (n+3, 8l_2)\,, (k,1))\,,\\
&\quad (n+3, \{4l_1+1\}\,, 4l_1+1; (n+1,1)\,, (n+3,8l_2)\,, (k,1))\,,\\
&\quad (n+3, \{4l_1+2\}\,, 4l_1+2; (n+2,1)\,, (n+3,8l_2)\,, (k,1))\,,\\
&\quad (n+3, \{4l_1+3\}\,, 4l_1+3; (n+1,1)\,, (n+2,1)\,, (n+3,8l_2)\,, (k,1))\\
&\quad \mid l_1 : (write\,(a_k)\,, l_2) \in P\}\\
&\cup \{(n+3, \{4l_1\}\,, 4l_1; (n+1,1)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+1\}\,, 4l_1+1; (n+1,2)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+2\}\,, 4l_1+2; (n+1,1)\,, (n+1,1)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+3\}\,, 4l_1+3; (n+1,2)\,, (n+1,1)\,, (n+3,8l))\\
&\quad \mid l_1 : (A\,(1)\,, l_2, l_3) \in P, l \in \{l_2, l_3\}\}\\
&\cup \{(n+3, \{4l_1\}\,, 4l_1; (n+2,1)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+1\}\,, 4l_1+1; (n+1,1)\,, (n+1,1)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+2\}\,, 4l_1+2; (n+2,2)\,, (n+3,8l))\,,\\
&\quad (n+3, \{4l_1+3\}\,, 4l_1+3; (n+1,1)\,, (n+2,2)\,, (n+3,8l))\\
&\quad \mid l_1 : (A\,(2)\,, l_2, l_3) \in P, l \in \{l_2, l_3\}\}\\
&\cup \{(n+3, \{4l_1\}\,, 4l_1; (n+3,8l_3))\,,\\
&\quad (n+3, \{4l_1+1\}\,, 4l_1+1; (n+3,8l_2))\,,\\
&\quad (n+3, \{4l_1+2\}\,, 4l_1+2; (n+2,1)\,, (n+3,8l_3))\,,\\
&\quad (n+3, \{4l_1+3\}\,, 4l_1+3; (n+2,1)\,, (n+3,8l_2))\\
&\quad \mid l_1 : (S\,(1)\,, l_2, l_3) \in P\}\\
&\cup \{(n+3, \{4l_1\}\,, 4l_1; (n+3,8l_3))\,,\\
&\quad (n+3, \{4l_1+1\}\,, 4l_1+1; (n+1,1)\,, (n+3,8l_3))\,,\\
&\quad (n+3, \{4l_1+2\}\,, 4l_1+2; (n+3,8l_2))\,,\\
&\quad (n+3, \{4l_1+3\}\,, 4l_1+3; (n+1,1)\,, (n+3,8l_2))\\
&\quad \mid l_1 : (S\,(2)\,, l_2, l_3) \in P\}\\
&\cup \{(n+1, \mathbb{N}_0, 1; (n+3,1))\,, (n+2, \mathbb{N}_0, 1; (n+3,2))\}\\
&\cup \{(n+3, \{8l+3\}\,, 4l+3; (n+1,1)\,, (n+2,1)) \mid 1 \le l < h\}\\
&\cup \{(n+3, \{8h+3\}\,, 8h+3;)\}\,.
\end{aligned}
$$

The label l of an instruction from P is encoded as $4l$ in the odd steps of a computation in Π and as $8l$ in the even steps. $4l$ in neuron $n+3$ indicates that both neurons $n+1$ and $n+2$ are empty, $4l+1/4l+2$ indicates that neuron $n+1/\ n+2$, respectively, is non-empty, whereas the other neuron is empty, and $4l+3$ indicates that both neurons are non-empty. As in the odd step both neurons $n+1$ and $n+2$ have received a spike from neuron $n+3$, in the even step the contents of neuron $n+3$ will be of the form $8l+3$. The final rule $(n+3, \{8h+3\}\,, 8h+3;)$ guarantees that the ESNP system stops a successful computation with the neurons $n+1$, $n+2$, and $n+3$ being empty. □

As can be seen from the construction given in the preceding proof, the only infinite checking sets are of the form $\mathbb{N}$ – corresponding with $\{a\}^*$ in the normal forms proved in [10].

Corollary 2. *Any recursively enumerable set of n-dimensional vectors can be generated by an ESNP system with $n+3$ neurons.*

Proof. The result directly can be proved by using Proposition 1 and a similar construction as that one elaborated in the proof of Theorem 2, yet it also follows from Theorem 2 by just taking the number of spikes in the output neurons as the value of the components of the n-dimensional vector, i.e., by taking the corresponding Parikh set of the generated language (because $PsRE\,(n) = RE\,(\mathbb{N}^n)$). □

5 Summary and Further Variants

In this paper, we have considered various extensions of the original model of spiking neural P systems, some of them arising from biological motivations, some others being more of mathematical interest than having a biological interpretation. The extensions considered in more detail here allowed for establishing computational completeness in an easy way, and moreover, we got a quite natural characterization of semilinear sets of (vectors of) non-negative integers and regular languages, respectively, by finite extended spiking neural P systems with a bounded number of neurons. On the other hand, in the future some other restrictions should be investigated allowing for the characterization of sets in a family between the families of regular and recursively enumerable sets (of vectors of non-negative integers or strings).

A quite natural feature found in biology and also used in the area of neural computation is that of inhibiting neurons or connections between neurons. We can include this feature in our extended model of spiking neural P systems considered above in the following way: we also allow negative values for the weights w in the productions (l, w, t), i.e., we take $w \in \mathbb{Z}$; when this "package" arrives at the target cell l, then it causes this cell to be closed for t time steps (all such negative values are summed up at the moment they arrive at the cell provided it is not closed). Then the definition of finite extended spiking neural P systems with inhibiting spikes still only demands all checking sets to be finite, and it is easy to see that finite extended spiking neural P systems with inhibiting spikes still characterize the regular sets. Inhibiting cells can also be achieved in another way in a variant closely related to the original model of spiking neural P systems by specifying certain connections from one neuron to another one as inhibiting ones – the spikes coming along such inhibiting axons then again would close the target neuron for a time period given by the sum of all inhibiting spikes.

The negative values propagated along the axons also allow for another interpretation (than that of inhibiting the cell for a specific time period) when using integer values in the cells: we can simply sum up all the values arriving at a specific moment at the target cell thus yielding an integer number to be added to the current contents of the cell. In that way, we obtain a quite natural mechanism for generating sets of (vectors of) integer numbers. The checking sets

E in the rules $(i, E, k, d; r)$ then have to be regular sets of integer numbers, i.e., $E \subseteq REG(\mathbb{Z})$. Moreover, these systems working with vectors of integers might also allow for a comparison with vector addition systems (e.g., as done in [9] for catalytic P systems). Finally, we might even generalize this variant to finitely generated commutative groups, as each such group is isomorphic to $\mathbb{Z}^k \times H$ for some $k \geq 1$ and a finite commutative group H; the elements of $\mathbb{Z}^k \times H$ can easily be represented by $k+1$ neurons, and in this way, for example, recursively enumerable subsets of $\mathbb{Z}^k \times H$ can be computed in a similar way as described in the preceding section.

6 Acknowledgements

The first and the second author very much appreciate the interesting discussions with Gheorghe Păun during the Brainstorming Week on Membrane Computing 2006 in Sevilla on the main features of spiking neural P systems. The work of Artiom Alhazov is partially supported by the project TIC2003-09319-C03-01 from Rovira i Virgili University. The work of Marion Oswald is supported by FWF-project T225-N04.

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On String Languages Generated by Spiking Neural P Systems. In: M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F. José Romero-Campero (Eds.), Fourth Brainstorming Week on Membrane Computing, Vol. I RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, Sevilla, 2006, 169–194.
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
3. H. Fernau, R. Freund, M. Oswald, K. Reinhardt: Refining the Nonterminal Complexity of Graph-controlled Grammars. In: C. Mereghetti, B. Palano, G. Pighizzini, D. Wotschke (Eds.), *Seventh International Workshop on Descriptional Complexity of Formal Systems*, 2005, 110–121.
4. R. Freund, M. Oswald: P Systems with activated/prohibited membrane channels. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop WMC 2002, Curteă de Argeş, Romania, Revised Papers. *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2003), 261–268.
5. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1–3 (2002), 81–102.
6. R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312** (2004), 143–188.
7. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2005), 101–116.
8. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.
9. O. Ibarra, Z. Dang, and O. Egecioglu: Catalytic P systems, semilinear sets, and vector addition systems. *Theoretical Computer Science* **312**, 2–3 (2004), 379–399.

10. O. H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal Forms for Spiking Neural P Systems. In: M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F. José Romero-Campero (Eds.), Fourth Brainstorming Week on Membrane Computing, Vol. II, RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, Sevilla, 2006, 105–136.
11. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae* **71**, 2–3 (2006), 279–308,.
12. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, **8**, 1 (2002), 32–36.
13. W. Maass, C. Bishop (Eds.): *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.
14. 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, *Lecture Notes in Computer Science* **2387**, Springer-Verlag, Berlin, 2002, 290–299.
15. M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
16. Gh. Păun: *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin, 2002.
17. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems, *Intern. J. Found. Computer Sci.*, to appear (also available at [21]).
18. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted 2006.
19. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen* **60** (2002), 635–660.
20. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin, 1997.
21. The P Systems Web Page, `http://psystems.disco.unimib.it`