Extended Spiking Neural P Systems with White Hole Rules

Artiom Alhazov[1], Rudolf Freund[2], Sergiu Ivanov[3], Marion Oswald[2], and Sergey Verlan[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] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
E-mail: {`rudi,marion`}`@emcc.at`

[3] Université Paris Est, France
E-mail: {`sergiu.ivanov,verlan`}`@u-pec.fr`

Summary. We consider extended spiking neural P systems with the additional possibility of so-called "white hole rules", which send the complete contents of a neuron to other neurons, and we show how this extension of the original model allow for easy proofs of the computational completeness of this variant of extended spiking neural P systems using only one actor neuron. Using only such white hole rules, we can easily simulate special variants of Lindenmayer systems.

1 Introduction

Based on the biological background of neurons sending electrical impulses along axons to other neurons, several models were developed in the area of neural computation, e.g., see [15], [16], and [10]. In the area of P systems, the model of *spiking neural P systems* was introduced in [14]. Whereas the basic model of membrane systems reflects hierarchical membrane structures, the model of tissue P systems considers cells to be placed in the nodes of a graph. This variant was first considered in [23] and then further elaborated, for example, in [9] and [17]. In spiking neural P systems, the cells are arranged as in tissue P systems, but the contents of a cell (neuron) consists of a number of so-called *spikes*, i.e., of a multiset over a single object. 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. All these biologically motivated features were included in the model of extended spiking neural P systems considered in [1], the most important theoretical feature being that neurons can send spikes along the axons with different magnitudes at different moments of time. In this paper, we will further extend the model of extended spiking neural P systems by using so-called "white hole rules", which allow us to use the whole contents of a neuron and send it to other cells, yet eventually multiplied by some constant rational number.

In the literature, several variants how to obtain results from the computations of a spiking neural P system have been investigated. For example, in [14] 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 [13]. In [4], 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 considered in [1], a specific output neuron was used for each symbol. Computational completeness could 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 could be characterized by finite extended spiking neural P systems; again, only a bounded number of neurons was needed.

In this paper, we now extend this model of extended spiking neural P systems by also using so-called "white hole rules", which may send the whole contents of a neuron along its axons, eventually even multiplied by a (positive) rational number. In that way, the whole contents of a neuron can be multiplied by a rational number, in fact, multiplied with or divided by a natural number. Hence, even one single neuron is able to simulate the computations of an arbitrary register machine.

The idea of consuming the whole contents of a neuron by white hole rules is closely related with concept of the exhaustive use of rules, i.e., an enabled rule is applied in the maximal way possible in one step; P systems with the exhaustive use of rules can be used in the usual maximally parallel way on the level of the whole system or in the sequential way, for example, see [27] and [26]. Yet all the approaches of spiking neural P systems with the exhaustive use of rules are mainly based on the classic definitions of spiking neural P systems, whereas the spiking neural P systems with white hole rules as investigated in this paper are based on the extended model as introduced in [1].

The rest of the paper is organized as follows: In the next section, we recall some preliminary notions and definitions from formal language theory, especially the definition and some well-known results for register machines. In section 3 we recall the definitions of the extended model of spiking neural P systems as considered in [1] as well as the most important results established there. Moreover, we show a new result for extended spiking neural P systems – such systems with only one actor neuron have exactly the same computational power as register machines with only one register that can be decremented.

In section 4, we define the model of extended spiking neural P systems extended by the use of white hole rules. Besides giving some examples, for instance showing how Lindenmayer systems can be simulated by extended spiking neural P systems only using white hole rules, we prove that the computations of an arbitrary register machine can be simulated by only one single neuron equipped with the most powerful variant of white hole rules. In that way we can show that extended spiking neural P systems equipped with white hole rules are even more powerful than extended spiking neural P systems, which need (at least) two neurons to be able to simulate the computations of an arbitrary register machine. Finally, in section 5 we give a short summary of the results obtained in this paper and discuss some future research topics for extended spiking neural P systems with white hole rules, for example, variants with inhibiting neurons or axons.

2 Preliminaries

In this section we recall the basic elements of formal language theory and especially the definitions and results for register machines; we here mainly follow the corresponding section from [1].

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [5] and [25]. 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; for any $w \in V^*$, $|w|$ denotes the number of symbols in w (the *length* of w). $\mathbb{N}_+$ 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 $\mathbb{Z}$ is the set of integers, i.e., $\mathbb{Z} = \mathbb{N}_+ \cup \{0\} \cup -\mathbb{N}_+$. The interval of non-negative integers between k and m is denoted by $[k..m]$, and $k \cdot \mathbb{N}_+$ denotes the set of positive multiples of k. Observe that there is a one-to-one correspondence between a set $M \subseteq \mathbb{N}$ and the one-letter language $L(M) = \{a^n \mid n \in M\}$; e.g., M is a regular (semilinear) set of non-negative integers if and only if $L(M)$ is a regular language. By $FIN(\mathbb{N}^k)$, $REG(\mathbb{N}^k)$, and $RE(\mathbb{N}^k)$, for any $k \in \mathbb{N}$, we denote the sets of subsets of $\mathbb{N}^k$ that are finite, regular, and recursively enumerable, respectively.

By REG ($REG(V)$) and RE ($RE(V)$) we denote the family of regular and recursively enumerable languages (over the alphabet V, 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, $PsRE(V)$ for a k-letter alphabet V 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 [18] for original definitions, and to [7] 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:

- $l_1 : (ADD(r), l_2, l_3)$ (ADD instruction)
  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) l_2 and l_3.
- $l_1 : (SUB(r), l_2, l_3)$ (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 $(s_1, \cdots, s_\beta)$ 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 $s_1, \cdots, s_\beta$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (ADD(r), l_2, l_3)$ and in each SUB instruction $l_1 : (SUB(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [9]).

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 $PsRE$.

Based on the results established in [18], the results proved in [7] and [8] 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 [18] and [6]):

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 [21] and [24]; comprehensive information can be found on the P systems web page [28]. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [14]. The definition of an *extended spiking neural P system* is mainly taken from [1], with the number of spikes k still be given in the "classical" way as a^k; later on, we simple will use the number k itself only instead of a^k.

3.1 The Definition of ESNP Systems

The definitions given in the following are taken from [1].

Definition 1. *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* $\left(i, E/a^k \to P; d\right)$ *such that* $i \in [1..m]$ *(specifying that this rule is assigned to cell* i*),* $E \subseteq REG(\{a\})$ *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* P *is a (possibly empty) set of* productions *of the form* (l, w, t) *where* $l \in [1..m]$ *(thus specifying the target cell),* $w \in \{a\}^*$ *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*). If the checking sets in all rules are finite, then* Π *is called a* finite ESNP system.

Definition 2. *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"* $\left(i, E/a^k \to P; d'\right)$ *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"* $\left(i, E/a^k \to P; d'\right)$ *waiting to be executed; if* $d' = 0$*, then neuron* i *"spikes", i.e., for every production* (l, w, t) *occurring in the set* P *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"* $\left(i, E/a^k \to P; d'\right)$*;*
- *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* $\left(i, E/a^k \to P; d'\right)$ *with* $d' > 0$*, we then sum up all weights* w *in such packages where* $t' = 0$ *and add this sum of spikes 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"* $\left(i, E/a^k \to P; d'\right)$ *(with* $d' > 0$*) or not; if we have not found an "activated rule", we now may apply any rule* $\left(i, E/a^k \to P; d\right)$ *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"* $\left(i, E/a^k \to P; d'\right)$ *in the*

neuron with $d' > 0$*, then we replace* d' *by* $d' - 1$ *and keep* $\left(i, E/a^k \to P; d' - 1\right)$ *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 [14], in the productions (l, w, t) of a rule $\left(i, E/a^k \to \{(l, w, t)\}; d\right)$, only $w = a$ (for *spiking rules*) or $w = \lambda$ (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 $\left(i, E/a^k \to P; d\right)$ 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. In [1], 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 that paper. As mentioned in [1], from a mathematical point of view, another interpretation would have been even more suitable: whenever a rule $\left(i, E/a^k \to P; d\right)$ is activated, the packages induced by the productions (l, w, t) in the set P of a rule $\left(i, E/a^k \to P; d\right)$ 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. 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 [14], [23], and [4]. Yet as in (the proofs of computational completeness given in) [1], we shall not need any of the delay features in this paper, hence we need not go into the details of these variants of interpreting the delays in more details.

Depending on the purpose the ESNP system is to 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*. There are several possibilities to define how the output values are computed; according to [14], we can take the distance between the first two spikes in an output neuron to define its value. As in [1], also in this paper, we 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 [4], 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.

Remark 1. As already mentioned, there is a one-to-one correspondence between (sets of) strings a^k over the one-letter alphabet $\{a\}$ and the corresponding non-negative integer k. Hence, in the following, we will consider the checking sets E of a rule $\left(i, E/a^k \to P; d\right)$ to be sets of non-negative integers and write k instead of a^k for any $w = a^k$ in a production (l, w, t) of P. Moreover, if no delays d or t are needed, we simply omit them. For example, instead of $\left(2, \left\{a^i\right\}/a^i \to \left\{(1, a, 0), \left(2, a^j, 0\right)\right\}; 0\right)$ we write $(2, \{i\}/i \to \{(1, 1), (2, j)\})$.

3.2 ESNP Systems as Generating Devices

As in [1], we first consider extended spiking neural P systems as generating devices. 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, $N = \{A_i \mid 1 \le i \le m\}$, 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 $(2, \{i\}/i \to \{(1, 1), (2, j)\})$ and $A_i \to \lambda$ is simulated by the rule $(2, \{i\}/i \to \emptyset)$, i.e., in sum we obtain

$$\begin{aligned}
\Pi &= (2, S, R), \\
S &= \{(1, 0), (2, 1)\}, \\
R &= \{(2, \{i\}/i \to \{(1, 1), (2, j)\}) \mid 1 \le i, j \le m, A_i \to aA_j \in P\} \\
&\cup \{(2, \{i\}/i \to \emptyset) \mid 1 \le i \le m, A_i \to \lambda \in P\}.
\end{aligned}$$

Neuron 2 keeps track of the actual non-terminal symbol and stops the derivation as soon as it simulates a production $A_i \to \lambda$, 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 $N(L(G)) = M$.

The following results were proved in [1]:

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

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.*

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.*

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 $n+2$ neurons.*

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

Besides these results already established in [1], we now prove a characterization of languages and sets of (vectors of) natural numbers generated by ESNPS with only one neuron. Roughly speaking, having only one actor neuron corresponds with, besides output registers, having only one register which can be decremented.

Lemma 2. *For any ESNP system with only one actor neuron we can effectively construct a register machine with output tape and only one register that can be decremented, generating the same language, respectively a register machine with one register that can be decremented, generating the same set of (vectors of) natural numbers.*

Proof. First we notice that the delays would not matter: the overall system is sequential, and therefore it is always possible to pre-compute what happens until the actor neuron re-opens; the weight of all pending packages is also bounded. All the details of storing and managing all these features by the finite control of the register machines are tedious, but very much straightforward. In the following, we therefore assume that the ESNPS is given as:

$$\Pi = (n+1, S, R),$$
$$S = \{(1, m_1), \cdots, (n, m_n), (n+1, m_{n+1})\},$$
$$R = \{(n+1, E_r/i_r \to \{(1, p_{r,1}), \cdots, (n, p_{r,n}), (n+1, p_{r,n+1})\}) \mid 1 \le r \le q\}.$$

Thus, given n, Π can be specified by the following non-negative integers: the number q of rules, initial spikes $m_1, \cdots, m_n, m_{n+1}$, and, for every rule r, the following ingredients: the number i_r of consumed spikes, the numbers $p_{r,1}, \cdots, p_{r,n+1}$ of produced spikes, and the regular sets E_r of numbers. Note that, as it will be obvious later, it is enough to only consider the case $m_1 = \cdots = m_n = 0$, because otherwise placing the initial spikes can be done by a 1-register machine in a preparatory phase, before switching to the instruction corresponding to starting the simulation.

The main challenge of the construction is to remember the actual "status" of the regular checking sets. It is known that every regular set E of numbers is semilinear, and it is possible to write $E_r = \bigcup_{j=1}^{l_r}(k_r\mathbb{N} + d_{r,j}) \cup D_r$, i.e., all the linear sets constituting E_r can be reduced to a common period k_r, and an additional finite set. Then, we can take a common multiple k of periods k_r, and represent each checking set as $E_r = \left(k\mathbb{N}_+ + \{d'_{r,j} \mid 1 \leq j \leq l'_r\}\right) \cup D'_r$, where D'_r is finite.

Finally, take a number M such that M is a multiple of k, that M is larger than any element of D_r, $1 \leq r \leq q$, that M is larger than any number $d'_{r,j}$, $1 \leq j \leq l'_r$, $1 \leq r \leq q$, that M is larger than any of i_r and $p_{r,n+1}$, $1 \leq r \leq q$. Then, if neuron $n+1$ has N spikes, the following properties hold:

- rule r is applicable if and only if $N \in E_r$ in case when $i_r \leq N < M$, and if and only if $M + (N \mathrm{mod\ M}) \in \mathrm{E_r}$ in case when $N \geq M$,
- the difference between the number of spikes in neuron $n+1$ in two successive configurations is not larger than M.

For neuron $n+1$, $Mk + j$ spikes (where $0 \leq j \leq M-1$) will be represented by value k of register 1 and state j.

We simulate Π by a register machine R with one register and an output tape of m symbols. Before we proceed, we need to remark that, without restricting the generality, we may have an arbitrary set of "next instructions" instead of $\{l_2, l_3\}$ in $l_1 : (ADD(r), l_2, l_3)$, and arbitrary sets of "next instructions" instead of $\{l_2\}$ and $\{l_3\}$ in $l_1 : (SUB(r), l_2, l_3)$. Indeed, non-determinism between choice of multiple instructions can be implemented by an increment followed by a decrement in each case, as many times as needed for the corresponding set of "next instructions". Clearly, $l_1 : (ADD(r), \{l_2\})$ is just a shorter form of $l_1 : (ADD(r), l_2, l_2)$.

Finally, besides instructions $ADD(r)$, $SUB(r)$, $write(a)$ and $halt$, we introduce the notation of NOP, meaning only a switch to a different instruction without modifying the register. This will greatly simplify the construction below, and such a notation can be reduced to either compressing the rules (by substituting the instruction label with the label of the next instruction in all other instructions), or be simulated by an $ADD(1)$ instruction, followed by a $SUB(1)$ instruction.

We take $b(m_{n+1} \mathrm{mod\ M})$ as the starting state of R, and the starting value of register 1 is $m_{n+1} \mathrm{div\ M}$.

For every class modulo M, $0 \leq j \leq M-1$, we define sets

$$L_{j,0} = \{l_{r,0} \mid 1 \leq r \leq q,\ j \in E_r,\ i_r \geq j\},$$
$$L_{j,+} = \{l_{r,+} \mid 1 \leq r \leq q,\ j + M \in E_r\}$$

of applicable rules corresponding to remainder j, subscripts 0 and + represent cases of having less than M spikes, and at least M spikes, respectively. Let us redefine any of these sets to $\{l_h\}$ if the expression above is empty.

We proceed with the actual simulation. A rule

$$(n+1, E_r/i_r \to \{(1, p_{r,1}), \cdots, (n, p_{r,n}), (n+1, p_{r,n+1})\})$$

can be simulated by the following rules of R:

$b(j) : (S(1), L_{j,+}, L_{j,0}),\ l_r \in L_{j,0};$
$l_{r,\alpha} : \cdots,$ (a sequence of p_{r_1} instructions $write(a_1), \cdots,$)
$\cdots, (p_{r_n}$ instructions $write(a_n)),$
$\cdots l'_{r,\alpha},$ (and $p_{r_{n+1}}$ instructions $ADD(1)), \alpha \in \{0, +\};$
$l'_{r,+} : (NOP, \{b((j - i_r + p_{r,n+1})\text{mod M})\}),\ \text{if } \mathrm{j - i_r + p_{r,n+1}} < 0;$
$l'_{r,+} : (ADD(1), \{l'_{r,0}\}),\ \text{if } j - i_r + p_{r,n+1} < M;$
$l'_{r,0} : (NOP, \{b((j - i_r + p_{r,n+1})\text{mod M})\}),\ \text{if } \mathrm{j - i_r + p_{r,n+1}} < \mathrm{M};$
$l'_{r,0} : (ADD(1), \{b((j - i_r + p_{r,n+1})\text{mod M})\}),\ \text{if } \mathrm{j - i_r + p_{r,n+1}} \geq \mathrm{M};$
$l_h : halt.$

Indeed, instruction $b(j)$ corresponds to checking whether neuron $n+1$ has at least M spikes, transitioning into the halting instruction, or into the set of instructions associated with the corresponding applicable rules, in the context of the result of the checking mentioned above. Sending spikes to output neurons is simulated by writing the corresponding symbols on the tape. This goal is obtained, knowing values j, i_r, $p_{r,n+1}$, and whether neuron 1 had at least M spikes or not, by transitioning to instruction $b((j - i_r + p_{r,n+1})\text{mod M})$ after incrementing register 1 the needed number of times (0, 1 or 2), which is equal to $(j - i_r + p_{r,n+1}\text{div M}) + d$, where $d = 0$ if neuron 1 had at least M spikes, and $d = 1$ otherwise (to compensate for the subtraction done by instruction $b(j)$ in the initial checking). The simulation of instructions continues until we reach the situation where no rules of the underlying spiking system are applicable, transitioning to some $L_{j,\alpha} = \{l_h\}$.

Finally, let us formally describe the instruction sequences from $l_{r,\alpha}$ to $l'_{r,\alpha}$. For the sake of simplicity of notation, we do not mention subscripts r, α in the notation of the intermediate instructions, keeping in mind that these are different instructions for different r, α. The difficulty for generating the string languages is that, by the definition, all permutations are to be considered if spikes are sent to multiple neurons $1, \cdots, m$.

$l_{r,\alpha} : (NOP, \{s(p_{r_1}, \cdots, p_{r_n})\});$
$s(i_1, \cdots, i_n) : (NOP, \{s^k(i_1, \cdots, i_n) \mid i_k > 0,\ 1 \leq k \leq n\}),$
$0 \leq i_j \leq p_{r_j},\ 1 \leq j \leq n, (i_1, \cdots, i_n) \neq (0, \cdots, 0);$
$s^{(k)}(i_1, \cdots, i_n) : (write(a_k), \{s(i'_1, \cdots, i'_n)\}),$
$i'_k = i_k - 1,$ and $i'_j = i_j,\ 1 \leq j \leq n,\ j \neq k,$
$0 \leq i_j \leq p_{r_j},\ 1 \leq j \leq n, (i_1, \cdots, i_n) \neq (0, \cdots, 0);$
$s(0, \cdots, 0) : (NOP, \{t(p_{r_{n+1}})\});$
$t(i) : (ADD(n+1), t(i-1)),\ 1 \leq i \leq p_{r_{n+1}};$
$t(0) : (NOP, l'_{r,\alpha}).$

The rules above describe precisely the following behavior: to produce any sequence with the desired numbers of occurrences of symbols $a_1, \cdots, a_n$, a symbol is non-deterministically chosen (out of those, the current desired number of occurrences of which is positive) and written, iterating until all desired symbols are written.

Next, the register is incremented the needed number of times. This finishes the explanation of the instruction sequences from $l_{r,\alpha}$ to $l'_{r,\alpha}$, as well as the explanation of the simulation.

Therefore, the class of languages generated by ESNP systems with only one neuron containing rules and n output neurons is included in the class of languages generated by 1-register machines with an output tape of n symbols.

Applying Parikh mapping to both classes, just replacing *write*-instructions by *ADD*-instructions on new registers associated with these symbols, it follows that the class of sets of vectors generated by ESNP systems with only one neuron containing rules and n output neurons is included in the class of sets of vectors generated by $n+1$-register machines where all registers except one are restricted to be increment-only. These observations conclude the proof. □

The inclusions formulated at the end of the proof given above are actually characterizations, as we can also prove the opposite inclusion.

Lemma 3. *For any register machine with output tape with only one register that can be decremented respectively for any register machine with only one register that can be decremented we can effectively construct an ESNP system generating the same language respectively the same set of (vectors of) natural numbers.*

Proof. By definition, output registers can only be incremented, so the main computational power lies in the register which can also be decremented. The decrementable register can be simulated together with storing the actual state by storing the number $dn + c_i$ where: n is the actual contents of the register, c_i is a number encoding the i-th instruction of the register machine, and d is a number bigger than all c_i. Then incrementing this first register by an instruction c_i and jumping to c_j means consuming c_i and adding $d + c_j$ in the actor neuron, provided the checking set guarantees that the actual contents is an element of $d\mathbb{N} + c_i$. Decrementing means consuming $d + c_i$ and adding c_j in the actor neuron, provided the checking set guarantees that the actual contents is an element of $d\mathbb{N}_+ + c_i$; if $n = 0$, then c_i is consumed and c_k is added in the actor neuron with c_k being the instruction to continue in the zero case. At the same time, with each of these simulation steps, the output neurons can be incremented in the exact way as the output registers; in the case of register machines with output tape, a spike is sent to the output neuron representing the symbol to be written. Further details of this construction are left to the reader. □

4 ESNP Systems with White Hole Rules

In this section, we extend the model of extended spiking neural P systems, introduced in [1] and described in the previous section, by white hole rules. We will show that with this new variant of extended spiking neural P systems, computational completeness can already be obtained with only one actor neuron, by proving that the computations of any register machines can already be simulated in only one neuron equipped with the most general variant of white hole rules. Using this single actor neuron to also extract the final result of a computation, we even obtain weak universality with only one neuron.

As already mentioned in Remark 1, we are going to describe the checking sets and the number of spikes by non-negative integers. The following definition is an extension of Definition 1:

Definition 3. *An* extended spiking neural P system with white hole rules *(of degree $m \geq 1$) (in the following we shall simply speak of an* EESNP 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;*
- *S describes the* initial configuration *by assigning an initial value (of spikes) to each neuron;*
- *R is a finite set of* rules *either being a* white hole rule *or a rule of the form as already described in Definition 3 $(i, E/k \to P; d)$ 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 P is a (possibly empty) set 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* white hole rule *is of the form $(i, E/all \to P; d)$ where all means that the whole contents of the neuron is taken out of the neuron; in the productions (l, w, t), either $w \in \mathbb{N}$ as before or else $w = (all + p) \cdot q + z$ with $p, q, z \in \mathbb{Q}$; provided $(c + p) \cdot q + z$, where c denotes the contents of the neuron, is non-negative, then $\lfloor (c + p) \cdot q + z \rfloor$ is the number of spikes put on the axon to neuron l.*
  If the checking sets in all rules are finite, then Π is called a finite EESNP system.

Allowing the white hole rules having productions being of the form $w = (all + p) \cdot q + z$ with $p, q, z \in \mathbb{Q}$ is a very general variant, which can be restricted in many ways, for example, by taking $z \in \mathbb{Z}$ or omitting any of the rational numbers $p, q, z \in \mathbb{Q}$ or demanding them to be in $\mathbb{N}$ etc.

Obviously, every ESNPS also is an EESNPS, but without white hole rules, and a finite EESNPS also is a finite ESNPS, as in this case the effect of white hole rules is also bounded, i.e., even with allowing the use of white hole rules, the following lemma as a counterpart of Lemma 1 is still valid:

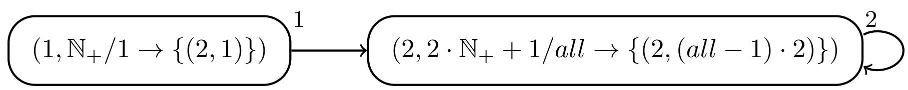
Lemma 4. *For any EESNP system where during any computation only a bounded number of spikes occurs in the actor neurons, the generated language is regular.*

Hence, in the following our main interest is in EESNPS which really make use of the whole power of white hole rules.

4.1 Examples for EESNPS

EESNPS can also be used for computing functions, not only for generating sets of (vectors of) integer numbers. As a simple example, we show how the function $n \mapsto 2^{n+1}$ can be computed by a deterministic EESPNS, which only has exactly one rule in each of its two neurons; the output neuron 2 in this case is not free of rules.

Example 2. Computing $n \mapsto 2^{n+1}$

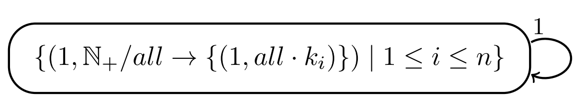


Initial value $= n$ Initial value $= 2$

The rule $(2, 2 \cdot \mathbb{N}_+ + 1/all \to \{(2, (all-1) \cdot 2)\})$ could also be written as $(2, 2 \cdot \mathbb{N}_+ + 1/all \to \{(2, (all) \cdot 2 - 2)\})$. In both cases, starting with the input number n (of spikes) in neuron 1, with each decrement in neuron 1, the contents of neuron 2 (not taking into account the enabling spike from neuron 1) is doubled. The computation stops with 2^{n+1} in neuron 1, as with 0 in neuron 1 no enabling spike is sent to neuron 2 any more, hence, the firing condition is not fulfilled any more. We finally remark that with the initial value 1 in neuron 2 we can compute the function $n \mapsto 2^n$.

Example 3. Pure White Hole Model of EESNPS for DT0L Systems

Let $G = (\{a\}, P, a^s)$ be a Lindenmayer system with the axiom a^s and the finite set of tables P each containing a finite set of parallel productions of the form $a \to a^k$. Such a system is called a tabled Lindenmayer system, abbreviated $T0L$ system, and it is called deterministic, abbreviated $DT0L$ system, if each table contains exactly one rule. Now let $G = (\{a\}, P, a^s)$ be a $DT0L$ system with $P = \{\{a \to a^{k_i}\} \mid 1 \leq i \leq n\}$. Then the following EESNPS using only white hole rules computes the same set of natural numbers as are represented by the language generated by G, with the results being taken with *unconditional halting*, i.e., taking a result at every moment (see [2]).



Initial value $= s$

If we want to generate with normal halting, we have to add an additional output neuron 2 and an additional rule $\{(1, \mathbb{N}_+/all \rightarrow \{(2, all \cdot 1)\})\}$ in neuron 1 which at the end moves the contents of neuron 1 to neuron 2.

4.2 Universality with EESNPS

Lemma 5. *The computation of any register machine can be simulated in only one single actor neuron of an EESPNS.*

Proof. Let $M = (n, P, l_0, l_h)$ be an n-register machine, where n is the number of registers, P is a finite set of instructions injectively labelled with elements from a set of labels $Lab\,(M)$, l_0 is the initial label, and l_h is the final label.

Then we can effectively construct an EESNPS $\Pi = (m, S, R)$ simulating the computations of M by encoding the contents n_i of each register i, $1 \leq i \leq n$, as $p_i^{n_i}$ for different prime numbers p_i. Moreover, for each instruction (label) j we take a prime number q_j, of course, also each of them being different from each other and from the p_i.

The instructions are simulated as follows:

- $l_1 : (ADD\,(r)\,, l_2, l_3)$ (ADD instruction)
  This instruction can be simulated by the rules
  $\{(1, q_{l_1} \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot q_{l_i} p_r/q_{l_1})\}) \mid 2 \leq i \leq 3\}$
  in neuron 1.
- $l_1 : (SUB\,(r)\,, l_2, l_3)$ (SUB instruction)
  This instruction can be simulated by the rules
  $(1, q_{l_1} p_r \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot q_{l_2}/\,(q_{l_1} p_r))\})$
  and
  $(1, q_{l_1} \cdot \mathbb{N}_+ \setminus q_{l_1} p_r \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot q_{l_2}/q_{l_1})\})$
  in neuron 1; the first rule simulates the decrement case, the second one the zero test.
- $l_h : halt$ (HALT instruction)
  This instruction can be simulated by the rule
  $(1, q_{l_h} \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot 1/q_{l_h})\})$
  in neuron 1.
  In fact, after the application of the last rule, we end up with $p_1^{m_1} \cdots p_n^{m_n}$ in neuron 1, where $(m_1, \cdots, m_n)$ is the vector computed by M and now, in the prime number encoding, by Π as well.

All the checking sets we use are regular, and the productions in all the white hole rules even again yield integer numbers. □

Remark 2. As the productions in all the white hole rules of the EESNPS constructed in the preceding proof even again yield integer numbers, we could also interpret this EESNPS as an ESPNS with exhaustive use of rules:

The white hole rules in the EESNPS constructed in the previous proof are of the general form

$(1, q \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot p/q)\})$

with p and q being natural numbers. Each of these rules can be simulated in a one-to-one manner by the rule

$(1, q \cdot \mathbb{N}_+/q \rightarrow p)$

used in an ESNPS with one neuron in the exhaustive way.

Theorem 3. *Any recursively enumerable set of n-dimensional vectors can be generated by an ESNP system with $n+1$ neurons.*

Proof. We only have to show how to extract the results into the additional output neurons from the single actor neuron which can do the whole computational task as exhibited in Lemma 5. Yet this is pretty easy:

When the actor neuron reaches the halting state, the desired result m_i for output neuron $i+1$ is stored as factor in this one number stored in the actor neuron within the prime number encoding, i.e., as $q_i^{m_i}$, for $1 \leq i \leq n$. Instead of using the final rule $(1, q_{l_h} \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot 1/q_{l_h})\})$ in neuron 1 we now take the rule $(1, q_{l_h} \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot r_1/q_{l_h})\})$.

With the rules $(1, r_i q_i \mathbb{N}_+/all \rightarrow \{(1, all \cdot 1/k_i), (i+1, 1)\})$, we can decode the factor $q_i^{m_i}$ to m_i into output neuron $i+1$, with the instruction code (prime number) r_i for $1 \leq i \leq n$. If the contents of the actor neuron is not dividable by q_i any more, we switch to the next instruction code r_{i+1} by the rule $(1, r_i \cdot \mathbb{N}_+ \setminus r_i q_i \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot r_{i+1}/r_i)\})$. At the end, we can end up with 0 in the actor neuron after having used the rule $(1, r_i \cdot \mathbb{N}_+ \setminus r_i q_i \cdot \mathbb{N}_+/all \rightarrow \emptyset)$ and then stop with m_i in output neuron $i+1$, $1 \leq i \leq n$. □

Theorem 4. *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 $n+1$ neurons.*

Proof. In the case of generating strings, we have to simulate a register machine with output tape; hence, in addition to the simulating rules already described in Lemma 5, we have to simulate the tape rule $l_1 : (write(a), l_2)$, which in the EESNPS means sending one spike to the output neuron $N(a)$ representing the symbol a. This task is accomplished by the rule $(1, l_1 \cdot \mathbb{N}_+/all \rightarrow \{(1, all \cdot l_2/l_1), (N(a), 1)\})$. The rest of the construction and of the proof is similar to that what we have done in the proof of Lemma 5. □

5 Summary and Further Variants

In this paper, we have extended the model of extended spiking neural P systems from [1] by white hole rules. With this new variant of extended spiking neural P systems, computational completeness can already be obtained with only one actor neuron, as the computations of any register machine can already be simulated in only one neuron equipped with the most general variant of white hole rules. Using this single actor neuron to also extract the final result of a computation, we even obtain weak universality with only one neuron.

A quite natural feature found in biology and also already used in the area of spiking neural P systems is that of inhibiting neurons or axons between neurons, i.e., certain connections from one neuron to another one can be specified as inhibiting ones – the spikes coming along such inhibiting axons then close the target neuron for a time period given by the sum of all inhibiting spikes, e.g., see [3]. Such variants can also be considered for extended spiking neural P systems with white hole rules.

References

1. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended spiking neural P systems. In: [11], 123–134.
2. M. Beyreder, R. Freund: Membrane systems using noncooperative rules with unconditional halting. In: D. W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science *5391*, Springer, 2009, 129–136.
3. A. Binder, R. Freund, M. Oswald, L. Vock: Extended spiking neural P systems with excitatory and inhibitory astrocytes. In: M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez (Eds.): *Fifth Brainstorming Week on Membrane Computing*, RGNC REPORT 01/2007, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2007, 63–72.
4. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In: [11], 169–194
5. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer, 1989.
6. H. Fernau, R. Freund, M. Oswald, K. Reinhardt: Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. *Journal of Automata, Languages and Combinatorics*, **12** (1-2) (2007), 117–138.
7. 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*, Curtea de Argeş, Romania. Lecture Notes in Computer Science **2597**, Springer, Berlin, 2002, 261–268.
8. R. Freund, Gh. Păun: From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science* **312**, 143–188.
9. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2004), 101–116.
10. W. Gerstner, W. Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.
11. M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero (Eds.): *Fourth Brainstorming Week on Membrane Computing*, Vol. I RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2006.

12. M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero (Eds.): *Fourth Brainstorming Week on Membrane Computing*, Vol. II RGNC REPORT 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2006.
13. O.H. Ibarra, A. Păun, Gh. Păun Gh, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. In: [12], 105–136, 2006.
14. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae* **71** (2 –3) (2006), 279–308.
15. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK* **8** (1) (2002), 32–36.
16. W. Maass, C. Bishop (Eds.): *Pulsed Neural Networks.* MIT Press, Cambridge, 1999.
17. Martín-Vide C, Pazos J, Păun Gh, Rodríguez-Patón A (2002) 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, 290–299
18. M. L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
19. L. Pan, Gh. Păun: Spiking Neural P Systems with Anti-Matter. *International Journal of Computers, Communications & Control* **4** (3), 273–282 (2009).
20. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
21. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
22. Păun Gh, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.* **17** (2006), 975–1002.
23. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen* **60** (2006), 635–660.
24. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
25. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.
26. X. Zhang, B. Luo, X. Fang, L. Pan: Sequential spiking neural P systems with exhaustive use of rules. *BioSystems* **108** (2012), 52–62.
27. X. Zhang, X. Zeng, L. Pan: On String Languages Generated by Spiking Neural P Systems with Exhaustive Use of Rules. *Natural computing* **7** (4) (2002), 535–549.
28. The P Systems Website: `www.ppage.psystems.eu`.