Computing Along the Axon

Haiming Chen[1], Tseren-Onolt Ishdorj[2], Gheorghe Păun[2,3]

[1] Computer Science Laboratory, Institute of Software
Chinese Academy of Sciences
Beijing, 100080 China
chm@ios.ac.cn

[2] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
tserren@yahoo.com

[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
george.paun@imar.ro, gpaun@us.es

Summary. We consider a special form of spiking neural P systems, called axon P systems, corresponding to the activity of Ranvier nodes of neuron axon, and we briefly investigate the language generative power of these devices

1 Introduction

Recently, ideas from neural computing based on spiking were incorporated in membrane computing in the form of spiking neural P systems (for short, SN P systems), see [1], [4] [5], etc. In SN P systems, the main "information-processor" is the neuron, while the axon is only a channel of communication, without any other role – which is not exactly the case in neuro-biology.

In the present paper we introduce a class of SN-like P systems, where the computation is done along the axon (this time we ignore the neurons). Actually, a sort of linear SN P system is considered, corresponding to the Ranvier nodes of axons. Spikes are transmitted along the axon, to the left and to the right, from a node to another node, and an output is provided by the rightmost node. Specifically, a symbol b_i is associated with a time unit when i impulses (spikes) exit the system, and thus a string is associated with a computation.

The relationships of the families of languages generated in this way with families from Chomsky hierarchy are investigated, then axon P systems with states are considered. Several open problems and research topics are formulated in the end of the paper.

2 The Axon as an Information Processor

We briefly describe here the neural cells structure and functioning and those features that are of interest from a computational point of view and from which we will abstract the (mathematical) features of our computing devices.

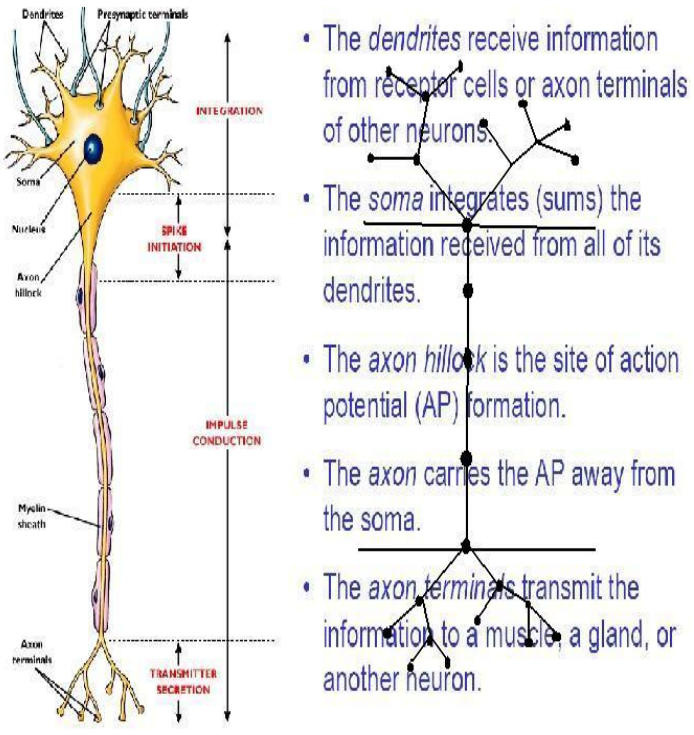
The nervous system consists of about 10^{10} nerve cells, the so-called *neurons*. Each of these neurons is connected to about 10000 other neurons via *synapses*. This gigantic network of neurons and synapses is placed in a small portion of human brain. Neural networks are generally considered as information-processing systems, i.e., as systems which operate transformations of their inputs in order to produce outputs.

Brain cells communicate in a process that begins with an electrical signal and ends with a neurotransmitter binding to a receptor on the receiving neuron. It lasts less than a thousandth of a second, and is repeated billions of times daily in each of the human brain's 100 billion neurons. The main part of the action takes place inside the secreting cell.

The basic transmitting unit in the nervous system is the neuron. The neuron is not a homogeneous integrative unit but is (potentially) divided in many sub-integrative units, each one with the ability of mediating a local synaptic output to another cell or local electro-tonic output to another part of the same cell.

Neurons are considered to have 3 main parts: a soma, the main part of the cell where the genetic material is present and life functions take place; a dendrite tree, the branches of the cell from where the impulses come in; an axon, the branch of the neuron over which the impulse (or signal) is propagated. The branches present at the end of the axons form the terminal tree. An axon can be provided with a structure composed by special sheaths. These sheaths are involved in molecular and structural modifications of axons needed to propagate impulse signals rapidly over long distance. The impulse in effect jumps from node to node, and this form of propagation is therefore called *saltatory conduction*. It is an efficient mechanism that achieves maximum conduction speed with a minimum of active membrane, metabolic machinery, and fiber size. There is a gap between neighboring myelinated regions that is known as the node of Ranvier, which contains a high density of voltage-gated Na^+ channels for impulse generation. When the transmitting impulses reach the node of Ranvier or junction nodes of dendrite and terminal trees, or the end bulbs of the trees, it causes the change in polarization of the membrane. The change in potential can be excitatory (moving the potential toward the threshold) or inhibitory (moving the potential away from the threshold). In Figure 1, the neuron structure is illustrated.

The impulse transmission through a neuron follows this path: from dendrite to soma to axon to terminal tree, and then to synapse. If different impulses reach at the same time a certain node, then it might happen that the combined effects of the *exciton and inhibition may cancel each other*. Once the threshold of the membrane potential is reached, an impulse is propagated along the neuron or to the next neuron.



- The *dendrites* receive information from receptor cells or axon terminals of other neurons.
- The *soma* integrates (sums) the information received from all of its dendrites.
- The *axon hillock* is the site of action potential (AP) formation.
- The *axon* carries the AP away from the soma.
- The *axon terminals* transmit the information to a muscle, a gland, or another neuron.

Fig. 1. A neuron structure

Some of the previously mentioned ideas will be incorporated (abstracted) in our computing model, defined in Section 4. More details about neural biology, can be found in the classical book [8].

3 Formal Language Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from [6] and [7], so that we introduce here only some notations and notions used later in the paper. For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The length of a string $x \in V^*$ is denoted by $|x|$. If $x = a_1a_2 \ldots a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n \ldots a_2a_1$.

A Chomsky grammar will be given in the form $G = (N, T, S, P)$, with N being the nonterminal alphabet, T the terminal alphabet, $S \in N$ the axiom, and P the set of rules. We denote by REG, LIN, CF, CS, RE the families of languages generated by regular, linear, context-free, context-sensitive, and of arbitrary grammars, respectively (RE stands for recursively enumerable languages). By FIN we denote the family of finite languages. The following strict inclusions hold:

$$FIN \subset REG \subset LIN \subset CF \subset CS \subset RE.$$

This is the Chomsky hierarchy.

Regular languages are defined (among many other possibilities) by means of *regular expressions*. 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 $(E_1) \cup (E_2)$, $(E_1)(E_2)$, $(E_1)^+$ 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 from $(E)^*$.

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

We will also invoke below the family MAT, of languages generated by matrix grammars without appearance checking (see [2] for details) and the family REC, of recursive languages (languages whose membership problem is decidable).

4 Axon P Systems

We pass directly to considering the device we investigate in this paper, incorporating in the spiking neural P systems framework the way the axon processes information, as described in Section 2.

An *axon P system* of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \rho_1, \ldots, \rho_m),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\rho_1, \ldots, \rho_m$ are (Ranvier) *nodes*, of the form

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

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in ρ_i;
   b) R_i is a finite set of *rules* of the form $E/a^c \to (a^l, a^r)$, where E is a regular expression over a, $c \geq 1$, and $l, r \geq 0$, with the restriction that R_1 contains only rules with $l = 0$.

The intuition is that the nodes are arranged along an axon in the order $\rho_1, \ldots, \rho_m$, with ρ_m at the end of the axon, hence participating to synapses (this is a way to say that node ρ_m is the *output* one of the system).

A rule $E/a^c \to (a^l, a^r)$ is used as follows. If the node ρ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can be applied, and this means that c spikes are removed from ρ_i (thus only $k - c$ remain in ρ_i), the node is fired, and it sends l spikes to its left hand neighbor and r spikes to its right hand neighbor; the first node, ρ_1 does not send spikes to the left, while in the case of the rightmost node, ρ_m, the spikes sent to the right are "lost" in the environment. The system is synchronized, a global clock is assumed, marking the time for all nodes.

If a rule $E/a^c \to (a^l, a^r)$ has $E = a^c$, then we will write it in the simplified form $a^c \to (a^l, a^r)$.

In each time unit, if a node ρ_i can use one of its rules, then a rule from R_i *must* be used. Since two rules $E_1/a^{c_1} \to (a^{l_1}, a^{r_1})$ and $E_2/a^{c_2} \to (a^{l_2}, a^{r_2})$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a node, and in that case, only one of them is chosen non-deterministically.

During the computation, a configuration is described by the number of spikes present in each node. The initial configuration is $C_0 = \langle n_1, \ldots, n_m \rangle$.

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation.* A computation halts if it reaches a configuration where no rule can be used. With any computation (halting or not) we associate a sequence of symbols by associating the symbol b_i with a step when the system outputs i spikes, with b_0 indicating the steps when no spike is emitted from ρ_m to the environment. When the computation is halting, this sequence is finite.

Let us denote by $L(\Pi)$ the language of strings computed as above by halting computations of the system Π and let $LAP_m(rule_k, cons_p)$ the family of languages $L(\Pi)$, generated by systems Π with at most m nodes, each node having at most k rules, and each of these rules consuming at most p spikes. As usual, a parameter m, k, p is replaced with $*$ if it is not bounded.

5 Examples

We consider here some simple axon P systems, given in the graphical form, following the style of spiking neural P systems: we specify the nodes along the axon, with two way arrows among them and with an arrow which exits from the output node, pointing to the environment; in each node we give the rules and the spikes present in the initial configuration.

Figure 2 presents the initial configuration of the system Π_1. We have two nodes, with node ρ_2 containing one spike. This spike will circulate among the two nodes as long as the second node uses the rule $a \to (a, a)$. In this way, in every second step one outputs a spike, starting with the first step of the computation. When

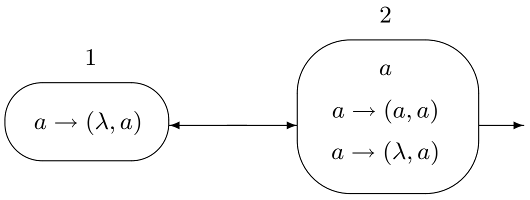


Fig. 2. The initial configuration of system Π_1

node ρ_2 uses the rule $a \to (\lambda, a)$ no spike will remain inside and the computation halts. Therefore, $L(\Pi_1) = (b_1 b_0)^* b_1$.

Consider also the system Π_2 from Figure 3. This time each node starts with a spike, hence the two nodes can interchange a spike as long as ρ_1 uses the rule $a \to (\lambda, a)$ and ρ_2 uses one of the rules $a \to (a, a)$, $a \to (a, \lambda)$. If at some moment the two nodes use simultaneously the other rules, then no spike remains in the system and the computation halts. In this way, one generates all strings in $\{b_0, b_1\}^+$, hence we have $L(\Pi_2) = \{b_0, b_1\}^+$.

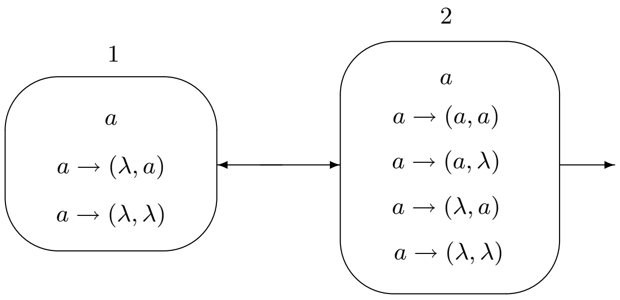


Fig. 3. The initial configuration of system Π_2

Clearly, this system can be extended to a system which generates the language V^+ for any alphabet V with at least two symbols.

The third example is slightly more complex: Π_3, from Figure 4. It has $n + 1$ nodes, for some given $n \geq 2$. The leftmost node has only one rule, the rightmost node contains the rules $a \to (a^2, a)$ and $a \to (\lambda, a)$, and all other nodes contain the rules $a \to (\lambda, a)$ and $a^2 \to (a^2, \lambda)$. We start with one spike in node ρ_{n+1}. This spike is moved continuously to the left and to the right of axon, and always when it arrives in the rightmost node a spike exits the system. The computation can be finished by the rightmost node, by using the rule $a \to (\lambda, a)$, which leaves the system without any spike inside. Consequently, we have $L(\Pi_3) = \{b_1(b_0^{2n-1}b_1)^* \mid n \geq 1\}$.

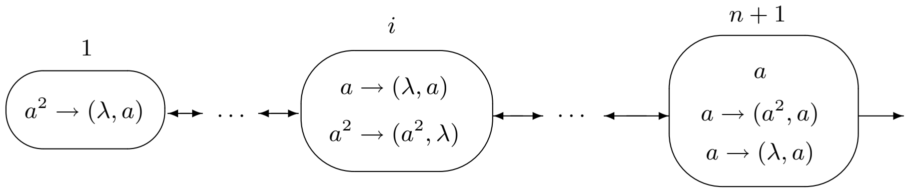


Fig. 4. The initial configuration of system Π_3

We do not present further examples, because the results in the next section are also based on effective constructions of axon P systems.

6 The Generative Power of SN P Systems

6.1 A Characterization of FIN

Lemma 1. *$LAP_1(rule_*, cons_*) \subseteq FIN$.*

Proof. In each step, the number of spikes present in an axon P system with only one node decreases by at least one, hence any computation lasts at most as many steps as the number of spikes present in the system at the beginning. Thus, the generated strings have a bounded length.

Lemma 2. *$FIN \subseteq LAP_1(rule_*, cons_*)$.*

Proof. Let $L = \{x_1, x_2, \ldots, x_n\} \subseteq V^*$, $n \geq 1$, be a finite language for some $V = \{b_1, \ldots, b_k\}$, $k \geq 1$. Let $x_i = x_{i,1} \ldots x_{i,r_i}$, $1 \leq i \leq n$. For $b \in V$, define $index(b) = i$ if $b = b_i$. Denote $\alpha_j = \sum_{i=1}^{j} |x_i|$, for all $1 \leq j \leq n$.

An axon P system that generates L is shown in Figure 5.

Initially, only a rule $a^{\alpha_n+1}/a^{\alpha_n+1-\alpha_j} \to a^{index(x_{j,1})}$ can be used, and in this way we non-deterministically chose the string x_j to generate. This rule outputs the necessary number of spikes for $x_{j,1}$. Then, because α_j spikes remain in the neuron, we have to continue with rules $a^{\alpha_j-t+2}/a \to a^{index(x_{j,t})}$, for $t = 2$, and then for the respective $t = 3, 4, \ldots, r_j - 1$; in this way we introduce $x_{j,t}$, for all $t = 2, 3, \ldots, r_j - 1$. In the end the rule $a^{\alpha_j - r_j + 2} \to a^{index(x_{j,r_j})}$ is used, which produces x_{j,r_j} and concludes the computation.

It is easy to see that the rules which are used in the generation of a string x_j cannot be used in the generation of a string x_k with $k \neq j$. Also, in each rule the spikes consumed are not less than the spikes produced. □

Theorem 1. *$FIN = LAP_1(rule_*, cons_*)$.*

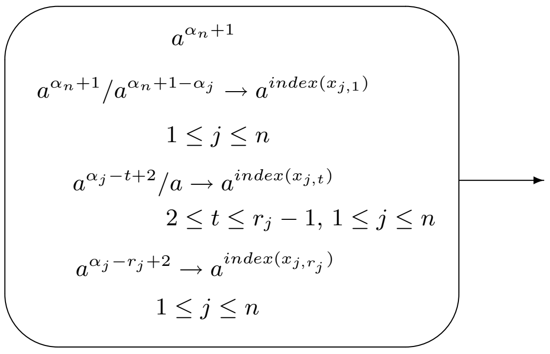

a^{α_n+1}
$a^{\alpha_n+1}/a^{\alpha_n+1-\alpha_j} \to a^{index(x_{j,1})}$
$1 \leq j \leq n$
$a^{\alpha_j-t+2}/a \to a^{index(x_{j,t})}$
$2 \leq t \leq r_j - 1,\ 1 \leq j \leq n$
$a^{\alpha_j-r_j+2} \to a^{index(x_{j,r_j})}$
$1 \leq j \leq n$

Fig. 5. An axon P system generating a finite language

6.2 Relationships with REG

Theorem 2. *$REG \subseteq LAP_2(rule_*, cons_*)$.*

Proof. For $L \in REG$, consider a grammar $G = (N, B, S, P)$ such that $L = L(G)$, where $N = \{A_1, A_2, \ldots, A_n\}$, $n \geq 1$, $S = A_n$, $V = \{b_1, \ldots, b_m\}$, and the rules in P are of the forms $A_i \to b_k A_j$, $A_i \to b_k$, $1 \leq i, j \leq n$, $1 \leq k \leq m$.

Then L can be generated by an axon P system as shown in Figure 6.

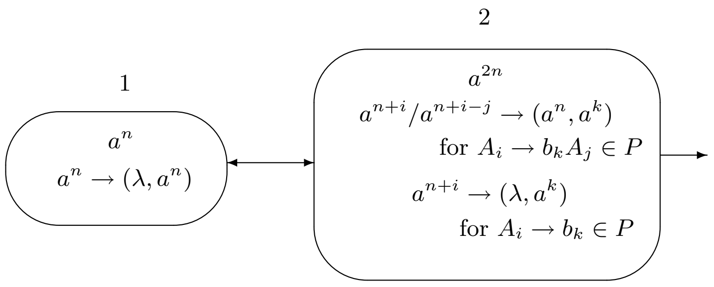

1
a^n
$a^n \to (\lambda, a^n)$
2
a^{2n}
$a^{n+i}/a^{n+i-j} \to (a^n, a^k)$
for $A_i \to b_k A_j \in P$
$a^{n+i} \to (\lambda, a^k)$
for $A_i \to b_k \in P$

Fig. 6. An axon P system generating a regular language

In the first step, node ρ_2 fires by a rule $a^{2n}/a^{2n-j} \to (a^n, a^k)$ (or $a^{2n} \to (\lambda, a^k)$) associated with a rule $A_n \to b_k A_j$ (or $A_n \to b_k$) from P, and sends k spikes to the environment. In this step node ρ_1 also fires and sends n spikes to node ρ_2. It will send n spikes back to node ρ_2 as long as it receives n spikes from node ρ_2.

Assume that in some step t, the rule $a^{n+i}/a^{n+i-j} \to (a^n, a^k)$, for $A_i \to b_k A_j$, or $a^{n+i} \to (\lambda, a^k)$, for $A_i \to b_k$, is used, for some $1 \leq i \leq n$, and n spikes are received from node ρ_1

If the first rule is used, then n spikes are sent to node ρ_1, k spikes are sent to the environment, $n+i-j$ spikes are consumed, and j spikes remain in node ρ_2. Then in step $t+1$, we have $n+j$ spikes in node ρ_2, and a rule for $A_j \to b_k A_l$ or $A_j \to b_k$ can be used. In step $t+1$ node ρ_2 also receives n spikes. In this way, the computation continues.

If the second rule is used, then no spike is sent to node ρ_1, k spikes are sent to the environment, all spikes in node ρ_2 are consumed, and n spikes are received in node ρ_1. Then the computation halts.

In this way, all the strings in L can be generated. □

Actually, as we will see in the next section, the inclusion above is proper.

6.3 Beyond REG

Theorem 3. *$LAP_m(rule_k, cons_p) - REG \neq \emptyset$ for all $m \geq 2, k \geq 3, p \geq 3$.*

Proof. An example of a non-regular language generated by an axon P system of the complexity mentioned in the theorem is presented in Figure 7. As long as the first rule of R_1 is used, no spike is output, and node ρ_1 accumulates continuously two more spikes. After n steps of this type, node ρ_2 will contain $2(n+1)+1$ spikes. At any step, node ρ_1 can use the rule $a(aa)^+/a^3 \to (\lambda, a^2)$. This both leaves an even number of spikes in node ρ_1, and sends two spikes to node ρ_2. The number of spikes from node ρ_1 becomes $2(n+2)$ (it receives four spikes and consumes three). In the next step, we do not output any spike, but from now on we begin output spikes in each step: node ρ_1 uses the rule $(aa)^+/a^2 \to (\lambda, a^3)$, thus continuously decreasing by two the number of spikes it contains. This can be done for $n+2$ steps, hence the generated string is $b_0^{n+2} b_1^{n+2}$, that is, $L(\Pi) = \{b_0^n b_1^n \mid n \geq 2\}$. This is not a regular language. □

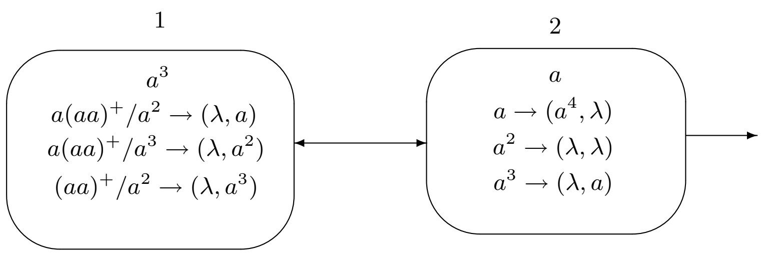

1
2
a^3
$a(aa)^+/a^2 \to (\lambda, a)$
$a(aa)^+/a^3 \to (\lambda, a^2)$
$(aa)^+/a^2 \to (\lambda, a^3)$
a
$a \to (a^4, \lambda)$
$a^2 \to (\lambda, \lambda)$
$a^3 \to (\lambda, a)$

Fig. 7. An axon P system generating a non-regular language

Also much more complex languages can be generated:

Theorem 4. *The family $LAP_2(rule_3, cons_3)$ contains non-semilinear languages.*

Proof. Consider the axon P system from Figure 8. Assume that we start from a configuration of the form $\langle 3^n+1, 0\rangle$; initially, this is the case, with $n=1$. As long as at least four spikes are present in node ρ_1, the rule $a(aaa)^+/a^3 \to (\lambda, a^9)$ is used and it moves all spikes to the second node, multiplied by 3. When we remain with only one spike in node ρ_1, we can use one of the other two rules of R_1.

If we use $a \to (\lambda, a)$, then in the second node we get a number of spikes of the form $3m+1$, hence the first rule is applied as much as possible, thus returning the spikes to node ρ_1. In the end, we have to use the rule $a \to (a, \lambda) \in R_2$, which makes again the number of spikes from node ρ_1 to be of the form $3m+1$ (note that no rule can be applied in any node when the number of spikes is multiple of 3). This process can be iterated any number of times, thus multiplying by 3 the number of spikes present in node ρ_1.

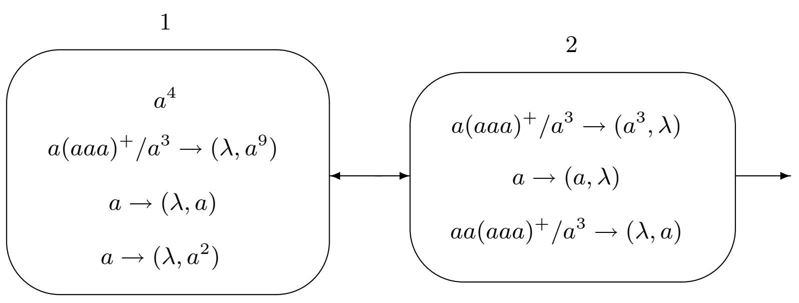

1
a^4
$a(aaa)^+/a^3 \to (\lambda, a^9)$
$a \to (\lambda, a)$
$a \to (\lambda, a^2)$
2
$a(aaa)^+/a^3 \to (a^3, \lambda)$
$a \to (a, \lambda)$
$aa(aaa)^+/a^3 \to (\lambda, a)$

Fig. 8. An axon P system generating a non-semilinear language

At any time, node ρ_1 can also use the rule $a \to (\lambda, a^2)$ instead of $a \to (\lambda, a)$. This makes the number of spikes from node ρ_2 to be of the form $3m+2$, hence the rule $aa(aaa)^+/a^3 \to (\lambda, a)$ should be applied. This rule does not change the 3-arity of the number of spikes, hence it is used as much as possible. In this way, a spike exits the system in each step, until exhausting the spikes from the output node (when only two spikes remain inside, no rule can be used).

This means that after a number of steps when no spike is sent to the environment, we output spikes for 3^n steps, for some $n \geq 1$. Actually, counting the number of steps for accumulating 3^{n+1} spikes in node ρ_2, we conclude that we have $L(\Pi) = \{b_0^{3^n+2n-3} b_1^{3^n} \mid n \geq 1\}$, which, obviously, is not a semilinear language. □

The previous language is not in MAT. However, as we will immediately see, this kind of systems has strong limitations.

Lemma 3. *The number of configurations reachable after n steps by an axon P system of degree m is bounded by a polynomial $g(n)$ of degree m.*

Proof. Let us consider an axon P system $\Pi = (O, \rho_1, \ldots, \rho_m)$ of degree m, let n_0 be the total number of spikes present in the initial configuration of Π, and denote

$\alpha = \max\{l + r \mid E/a^c \to (a^l, a^r) \in R_i, 1 \leq i \leq m\}$ (the maximal number of spikes produced by any of the rules of Π). In each step of a computation, each node ρ_i produces some l and r spikes to br sent to the left and right nodes of node ρ_i, respectively. We have $l + r \leq \alpha$. Each node can do the same, hence the maximal number of spikes produced in one step is at most αm. In n consecutive steps, this means at most αmn spikes. Adding the initial n_0 spikes, this means that after any computation of n steps we have at most $n_0 + \alpha mn$ spikes in Π, hence the number of configurations are no more than $(n_0 + \alpha mn)^m$. This is a polynomial of degree m in n (α is a constants) which bounds from above the number of possible configurations obtained after computations of length n in Π. $\square$

Theorem 5. *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no axon P system Π such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L(\Pi)$.*

Proof. Assume that there is an axon P system Π of degree m such that $L(\Pi) = L_f(V)$ for some f and V as in the statement of the theorem. According to the previous lemma, there are only polynomially many configurations of Π which can be reached after n steps. However, there are $card(V)^n \geq 2^n$ strings of length n in V^+. Therefore, for large enough n there are two strings $w_1, w_2 \in V^+, w_1 \neq w_2$, such that after n steps the system Π reaches the same configuration when generating the strings $w_1\, f(w_1)$ and $w_2\, f(w_2)$, hence after step n the system can continue any of the two computations. This means that also the strings $w_1\, f(w_2)$ and $w_2\, f(w_1)$ are in $L(\Pi)$. Due to the injectivity of f and the definition of $L_f(V)$ such strings are not in $L_f(V)$, hence the equality $L_f(V) = L(\Pi)$ is contradictory. $\square$

Corollary 1. *The following languages are not in $LAP_*(rule_*, cons_*)$ (in all cases, $card(V) = k \geq 2$):*

$$\begin{aligned} L_1 &= \{x\, mi(x) \mid x \in V^+\}, \\ L_2 &= \{xx \mid x \in V^+\}, \\ L_3 &= \{x\, c^{val_k(x)} \mid x \in V^+\}, c \notin V. \end{aligned}$$

Note that language L_1 above is a non-regular minimal linear one (generated by linear grammars with only one nonterminal symbol), L_2 is context-sensitive non-context-free, and L_3 is non-semilinear.

Theorem 6. *$LAP_*(rule_*, cons_*) \subseteq REC$.*

Proof. This is a direct consequence of the fact that a string of length n is produced by means of a computation of length n; thus, given an axon P system Π and a string x, in order to check whether or not $x \in L(\Pi)$ it is enough to produce all computations of length $|x|$ in Π and to check whether any of them generates the string x. $\square$

7 Axon P Systems with States

In this section, we consider a way to control the rule applications by means of node *states*. Specifically, the rules we are going to use are of the form $sE/a^c \rightarrow (s', a^l, a^r)$, where s, s' are states, E is regular expression, and $c \geq 1, l \geq 0, r \geq 0$.

Formally, an axon P system of degree $m \geq 1$ with states is a construct of the form

$$\Pi = (O, Q, \rho_1, \ldots, \rho_m),$$

where:

1. $O = \{a\}$ is the singleton alphabet;
2. Q is the finite set of states;
3. $\rho_1, \ldots, \rho_m$ are (Ranvier) *nodes*, of the form

$$\rho_i = (s_0, n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $s_0 \in Q$ is the *initial state* of node;
   b) $n_i \geq 0$, is the initial number of spikes in node;
   c) R_i is a finite set of *rules* of the form $sE/a^c \rightarrow (s', a^l, a^r)$, where s is the current state of the node, E is a regular expression over a, $c \geq 1$, and $l, r \geq 0$, with the restriction that R_1 contains only rules with $l = 0$.

   A rule $sE/a^c \rightarrow (s', a^l, a^r)$ from R_i is applied like the rule $E/a^c \rightarrow (a^l, a^r)$, but only if node ρ_i is in state s; after the use of the rule, the state of ρ_i becomes s'.

Let us examine some examples, in order to clarify the definitions and to illustrate the way our devices work.

Consider first the simple system

$$\Pi_1 = (O, Q, \rho_1, \rho_2),$$

where:

1. $O = \{a\}$;
2. $Q = \{s_0, s_1\}$;
3. $\rho_1 = (s_0, 1, R_1)$, $\rho_2 = (s_0, 1, R_2)$;
   $R_1 = \{r_1 : s_0a^+/a \rightarrow (s_0, \lambda, a)\}$;
   $R_2 = \{r_1 : s_0a^+/a \rightarrow (s_0, a^2, \lambda)$;
   $r_2 : s_0a \rightarrow (s_1, \lambda, \lambda)$; $r_3 : s_1a \rightarrow (s_1, \lambda, a)\}$.

It generates the non-regular context-free language

$$L(\Pi_1) = \{b_0^n b_1^n \mid n \geq 1\}.$$

The computation steps are presented in Table 1.

Let us observe on Table 1 how the system works. Initially each node contains one spike (a), and they are in the initial state s_0. Node ρ_1 has only one rule and

step	a	a		step	a	a	
1	$r_1 : a$	$r_2 : a$	b_0	1	$r_1 : a$	$r_1 : a$	b_0
2	–	$r_3 : a$	b_1	2	$r_1 : a^2$	$r_1 : a$	b_0
				3	$r_1 : a$	$r_2 : a$	b_1
				4	–	$r_3 : a$	b_1

step	a	a		step	a	a	
1	$r_1 : a$	$r_1 : a$	b_0	1	$r_1 : a$	$r_1 : a$	b_0
2	$r_1 : a^2$	$r_1 : a$	b_0	2	$r_1 : a^2$	$r_1 : a$	b_0
3	$r_1 : a^3$	$r_1 : a$	b_0	3	$r_1 : a^3$	$r_1 : a$	b_0
4	$r_1 : a^4$	$r_1 : a$	b_0	4	$r_1 : a^4$	$r_1 : a$	b_0
5	$r_1 : a^5$	$r_1 : a$	b_0	.	. . .	. . .	.
6	$r_1 : a^6$	$r_2 : a$	b_0	$n-1$	$r_1 : a^{n-1}$	$r_1 : a$	b_0
7	$r_1 : a^5$	$r_3 : a$	b_1	n	$r_1 : a^n$	$r_2 : a$	b_0
8	$r_1 : a^4$	$r_3 : a$	b_1	$n+1$	$r_1 : a^{n+1}$	$r_3 : a$	b_1
9	$r_1 : a^3$	$r_3 : a$	b_1	$n+2$	$r_1 : a^n$	$r_3 : a$	b_1
10	$r_1 : a^2$	$r_3 : a$	b_1	.	. . .	. . .	.
11	$r_1 : a$	$r_3 : a$	b_1	$2n-1$	$r_1 : a$	$r_3 : a$	b_1
12	–	$r_3 : a$	b_1	$2n$	–	$r_3 : a$	b_1

Table 1. Some computations in Π_1

it never change its state. In turn, node ρ_2 has 2 states and 3 rules; rules r_1 and r_2 apply in state s_0, but once rule r_2 is chosen the state of the node is changed to s_1, and from now on rule r_3 should apply until the computation halts.

It is easy to see that the system generates strings of the form $b_0^n b_1^n$ in $2n$ steps, for some $n \geq 1$. Let us follow the table. We apply rule r_1 in node ρ_2 for a number of times, thus generating symbols b_0 (no spike is sent to the environment); in the meantime, node ρ_1 accumulates continuously spikes growing. Once rule r_2 is chosen (suppose this happens in the n-th step), the node changes its state to s_1 and starts to send the spikes of node ρ_1 to the environment, using the rule r_3.

Let us now consider a system with a more sophisticated functioning, namely

$$\Pi_2 = (O, Q, \rho_1, \rho_2),$$

where:

1. $O = \{a\}$;
2. $Q = \{s_0, s_1, s_1', s_2, s_2', s_3, s_\#\}$;
3. $\rho_1 = (s_0, 0, R_1)$, $\rho_2 = (s_0, 2, R_2)$;
   $R_1 = \{r_1 : s_0 a \to (s_0, \lambda, a^2)\}$,
   $R_2 = \{r_1 : s_0 a^2 \to (s_\#, \lambda, \lambda)$; $r_2 : s_0 a^+/a \to (s_3, \lambda, a)$;
   $r_3 : s_0 a^+/a \to (s_3, \lambda, a^2)$; $r_4 : s_0 a^+/a \to (s_2, a, a^2)$;
   $r_5 : s_0 a^+/a \to (s_1, a, a)$; $r_6 : s_1 a^+/a \to (s_1, a, a)$;
   $r_7 : s_1 a^+/a \to (s_2, a, a^2)$; $r_8 : s_1 a^+/a \to (s_1', \lambda, a^3)$;
   $r_9 : s_1' a^+/a^3 \to (s_3, \lambda, a^3)$; $r_{10} : s_2 a^+/a \to (s_2, a, a^2)$;
   $r_{11} : s_2 a^+/a \to (s_2', \lambda, a^3)$; $r_{12} : s_2' a^+/a^3 \to (s_3, \lambda, a^3)$;
   $r_{13} : s_3 a^+/a \to (s_3, \lambda, a^3)\}$.

The language generated is

$$L(\Pi_2) = \{b_1^n b_2^m b_3^{n+m} \mid n, m \geq 0\},$$

which is in CF. Initially, there is no spike in node ρ_1, but there are two spikes in node ρ_2. The main work of controlling the computation is done by node ρ_2, while node ρ_1 helps to grow spikes in node ρ_2. There are 5 rules in state s_0 in node ρ_2 and one of them is non-deterministically chosen when computation starts. Rule r_1 is for word λ ($n = m = 0$), rule r_2 is for $b_1 b_3$ ($n = 1, m = 0$), rule r_3 is for $b_2 b_3$ ($n = 0, m = 1$), and rule r_4 is for words $b_2^m b_3^m$ ($n = 0, m \geq 1$). Words of the form $b_1^n b_2^m b_3^{n+m}$ ($n \geq 1, m \geq 0$) can be generated if the computation starts by rule r_5. For the reader's exercise, we omit the detailed explanation here.

The last example we present is $\Pi_3 = (O, Q, \rho_1, \rho_2, \rho_3)$, given in a graphical form in Figure 9; it generates the non-context-free context-sensitive language $\{b_1^n b_2^n b_0^n \mid n \geq 1\}$.

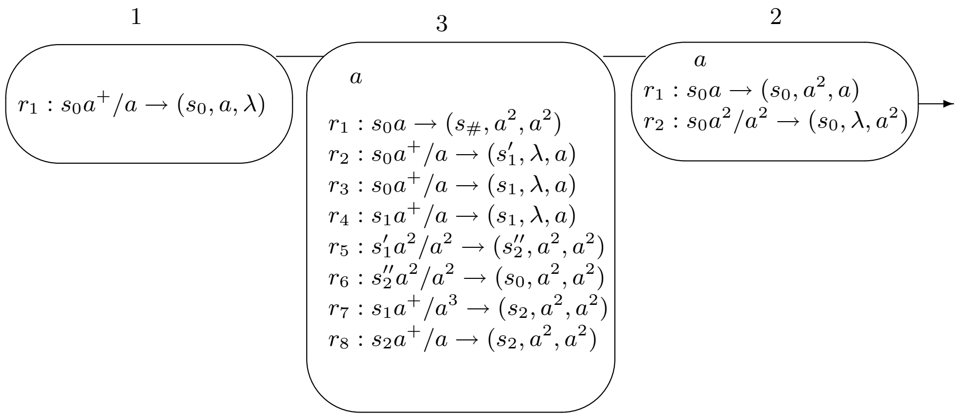

1
$r_1 : s_0 a^+/a \to (s_0, a, \lambda)$
3
a
$r_1 : s_0 a \to (s_\#, a^2, a^2)$
$r_2 : s_0 a^+/a \to (s_1', \lambda, a)$
$r_3 : s_0 a^+/a \to (s_1, \lambda, a)$
$r_4 : s_1 a^+/a \to (s_1, \lambda, a)$
$r_5 : s_1' a^2/a^2 \to (s_2'', a^2, a^2)$
$r_6 : s_2'' a^2/a^2 \to (s_0, a^2, a^2)$
$r_7 : s_1 a^+/a^3 \to (s_2, a^2, a^2)$
$r_8 : s_2 a^+/a \to (s_2, a^2, a^2)$
2
a
$r_1 : s_0 a \to (s_0, a^2, a)$
$r_2 : s_0 a^2/a^2 \to (s_0, \lambda, a^2)$

Fig. 9. An axon P system generating a non-context-free language

The string $b_1^n b_2^n b_0^n, n \geq 1$, is generated in $3n$ steps. Initially, node ρ_1 contains no spike, while each node ρ_2 and ρ_3 contain only one spike. The computation steps are mainly controlled by node ρ_2. There are three rules in node ρ_3 (r_1, r_2, r_3) which can be applied in the first step of the computation. If computation starts applying rule r_1 (r_2, or r_3), a word $b_1 b_2 b_0$ (resp. $b_1^2 b_2^2 b_0^2$ or $b_1^n b_2^n b_0^n$) will be generated. The reader can easily trace the computation steps from Table 2.

8 Final Remarks

The present paper has a preliminary character, and many open problems and research topics about axon P systems remain to be considered. We only mention here some of them.

step		a	a		step		a	a	
1	–	$r_1 : a$	$r_1 : a$	b_1	1	–	$r_3 : a$	$r_1 : a$	b_1
2	$r_1 : a^2$	a^2	$r_2 : a^2$	b_2	2	–	$r_4 : a^2$	$r_1 : a$	b_1
3	$r_1 : a$	–	–	b_0	3	–	$r_4 : a^3$	$r_1 : a$	b_1
					4	–	$r_4 : a^4$	$r_1 : a$	b_1
					5	–	$r_7 : a^5$	$r_1 : a$	b_1
					6	$r_1 : a^2$	$r_8 : a^4$	$r_2 : a^2$	b_2
		a	a		7	$r_1 : a^3$	$r_8 : a^3$	$r_2 : a^2$	b_2
1	–	$r_2 : a$	$r_1 : a$	b_1	8	$r_1 : a^4$	$r_8 : a^2$	$r_2 : a^2$	b_2
2	–	$r_5 : a^2$	$r_1 : a$	b_1	9	$r_1 : a^5$	$r_8 : a$	$r_2 : a^2$	b_2
3	$r_1 : a^2$	$r_6 : a^2$	$r_2 : a^2$	b_2	10	$r_1 : a^6$	–	$r_2 : a^2$	b_2
4	$r_1 : a^3$	–	$r_2 : a^2$	b_2	11	$r_1 : a^5$	–	–	b_0
5	$r_1 : a^2$	–	–	b_0	12	$r_1 : a^4$	–	–	b_0
6	$r_1 : a$	–	–	b_0	13	$r_1 : a^3$	–	–	b_0
					14	$r_1 : a^2$	–	–	b_0
					15	$r_1 : a$	–	–	b_0

Table 2. Some computations in Π_3

Actually, many questions are suggested by the research about SN P systems. For instance, in the systems considered here we have no delay associated with the rules, the spikes are emitted immediately after firing the rule – otherwise stated, the delay is always 0. However, an arbitrary delay can be considered, as usual in SN P systems. Is this of any help? What about considering infinite sequences generated by axon P systems, as investigated in [5] for SN P systems? Can any interesting class of languages or of infinite sequences be characterized/represented in this framework?

Are the hierarchies on the number of nodes infinite? The universality implies the fact that the hierarchies on the number of nodes collapse, but, in view of Theorem 6, our systems are not universal. Another problem related to the non-universality result is to find decidable properties other than the membership one.

What about associating a language in the following way: for each node i we consider a symbol c_i and a configuration $\langle k_1, \ldots, k_m \rangle$ is described by the string $c_1^{k_1} \ldots c_m^{k_m}$. We obtain a language (strictly bounded). Variant: to take only the strings which describe configurations which send out a spike (thus we have a selection of strings). Any relation with L systems?

Acknowledgements

The work of the first author was supported by the National Natural Science Foundation of China under Grants numbers 60573013 and 60421001. The work of the last two authors was supported by Project TIN2005-09345-C04-01 of the Ministry of Education and Science of Spain, cofinanced by FEDER funds.

References

1. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
2. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. Gh. Păun: *Membrane Computing – An Introduction.* Springer-Verlag, Berlin, 2002.
4. 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 [9]).
5. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
6. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
7. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.
8. G.M. Shepherd: *Neurobiology.* Oxford University Press, NY Oxford, 1994.
9. The P Systems Web Page: `http://psystems.disco.unimib.it`.