Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 1, pp. 92-98

A Note on the Generative Power of Axon P Systems

Xingyi Zhang, Jun Wang, Linqiang Pan

Huazhong University of Science and Technology, Department of Control Science and Engineering
Key Laboratory of Image Processing and Intelligent Control
Wuhan 430074, Hubei, People's Republic of China
E-mail: xyzhanghust@gmail.com, junwangjf@gmail.com, lqpan@mail.hust.edu.cn

Abstract: Axon P systems are a class of spiking neural P systems. In this paper, the axon P systems are used as number generators and language generators. As a language generator, the relationships of the families of languages generated by axon P systems with finite and context-free languages are considered. As a number generator, a characterization of the family of finite sets can be obtained by axon P systems with only one node. The relationships of sets of numbers generated by axon P systems with semilinear sets of numbers are also investigated. This paper partially answers some open problems formulated by H. Chen, T.-O. Ishdorj and Gh. Păun.
Keywords: Membrane computing, SN P systems, Axon P systems

1 Introduction

The spiking neural P systems (in short, SN P systems) are a class of bio-inspired computing devices introduced in [6], which attempts to incorporate the idea of spiking neurons into the area of membrane computing. The resulting models are a variant of tissue-like and neural-like P systems, with specific ingredients and way of functioning inspired from spiking neurons. 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 neurobiology. So, recently, a special form of spiking neural P systems, called axon P systems, is introduced in [3], which corresponds to the activity of Ranvier nodes of neuron axon. Actually, axon P systems are a sort of linear SN P systems. Spikes are transmitted along the axon, to the left and to the right, from one node to another node, and an output is provided by the rightmost node. A symbol b_i is associated with a step when i spikes exit the system, in this way a string is associated with a computation. In [3], the language generating power of axon P systems under the above definition was investigated, and many open problems and research topics were formulated. In this paper, we continue the study of axon P systems, specifically, the number generative power and the language generative power are investigated, and in this context we answer some open problems formulated in [3].

SN P systems can be used as number generators (e.g., [2, 4, 6, 7]) or language generators (e.g., [1, 2, 5, 10, 12]). As a variant of SN P systems, axon P systems can also be used as number generators and language generators. As a number generator, we do not care whether or not the computation halts, but we only request that the output node spikes exactly twice during the computation, and the result of a computation is the number of steps elapsed between the two moments when the output node spikes. In this case, a characterization of the family of finite sets is given by axon P systems with one node. Also, the relationships of sets of numbers generated by axon P systems with semilinear sets of numbers are also investigated. As a language generator, each configuration is described by a corresponding string, and the result of a halting computation is defined as the strings associated with configurations where the system emits a spike. In this case, the relationships of the families of languages generated by axon P systems with finite and context-free languages are considered.

The paper is organized as follows. In Section 2, formal language theory prerequisites useful in the following sections are recalled. In Section 3, the definition of axon P systems and the problems considered in this paper are given. Axon P systems as number generators and language generators are investigated respectively in Section 4 and Section 5. Conclusions and remarks are drawn in Section 6.

2 Formal Language Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, as well as basic membrane computing [8] (for more updated information about membrane computing, please refer to [11]), so that we specify here only a few notations and basic definitions.

Let us start by mentioning the following convention: when comparing two generative or accepting devices, number zero is ignored (this corresponds to the usual convention in language theory of ignoring the empty string).

For an alphabet V, V^* denotes the set of all finite strings over V, with the empty string denoted by λ. 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 $|x|$. For a language $L \subseteq V^*$, the set $length(L) = \{|x| \mid x \in L\}$ is called the length set of L. The families of finite, regular, linear and context-free languages are denoted by FIN, REG, LIN, CF, respectively. The families of length sets of languages in FIN, REG, LIN and CF are denoted by $NFIN$, $NREG$, $NLIN$, NCF, respectively. The family of languages generated by oL systems is denoted by oL, and we add the letter E in front of oL if extended oL systems are used. We also denote by $SLIN_1$ the family of semilinear sets of numbers (the subscript indicates that we work with one-dimensional vectors, not with semilinear sets of vectors in general). It is known that the following equalities hold true: $NREG = NLIN = NCF = SLIN_1$ (see, e.g., [8]).

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. 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 also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

A Chomsky grammar is 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 is the finite set of productions. For regular grammars, the productions are of the form $u \to v$, for some $u \in N, v \in T \cup TN$ (in regular grammars, we also allow productions of the form $u \to \lambda$, but only when this is useful for simplifying the grammar: because of the convention that the empty string is not counted when comparing the languages generated by two grammars, such productions are not necessary in regular grammars).

3 Axon P Systems

We now introduce the axon P systems.

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 nodes are arranged along an axon in the order $\rho_1, \ldots, \rho_m$, with ρ_m at the end of the axon; this means that node ρ_m is the output node of the system.

A rule $E/a^c \to (a^l, a^r) \in R_i$ 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 consuming (removing) c spikes from ρ_i (thus only $k - c$ spikes remain in ρ_i), 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. A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

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

If several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

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

Using the rules as described above, one can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Rightarrow C_2$. If $C_1 = \langle k_1, k_2, \ldots, k_m \rangle$, then $C_2 = \langle k'_1, k'_2, \ldots, k'_m \rangle$, where $k'_i = (k_i - c_i) + r_{i-1} + l_{i+1}$, i is the current node, c_i is number of spikes consumed in this node, and r_{i-1}, l_{i+1} are the numbers of spikes sent to this node by the neighboring nodes, $2 \leq i \leq m-1$. In the case of $i = 1$ or $i = m$, there is only one neighbor. 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.

In this paper, we consider the following two ways of defining the result of a computation for axon P systems.

(i) Similar to [6] and [10], we do not care whether or not the computation halts, but we only request that the output node spikes exactly twice during the computation. Then, the number of steps elapsed between the two spikes is the number computed by the axon P system along that computation. We denote by $N_2(\Pi)$ the set of numbers computed in this way by the axon P system Π, and by $Spik_2AP_m(rule_k, cons_p)$ the family of sets $N_2(\Pi)$ generated by axon P systems with at most m nodes, at most k rules in each node, each rule consuming at most p spikes. As usual, any of these parameters is replaced by $*$ if it is not bounded.

(ii) As formulated in [3], a language is associated with a computation of axon P systems in the following way: for each node ρ_i we consider a symbol c_i and a configuration $\langle k_1, \ldots, k_m \rangle$ is described by the string $c_1^{k_1} \ldots c_m^{k_m}$; then, the result of a halting computation is defined as the strings associated with configurations where the system emits a spike. All these strings form the language generated by the system. We denote by $L(\Pi)$ the language generated in this way by the axon P system Π, and by $LAP_m(rule_k, cons_p)$ the family of languages $L(\Pi)$ with m, k, p having the same meaning as above.

4 Axon P Systems as Number Generators

In this section, we investigate the number generative power of axon P systems.

4.1 A Characterization of $NFIN$

In [6], it has been proved that SN P systems can characterize $NFIN$ by using only one neuron. For axon P systems, we have a similar result.

Theorem 1. $NFIN = Spik_2AP_1(rule_*, cons_*)$.

Proof. The inclusion $Spik_2AP_1(rule_*, cons_*) \subseteq NFIN$ can be easily proved: in each step, the number of spikes 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, axon P systems with only one node can only compute finite sets of numbers.

To prove the opposite inclusion $NFIN \subseteq Spik_2AP_1(rule_*, cons_*)$, let us take a finite set of numbers, $F = \{n_1, \ldots, n_k\}$, and we assume that we have $n_1 < n_2 < \ldots < n_k$.

An axon P system that generates F is shown in Figure 1.

Initially, the node contains $n_k + 1$ spikes, hence, only the rule $a^{n_k+1}/a \to (\lambda, a)$ can be used in the first step. It consumes one spike and immediately sends a spike to the environment. In the next step, we have to continue with rules $a^{n_k+1-t}/a \to (\lambda, \lambda)$, for $t = 1$, and then for the respective $t = 2, \ldots, n_1$. But for $t = n_1$ (in step $n_1 + 1$), there is another rule $a^{n_k+1-t} \to (\lambda, a)$ which can be used, non-deterministically chosen. If we choose the second rule, the system will send a spike to the environment again and then the computation halts. Therefore, the number n_1 will be generated. If we choose the first rule, the process will continue, and in a similar way the numbers $n_2, \ldots, n_{k-1}$ can be generated in turn. In step $n_k + 1$ (the last step) only the rule $a^{n_k+1-t} \to (\lambda, a)$ with $t = n_k$ can be used, therefore, the number n_k can be generated and this concludes the computation.

Consequently, all the numbers in F can be generated by the above axon P system, which concludes the proof. □

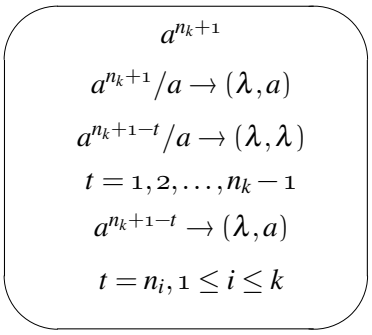

Figure 1: An axon P system generating a finite set of numbers

4.2 Relationships with Semilinear Sets of Numbers

As mentioned in Section 4.1, SN P systems with one neuron generate exactly the family of finite sets of numbers. Actually, SN P systems with two neurons also characterize the family of finite sets. So, the following results on axon P systems are unexpected: such systems with two nodes generate all semilinear sets of numbers.

Theorem 2. $SLIN_1 \subseteq Spik_2AP_2(rule_*, cons_*)$.

Proof. Consider a regular grammar $G = (N, T, S, P)$ with $N = \{A_1, A_2, \ldots, A_n\}, n \geq 1$, $S = A_n$, $T = \{b_1, b_2, \ldots, b_s\}$, and the productions in P are of the forms $A_i \to b_k A_j$, $A_i \to b_k$, $1 \leq i, j \leq n$, $1 \leq k \leq s$.

Then $length(L(G))$ can be generated by an axon P system as shown in Figure 2.

In the initial configuration we have $n+1$ spikes in node ρ_1 and $2n+1$ spikes in node ρ_2, therefore, in the first step, only the rule $a^{2n+1}/a \to (\lambda, a)$ can be used in node ρ_2, and the rule $a^{n+1}/a \to (\lambda, \lambda)$ is used in node ρ_1. Thus, a spike is sent to the environment and n spikes remain in node ρ_1 and $2n$ spikes in node ρ_2. In the next step, node ρ_2 fires by a rule $a^{n+i}/a^{n+i-j} \to (a^n, \lambda)$ or $a^{n+i} \to (\lambda, a)$ associated with a production $A_n \to b_k A_j$ or $A_n \to b_k$ from P, for $i = n$. If the second rule is used, another spike is sent to the environment and the computation halts. In this way, the generated number is 1. If the first rule is used, $2n - j$ spikes are consumed from node ρ_2. 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 $a^{n+i}/a^{n+i-j} \to (a^n, \lambda)$, for $A_i \to b_k A_j$, or $a^{n+i} \to (\lambda, a)$, 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, $n + i - j$ 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 $A_i \to b_k A_j$ or $A_i \to b_k$ 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, all spikes in node ρ_2 are consumed, and n spikes are received from node ρ_1, which will remain in node ρ_2 forever. Moreover, in this step another spike is sent to the environment again. Then the computation halts.

We know that, if we use a production $A_i \to b_k A_j$ or $A_i \to b_k$ one time, then the length of a string generated by the grammar G increases by one, and this corresponds to the number of steps increasing by one by using the associated rule $a^{n+i}/a^{n+i-j} \to (a^n, \lambda)$ or $a^{n+i} \to (\lambda, a)$ one time in the axon P system. Moreover, the second spike is sent to the environment and the computation halts whenever a rule $a^{n+i} \to (\lambda, a)$ is used, which corresponds to a production $A_i \to b_k$ being used in G.

Therefore, the length set of all the strings in $L(G)$ can be generated, and the proof is complete. □

The inclusion in the previous theorem is proper.

Theorem 3. $Spik_2AP_2(rule_5, cons_6) - SLIN_1 \neq \emptyset$.

Proof. Let us consider the axon P system from Figure 3. In the initial configuration, only node ρ_1 contains 5 spikes, hence it fires in the first step by the rule $a(a^4)^+/a^4 \to (\lambda, a^{16})$ and sends 16 spikes to node ρ_2. In the next step, the rule $a \to (a, \lambda)$ or $a \to (\lambda, a^3)$ can be used in node ρ_1, non-deterministically chosen.

If we use the rule $a \to (a, \lambda)$, then we get a number of spikes of the form $4k+1$ in the second node, hence the first rule $a(a^4)^+/a^4 \to (a^4, \lambda)$ in node ρ_2 is applied as many times as possible, thus returning

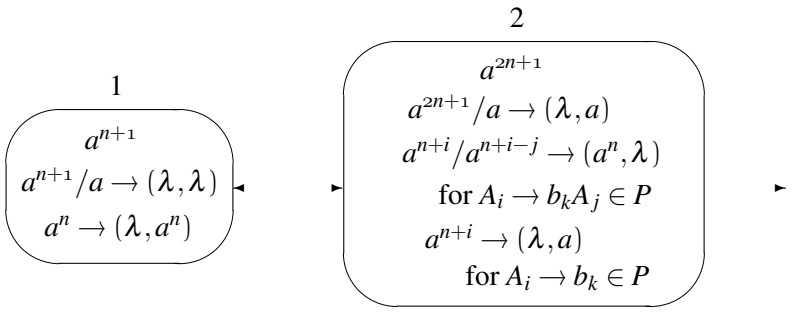

Figure 2: An axon P system generating a semilinear set

the spikes to node ρ_1. To end this returning, we have to use the rule $a \to (a, \lambda)$ in node ρ_2, which makes again the number of spikes from node ρ_1 be of the form $4k+1$ (note that no rule can be applied in any node when the number of spikes is multiple of 4). This process can be iterated any number of times, thus multiplying by 4 the number of spikes present in node ρ_1.

Assume that we have a configuration $\langle 4^n + 1, 0 \rangle$ in step t, for some $n \geq 1$; initially, this is the case, with $n = 1$. In node ρ_1, the rule $a(a^4)^+/a^4 \to (\lambda, a^{16})$ can be used as many times as possible until the node remains with only one spike. It moves all spikes to the second node, multiplied by 4. Therefore, in step $t + 4^{n-1} + 1$, we have a configuration $\langle 1, 4^{n+1} \rangle$. In the next step, node ρ_1 can also use the rule $a \to (\lambda, a^3)$ instead of $a \to (a, \lambda)$. Then the number of spikes from node ρ_2 is of the form $4k+3$ ($4^{n+1}+3$ spikes), hence the rule $a^3(a^4)^+/a \to (\lambda, a)$ should be applied in step $t + 4^{n-1} + 3$ and a spike is sent to the environment. At the same time, the number of spikes decreases by one and will be of the form $4k+2$ ($4^{n+1}+2$ spikes). Therefore, the rule $a^2(a^4)^+/a^4 \to (\lambda, \lambda)$ should be applied in step $t + 4^{n-1} + 4$. This rule does not change the form of the number of spikes and it remains in the form of $4k+2$, hence it is used as many times as possible. In this way, no spike is sent to the environment until the number of spikes from ρ_2 becomes 6 and this process needs $4^n - 1$ steps. Then, in the step $t + 4^{n-1} + 4^n + 3$, the rule $a^6 \to (\lambda, a)$ should be used and the second spike is sent to the environment. Therefore, the number of steps elapsed between the two spikes which were sent to the environment is 4^n and the computation halts.

Consequently, the set $\{4^n \mid n \geq 1\}$ can be generated by this system, which, obviously, is not a semilinear set. □

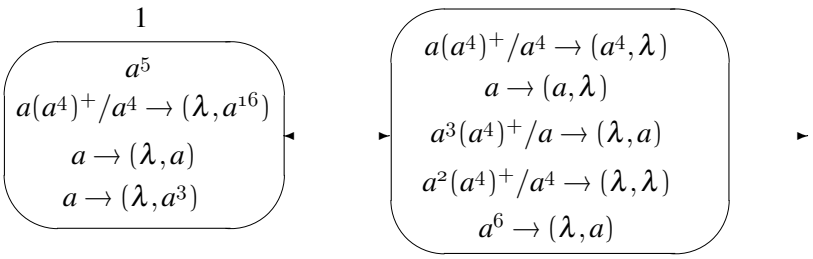

Figure 3: An axon P system generating a non-semilinear set $\{4^n | n \geq 1\}$

In [6], it has been shown that semilinear sets of numbers can be characterized by SN P systems with a bound on the number of spikes present in any neuron, but the number of neurons is not bounded. The following theorem also gives a characterization of semilinear sets of numbers, but here only two nodes are used in the axon P systems.

Theorem 4. *$SLIN_1 = Spik_2AP_2(rule_*, cons_*, bound_*)$, where $bound_*$ indicates that axon P systems have a bound on the number of spikes present in any node, but this bound is not specified.*

Proof. From Theorem 2, it is enough to prove the inclusion $Spik_2AP_2(rule_*, cons_*, bound_*) \subseteq SLIN_1$.
The proof is similar to the one of Lemma 9.1 in [6], so we do not recall it here. □

5 Axon P Systems as Language Generators

We now pass to considering the language generative power of axon P systems.

All strings generated by an axon P system in the way mentioned in Section 3 are of the form $c_1^{k_1}c_2^{k_2}\dots c_m^{k_m}$, where $\langle k_1,k_2,\dots,k_m\rangle$ is a configuration of the system. Therefore, if there is a bound on the number of spikes in any node of axon P systems, we can directly have the following result.

Remark 5. $LAP_n(rule_*,cons_*,bound_*) \subseteq FIN$, for $n \geq 1$.

Moreover, as a direct consequence of this restrictive way of defining the languages associated with axon P systems, we cannot find a characterization of finite languages for the general axon P systems. For instance, it is not difficult to find that the string $c_1c_2c_1$ cannot be generated by any axon P system.

5.1 Beyond *CF*

Theorem 6. *There exists at least a language $L \in E0L-CF$, which can be generated by axon P systems.*

Proof. Let us consider the language $L=\{c_1^{2n}c_2^{2n}c_3^{2n} \mid n \geq 1\}$; it is obvious that $L \in E0L-CF$.

We construct the axon P system whose initial configuration is shown in Figure 4.

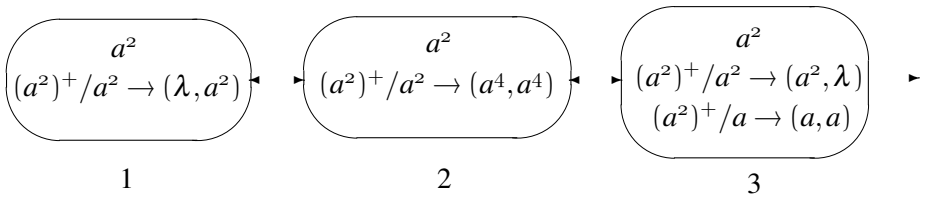

Figure 4: An axon P system generating the language $\{c_1^{2n}c_2^{2n}c_3^{2n} \mid n \geq 1\}$

Initially, each node of the system contains two spikes. Hence, each node can be activated, and in node ρ_3 both the rule $(a^2)^+/a^2 \to (a^2,\lambda)$ and the rule $(a^2)^+/a \to (a,a)$ can be applied, non-deterministically chosen. If the first rule is applied, then the two spikes in each node are consumed and no spike is sent to the environment. At the same time, each node accumulates 4 spikes (the 4 spikes in nodes ρ_1 and ρ_3 are received from node ρ_2; the 4 spikes in node ρ_2 are received from both node ρ_1 and node ρ_3, two from node ρ_1 and two from node ρ_3). In this way, all the nodes can be activated in the next step. If in node ρ_3 we continue using the rule $(a^2)^+/a^2 \to (a^2,\lambda)$, then in a similar way each node obtains 6 spikes. This process can be iterated until the second rule is applied. In this case, the number of spikes in each node increases by two in each step. Therefore, each node of the system accumulates $2n$ spikes in step n, for $n \geq 1$.

At any moment, we can also apply the second rule in node ρ_3. Assume that it is applied in step $n+1$, then node ρ_3 sends a spike to the environment, which means that the string $c_1^{2n}c_2^{2n}c_3^{2n}$ is generated by this system. At the same time, node ρ_1 accumulates $2n+2$ spikes, node ρ_2 accumulates $2n+1$ spikes, and node ρ_3 accumulates $2n+3$ spikes. In the next step, only node ρ_1 can be activated. Thus, it consumes two spikes and sends two spikes to node ρ_2. This process can be iterated until all the spikes in node ρ_1 are moved to node ρ_2, and then the computation halts.

Therefore, the language $\{c_1^{2n}c_2^{2n}c_3^{2n} \mid n \geq 1\}$ can be generated by the axon P system and this concludes the proof. □

6 Conclusions and Remarks

In this paper, the number generative power and the language generative power of axon P systems are investigated. As a variant of SN P systems, there remain many open problems and research topics about axon P systems to be considered. One important aspect is suggested by the research about SN P systems. For instance, as usual in SN P systems, an arbitrary delay can be considered in axon P systems. What about considering the spike trains (finite or infinite) themselves as the result of a computation in axon P systems, as investigated in [10]. The various applications for axon P systems also deserve to be considered.

As suggested by Gheorghe Păun in [9], a more general and probably more interesting problem is combining neurons and axons in a global model; maybe also astrocytes can be added, thus obtaining a

more complex model, closer to reality. Please refer to [3, 9] for more open problems and research topics related to axon P systems.

Acknowledgements

Comments and suggestions from Gheorghe Păun are greatly acknowledged. The project was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, 60503002, and 60533010), 863 Program of China (2006AA01Z104), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), and HUST-SRF (2007Z015A).

Bibliography

[1] H.M. Chen, M. Ionescu, M. J. Pérez-Jiménez, R. Freund, and Gh. Păun, On String Languages Generated by Spiking Neural P Systems, *Fundamenta Informaticae*, Vol. 75(1–4), pp. 141–162, 2007.

[2] H. M. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun and M. J. Pérez-Jiménez, Spiking Neural P Systems with Extended Rules, in: 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, pp. 241–266, 2006.

[3] H. M. Chen, T.-O. Ishdorj and Gh. Păun, Computing Along the Axon, *Progress in Natual Science*, vol. 17(4), pp. 417–423, 2007.

[4] O. H. Ibarra, S. Woodworth, F. Yu and A. Păun, On Spiking Neural P Systems and Partially Blind Counter Machines, *Natural Computing*, Vol. 7(1), pp. 3–19, 2008.

[5] O. H. Ibarra and S. Woodworth, Characterizing Regular Languages by Spiking Neural P Systems, *International Journal of Foundations of Computer Science*, Vol. 18(6), pp. 1247–1256, 2007.

[6] M. Ionescu, Gh. Păun and T. Yokomori, Spiking Neural P Systems, *Fundamenta Informaticae*, Vol. 71(2–3), pp. 279–308, 2006.

[7] M. Ionescu, Gh. Păun and T. Yokomori, Spiking Neural P Systems with Exhaustive Use of Rules, *International Journal of Unconventional Computing*, Vol. 3(2), pp. 135–154, 2007.

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

[9] Gh. Păun, Twenty Six Research Topics about Spiking Neural P Systems, 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, pp. 263–280, 2007.

[10] Gh. Păun, M. J. Pérez-Jiménez and G. Rozenberg, Spike Trains in Spiking Neural P Systems, *International Journal of Foundations of Computer Science*, Vol. 17(4), 975–1002, 2006.

[11] The P System Web Page: http://ppage.psystems.eu

[12] X. Y. Zhang, X. X. Zeng and L. Q. Pan, On String Languages Generated by Spiking Neural P Systems with Exhaustive Use of Rules, *Natural Computing*, to appear.

Xingyi Zhang was born in China on June 6, 1982. He received his master degree in applied mathematics from Huazhong University of Science and Technology in 2006. Currently, his main research fields are formal language theory and its applications, unconventional models of computation, especially, membrane computing. He has published several scientific papers in international journals.

Linqiang Pan was born in Zhejiang, China on November 22, 1972. He got PhD at Nanjing University in 2000. Since 2004, he is a professor at Huazhong University of Science and Technology, China. His main research fields are graph theory and membrane computing.