Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 4 (December), pp. 647-655

Spiking Neural P Systems with Several Types of Spikes

M. Ionescu, G. Păun, M. J. Pérez-Jiménez, A. Rodríguez-Patón

Mihai Ionescu
University of Piteşti
Str. Târgu din Vale, nr. 1, 110040 Piteşti, Romania
E-mail: armandmihai.ionescu@gmail.com

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

Mario J. Pérez-Jiménez
Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: marper@us.es

Alfonso Rodríguez-Patón
Department of Artificial Intelligence, Faculty of Computer Science
Polytechnical University of Madrid, Campus de Montegancedo
Boadilla del Monte 28660, Madrid, Spain
E-mail: arpaton@fi.upm.es

Abstract: With a motivation related to gene expression, where enzymes act in series, somewhat similar to the train spikes traveling along the axons of neurons, we consider an extension of spiking neural P systems, where several types of “spikes" are allowed. The power of the obtained spiking neural P systems is investigated. Some further extensions are mentioned, such as considering a process of decay in time of the spikes.

Keywords: Natural computing, Membrane computing, P system, Turing computability

1 Introduction

The present note lies at the intersection of two active research branches of bio-informatics/natural computing, namely, gene expression and membrane computing. Specifically, an extension of so-called spiking neural P systems (in short, SN P systems) is considered, with motivations related to gene expression processes.

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. Such a rule has the general form $E/a^c \to a^p; d$, where E is a regular expression (equivalently, we can consider it a regular language) and a denotes the spike; if the contents of the neuron is described by an element of the regular language (identified by) E, then the rule is enabled, c spikes are consumed and p are produced, and sent, after a time delay of d steps, along the synapses leaving the neuron. There also are forgetting rules of the form $a^c \to \lambda$, with the meaning that, if the neuron contains exactly c spikes, then they can be removed (forgotten). One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train.* Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron.

These computing devices were introduced in [7] and then investigated in a large number of papers; we refer to the corresponding chapter from [13] and to the membrane computing website [16] for details.

In turn, gene expression is an important research area where various transcription factors appears and, important for their activity, their frequency matters – see, for instance, [1], [10], [12]. This means that a spiking like process is encountered, but with several "spikes", the regulator proteins which bind to a promotor depending on their concentration. In some sense we have here a communication process in which a signal encoded in a concentration (the transcription factor) is transduced to a frequency signal (the bursts of mRNA associated to the bindings of the transcription factor with the promotor) and again transduced back to a concentration (the level of protein produced). Thus, conceptually, we can approach this process in terms of theoretical machineries developed for spiking neurons – with the necessity of considering a variety of spikes, not only one as in the neural case. This is also suggested in [1]: "...we anticipate that frequency-modulated regulation may represent a general principle by which cells coordinate their response to signals."

Starting from these observations, we relate here the two research areas, introducing SN P systems with several types of spikes. Such a possibility was somehow forecasted already from the way the definition in [7] is given, with an alphabet, O, for the set of spikes, but with only one symbol in O; up to now, only a second type of spikes was considered, in [11], namely *anti-spikes*, which, when introduced, are immediately annihilated, in pairs with usual spikes. This extension to several types of spikes is natural also in view of the fact that all classes of P systems investigated in membrane computing work with arbitrary alphabets of objects.

As expected, having several types of spikes helps in proofs; in particular, we obtain the universality of the SN P systems with several types of spikes for systems with a very reduced number of neurons – remember that for systems with only one type of spikes the proofs do not bound the number of neurons (but such a bound can be found due to the existence of universal SN P systems, hence with a fixed number of neurons, but used in the computing mode, having both an input and an output). Three ways to define the result of a computation are considered: as the number of objects inside a specified neuron, as the number of objects sent out by the output neuron, and as the distance in time between the first two spikes sent out during the computation.

What is not investigated is the case of generating strings, in the sense of [3], [4], or even in the distributed case of [8]. Other open problems are mentioned in the rest of the paper and in the final section of it.

2 Formal Language Theory Prerequisites

We assume the reader to be familiar with basic language and automata theory, e.g., from [14] and [15], 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 $\{a\}^*, \{a\}^+$. For a language L, we denote by $sub(L)$ the set of all substrings of strings in L.

As usual in membrane computing, the multisets over a finite universe set U are represented by strings in U^* (two strings equal modulo a permutation represent the same multiset). If $u, v \in U^*$, we write the fact that u is a submultiset of v in the form $u \subseteq v$, with the understanding that there is a permutation of v having u as a substring (this can be formally formulated also in terms of Parikh mapping, but we do not enter into details). Similarly, we write $u \in sub(L)$ for a multiset u and a set L of multisets, meaning that u is a submultiset of a multiset in L.

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

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k non-deterministically chosen),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \mathtt{HALT}$ (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we continue to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n present in register 1 at that time is said to be generated by M. Without loss of generality we may assume that in the halting configuration all other registers are empty. It is known that register machines generate all sets of numbers which are Turing computable – we denote this family with NRE (RE stands for "recursively enumerable"). By $NFIN$ we denote the family of finite sets of natural numbers.

In the following sections, when comparing the power of two computing devices, number 0 is ignored (this corresponds to the fact that when comparing the power of language generating or accepting devices, the empty string λ is ignored).

3 Spiking Neural P Systems with Several Types of Spikes

We directly introduce the type of SN P systems we investigate in this paper; although somewhat far from the idea of a spike from the neural area, we still call the objects processed in our devices *spikes*.

A *spiking neural P system with several types of spikes* (abbreviated as *SN*$^+$ *P system*, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0), \text{ where:}$$

1. O is the alphabet of *spikes* (we also say *objects*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (w_i, R_i), 1 \leq i \leq m,$ where:
   a) $w_i \in O^*$ is the *initial multiset* of spikes contained in σ_i;
   b) R_i is a finite set of *rules* of the forms

(i) $E/u \to a$, where E is a regular language over O, $u \in O^+$, and $a \in O$ (spiking rules);

(ii) $v \to \lambda$, with $v \in O^+$ (forgetting rules) such that there is no rule $E/u \to a$ of type (i) with $v \in E$;

3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);

4. $i_0 \in \{1, 2, \dots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

A rule $E/u \to a$ is applied as follows. If the neuron σ_i contains a multiset w of spikes such that $w \in L(E)$ and $u \in sub(w)$, then the rule can *fire*, and its application means consuming (removing) the spikes identified by u and producing the spike a, which will exit immediately the neuron. In turn, a rule $v \to \lambda$ is used if the neuron contains exactly the spikes identified by v, which are removed ("forgotten"). A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

If a rule $E/u \to a$ has $E = \{u\}$, then we will write it in the simplified form $u \to a$.

The spike emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$.

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

Using the rules as described above, one can define transitions among configurations. 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.

There are many possibilities to associate a result with a computation, in the form of a number. Three possibilities are considered here: the number of objects in the output neuron in the halting configuration, the number of spikes sent to the environment by the output neuron, and the number of steps elapsed between the first two steps when the output neuron spikes. In the first two cases only halting computations provide an output, in the last case we can define the output also for ever going computations – but in what follows we only work with halting computations also for this case.

We denote by $N_\alpha(\Pi)$ the set of numbers generated as above by an SN$^+$ P system Π with the result defined in the mode $\alpha \in \{i, o, d\}$, where i indicate the internal output, o the external one (as the number of spikes), and d the fact that we count the distance between the first two spikes which exit the system. Then, $N_\alpha SN^+P_m$ is the family of sets of numbers $N_\alpha(\Pi)$, for SN$^+$ P systems with at most $m \geq 1$ neurons. As usual, the subscript m is replaced by $*$ if the number of neurons does not matter.

Before passing to investigate the power of the previously defined systems, let us mention that we have introduced here SN P systems of the *standard* type in what concerns the rules, i.e., producing only one spike, and without *delay*; *extended* rules are natural ($E/u \to v$, with both u and v multisets), but this is a too general case from a computability point of view, corresponding to cooperating P systems. It is important also to note that the rules we use have both additional powerful features – context sensitivity induced by the existence of the control regular language E, and strong restrictions – the produced spike (only one) should leave immediately the neuron, it cannot be further used in the same place without being sent back by the neighboring neurons. These features are essentially present in the proofs from the next section.

4 The Power of SN$^+$ P Systems

We start by considering the case when the result is counted inside the system (like in general P systems, hence somewhat far from the style of SN P systems).

Lemma 1. $NRE \subseteq N_i SN^+ P_3$.

Proof: Let us consider a register machine $M = (n, H, l_0, l_h, I)$. We construct the following SN$^+$ P system

$$
\begin{aligned}
\Pi &= (O, \sigma_1, \sigma_2, \sigma_3, syn, 1), \text{ with:}\\
O &= \{a_i \mid 1 \leq i \leq n\} \cup \{l, l', l'' \mid l \in H\},\\
\sigma_1 &= (l_0, R_1),\\
R_1 &= \{O^*/l_i \to l'_i \mid (l_i : \mathtt{ADD}(r), l_j, l_k) \in I\}\\
&\cup \{O^* a^r O^*/l_i a_r \to l''_j,\ (O^* - O^* a^r O^*)/l_i \to l''_k \mid (l_i : \mathtt{SUB}(r), l_j, l_k) \in I\}\\
&\cup \{l_h \to \lambda\},\\
\sigma_2 &= (\lambda, R_2),\\
R_2 &= \{l'_i \to l_j,\ l'_i \to l_k \mid (l_i : \mathtt{ADD}(r), l_j, l_k) \in I\} \cup \{l'' \to l \mid l \in H\},\\
\sigma_3 &= (\lambda, R_3),\\
R_3 &= \{l'_i \to a_r \mid (l_i : \mathtt{ADD}(r), l_j, l_k) \in I\} \cup \{l'' \to \lambda \mid l \in H\},\\
syn &= \{(1,2),\ (1,3),\ (2,1),\ (3,1)\}.
\end{aligned}
$$

The functioning of this system can be easily followed. The contents of register r is represented by the number of copies of object a_r present in the system. There are also objects associated with the labels of M.

Initially, we have only the object l_0 in neuron σ_1. In general, in the presence of a label l_i of an instruction in I, the instruction is simulated by the system Π. For the ADD instructions, the change of labels is done with the help of neuron σ_2 and the addition of a further object a_r is done in neuron σ_3. For the SUB instructions, the check for zero is performed by means of the regular language associated with the rules in R_1. The computation continues as long as the work of the machine M continues. When the label l_h is introduced – by the neuron σ_2 – the computation stops after one further step, when this object is removed from the output neuron, σ_1. Thus, in the end, this neuron only contains copies of object a_1, hence their number represents the value present in the first register of M in the end of the computation. Thus, $N(M) = L_i(\Pi)$. $\square$

Theorem 2. $NFIN = N_i SN^+ P_1 = N_i SN^+ P_2 \subset N_i SN^+ P_3 = NRE$.

Proof: The inclusions $N_i SN^+ P_1 \subseteq N_i SN^+ P_2 \subseteq N_i SN^+ P_3$ are obvious from the definitions. The inclusion $N_i SN^+ P_3 \subseteq NRE$ is straightforward (we can also invoke for it the Turing-Church thesis).

In an SN$^+$ P system with two components, the number of spikes present inside the two neurons cannot be increased (each spiking rule consumes at least one spike and produces only one spike, while there is no duplication of spikes because of multiple synapses which exit a neuron), hence we have $N_i SN^+ P_2 \subseteq NFIN$.

On the other hand, $NFIN \subseteq N_i SN^+ P_1$. Indeed, consider a finite set of numbers, $F = \{n_1, n_2, \dots, n_k\}$; assume that $1 \leq n_1 < n_2 < \cdots < n_k$ (remember that we ignore the number 0). We construct the system

$$\Pi = (\{a\}, (a^{n_k+1}, \{a^{n_k+1}/a^{n_k+1-n_i} \to a \mid 1 \leq i \leq k\}), \emptyset, 1).$$

We have $L_i(\Pi) = F$: each computation has only one step, which non-deterministically uses one of the rules in R_1. Each such rule just consumes a number of spikes, passing from the initial $n_k + 1$ spikes to any number $n_i \in F$, which cannot be further processed.

Together with Lemma 1, this concludes the proof of the theorem. $\square$

Let us note in the construction from the proof of Lemma 1 that all neurons spike a large number of times (related to the length of the computation), not directly related to the number computed in the first register of M. This makes difficult to imagine a system with only three neurons which is universal when the result is defined as the number of spikes sent out. However, one additional neuron suffices in such a case.

Theorem 3. $NFIN = N_oSN^+P_1 \subset N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4 = NRE$.

Proof: Again, the inclusions $N_oSN^+P_1 \subseteq N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4 \subseteq NRE$ are obvious from the definitions.

The inclusion $NRE \subseteq N_oSN^+P_4$ can be obtained by a slight extension of the construction in the proof of Lemma 1: we replace the rule $l_h \to \lambda$ from R_1 with the rule

$$a_1^+ l_h / l_h a_1 \to l_h.$$

We also add a neuron σ_4, considered as output neuron, linked by synapses (1, 4), (4, 1) to the neuron σ_1 and containing the unique rule

$$l_h \to l_h.$$

When the computation of M stops, hence l_h is introduced in σ_1, this object removes one by one the objects a_1 and moves to the output neuron. This neuron both sends l_h out and back to σ_1, hence the number of copies of l_h sent out is equal with the number stored in the first register of M.

This time, an SN$^+$ P system with two components can compute an arbitrarily large number, by sending out an arbitrarily large number of spikes. For instance,

$$\Pi = (\{a\}, (a, \{a \to a\}), (aa, \{aa/a \to a,\ aa \to a\}), \{(1,2),(2,1)\}, 2),$$

has $L_o(\Pi) = \{1, 2, \dots\}$ (neuron σ_2 spikes step by step, until using the rule $aa \to a$, when only one spike remains in the system and the computation halts).

If we have only one neuron, the computation can last as may steps as many spikes are initially inside, hence $N_oSN^+P_1 \subseteq NFIN$.

On the other hand, $NFIN \subseteq N_iSN^+P_1$. Indeed, consider again a finite set of numbers, $F = \{n_1, n_2, \dots, n_k\}$ such that $1 \le n_1 < n_2 < \dots < n_k$ and construct the system

$$\begin{aligned} \Pi &= (\{a\}, \sigma_1, \emptyset, 1), \text{ with} \\ \sigma_1 &= (a^{n_k+1}, R_1), \\ R_1 &= \{a^{n_k+1}/a^{n_k+1-n_i+1} \to a \mid 1 \le i \le k\} \\ &\cup \ \{a^r/a \to a \mid 1 \le r \le n_k - 1\}. \end{aligned}$$

We have $L_i(\Pi) = F$: each computation starts with a step which uses non-deterministically a rule $a^{n_k+1}/a^{n_k+1-n_i+1} \to a$, which decreases the number of spikes from the initial n_k+1 to some $n_i - 1$; at this time, one spike was sent out. From now on, we use deterministically rules of the form $a^r/a \to a$, for all $r = 1, 2, \dots, n_i - 1$, hence for $n_i - 1$ steps, always sending out one spike. Thus, in total, we send out n_i spikes, for each $n_i \in F$.

Combining all these remarks, we have the theorem. □

It is an *open problem* whether or not the inclusions $N_oSN^+P_2 \subseteq N_oSN^+P_3 \subseteq N_oSN^+P_4$ are proper.

Theorem 4. $NFIN = N_dSN^+P_1 = N_dSN^+P_2 \subset N_dSN^+P_3 \subseteq N_dSN^+P_4 = NRE$.

Proof: As above, the inclusions $N_dSN^+P_1 \subseteq N_dSN^+P_2 \subseteq N_dSN^+P_3 \subseteq N_dSN^+P_4 \subseteq NRE$ are obvious from the definitions.

The inclusion $NFIN \subseteq N_dSN^+P_1$ is already proved for SN P systems with only one type of spikes. Like in that case, we also obtain the inclusion $N_dSN^+P_2 \subseteq NFIN$: in order to generate an arbitrarily large number, the output neuron should not spike for an arbitrarily large number of steps, but this is not possible in a system with only two neurons, because if only one neuron is working, it can perform only a number of steps bounded by the number of spikes initially present in it.

The fact that $N_dSN^+P_3$ contains infinite sets of numbers is also known for standard SN P systems.

What remains to prove is the inclusion $NRE \subseteq N_dSN^+P_4$ and this can again be obtained by an extension of the construction in the proof of Lemma 1; because this extension is not immediate, we give the construction in full details.

We consider a register machine $M = (n, H, l_0, l_h, I)$ and construct the SN$^+$ P system Π as indicated in Figure 1 – this time we do not give the system formally, but we represent it graphically, in the way usual in the SN P systems area.

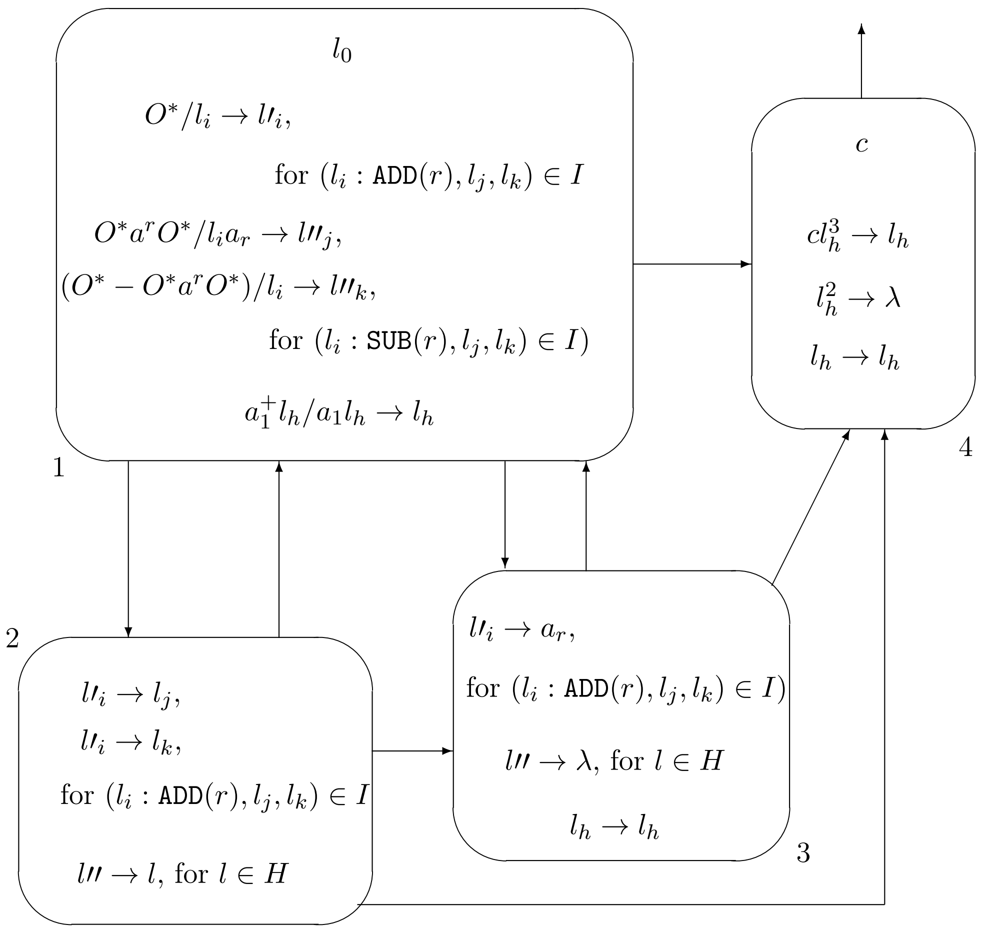
l_0
$O^*/l_i \to l\prime_i$,
for $(l_i : \texttt{ADD}(r), l_j, l_k) \in I$
$O^*a^rO^*/l_ia_r \to l\prime\prime_j$,
$(O^* - O^*a^rO^*)/l_i \to l\prime\prime_k$,
for $(l_i : \texttt{SUB}(r), l_j, l_k) \in I)$
$a_1^+l_h/a_1l_h \to l_h$
1
c
$cl_h^3 \to l_h$
$l_h^2 \to \lambda$
$l_h \to l_h$
4
2
$l\prime_i \to l_j$,
$l\prime_i \to l_k$,
for $(l_i : \texttt{ADD}(r), l_j, l_k) \in I$
$l\prime\prime \to l$, for $l \in H$
$l\prime_i \to a_r$,
for $(l_i : \texttt{ADD}(r), l_j, l_k) \in I)$
$l\prime\prime \to \lambda$, for $l \in H$
$l_h \to l_h$
3

Figure 1: The SN$^+$ P system from the proof of Theorem 4

The work of this system is identical to that in the proof of Lemma 1, until producing the object l_h (the objects which arrive in the output neuron σ_4 from all other neurons remain here unused).

When σ_2 introduces the object l_h, it is sent to all other neurons. It waits unused in σ_4, but in

σ_1 and σ_3 it is reproduced in each step, hence these two neurons feed repeatedly each other with one copy of l_h. In σ_1, each use of the rule $a_1^+ l_h/a_1 l_h \to l_h$ removes one copy of a_1. In the end of step 1 (we count here only the steps after having l_h in the system, hence for the phase when the output is produced), neuron σ_4 contains three copies of l_h. Thus, in step 2, this neuron spikes.

From now on, neurons σ_1, σ_3 spike repeatedly, exchanging copies of l_h, σ_4 always forgets the two copies of l_h received from σ_1, σ_3 (while σ_2 just accumulates copies of l_h, which cannot be processed here). When the last copy of a_1 is removed from σ_1 (if m copies of a_1 were present here when l_h was introduced, then this happens in step m, after having l_h in the system), this is the last step when σ_4 receives two spikes. In the next step $(m+1)$ it receives only the spike produced by σ_3, which is used (in step $m+2$) by the rule $l_h \to l_h$ in σ_4. The computation stops. The number of steps between the two spikes sent out by the output neuron is $(m+2)-2=m$, hence the number computed by the register machine in its first register.

The proof of the theorem is now complete. □

It is an *open problem* whether or not the inclusion $N_gSN^+P_3 \subseteq N_gSN^+P_4$ is proper.

5 Final Remarks

In gene expression it is also the case that the enzymes have a time dependency of their reactivity, which can be captured in terms of SN P systems by considering decaying spikes, in the sense of [6]. For instance, we can associate an *age* with each produced spike, by using rules of the form $E/u \to (a,t)$, where $t \geq 1$ is the "duration of life" of this spike. If the spike is not used in a step, then its life is decreased by one unit (this is like having rewriting rules $(a,s) \to (a,s-1)$, used in parallel for all spikes not used in spiking or forgetting rules), until reaching the state $(a,0)$, when a rule $(a,0) \to \lambda$ is assumed to be applied. This feature remains to be further investigated.

Let us close by recalling the fact that besides the synchronized (sequential in each neuron) mode of evolution, there were also introduced other modes, such as the exhaustive one, [9], and the non-synchronized one, [2], which also deserve to be considered for SN P systems with several types of spikes.

Acknowledgements

The work of M. Ionescu was possible due to CNCSIS grant RP-4 12/01.07.2009. The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200. The work of A. Rodriguez-Paton has been supported by Spanish Ministry of Science and Innovation under project TIN2009 - 14421. The careful reading of the paper by two anonymous referees is gratefully acknowledged.

Bibliography

[1] L. Cai, C.K. Dalal, M.B. Elowitz: Frequency-modulated nuclear localization bursts coordinate gene regulation. *Nature*, 455 (25 September 2008).

[2] M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 24-25 (2009), 2352–2364.

[3] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.

[4] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.

[5] H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.

[6] R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking. *Intern. J. Found. Computer Sci.*, 19 (2008), 1223–1234.

[7] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.

[8] M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: T. Yokomori: Spiking neural dP systems. *Proc. Ninth Brainstorming Week on Membrane Computing*, Sevilla, 2011, RGNC Report 01/2011.

[9] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.

[10] E.M. Ozbudak, M. Thattai, I. Kurtser, A.D. Grossman, A. van Oudenaarden: Regulation of noise in the expression of a single gene. *Nature Genetics*, 31 (May 2002).

[11] L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *Intern. J. Computers, Comm. Control*, 4, 3 (2009), 273–282.

[12] J. Paulsson: Models of stochastic gene expression. *Physics of Life Reviews*, 2 (2005), 157–175.

[13] Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.

[14] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, 1998.

[15] A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.

[16] The P Systems Website: `http://ppage.psystems.eu`.