# When Matrices Meet Brains

Xiangxiang Zeng[1,3], Henry Adorna[2],
Miguel Ángel Martínez-del-Amor[3] Linqiang Pan[1]

[1] Key Laboratory on Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074 Hubei, Peoples Republic of China
[2] Department of Computer Science (Algorithms and Complexity)
University of the Philippines
Diliman 1101 Quezon City, Philippines
[3] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

**Summary.** Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. In this work, a discrete structure representation of SN P systems is proposed. Specifically, matrices are used to represent SN P systems. In order to represent the computations of SN P systems by matrices, configuration vectors are defined to monitor the number of spikes in each neuron at any given configuration; transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. Nondeterminism of the systems is assured by a set of spiking transition vectors that could be used at any given time during the computation. With such matrix representation, it is quite convenient to determine the next configuration from a given configuration, since it involves only multiplying vectors to a matrix and adding vectors.

## 1 Introduction

*Membrane computing* was initiated by Păun [6] and has developed very rapidly (already in 2003, ISI considered membrane computing as "fast emerging research area in computer science", see `http://esi-topics.com`). It aims to abstract computing ideas (data structures, operations with data, computing models, etc.) from the structure and the functioning of single cell and from complexes of cells, such as tissues and organs including the brain. The obtained models are distributed and parallel computing devices, called *P systems*. For updated information about membrane computing, please refer to [8].

This work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short) [3]. SN P systems were inspired by the neurophysiological behavior of neurons (in brain) sending electrical impulses along axons to other neurons, with the aim of incorporating specific ideas from spiking neurons into membrane computing. Generally speaking, in an SN P system the processing elements are called *neurons* and are placed in the nodes of a directed graph, called the *synapse graph*. The content of each neuron consists of a number of copies of a single object type, namely the *spike*. Each neuron may also contain rules which allow to remove a given number of spikes from it, or send spikes (possibly with a delay) to other neurons. The application of every rule is determined by checking the content of the neuron against a regular set associated with the rule.

Representation of P systems by discrete structures has been one topic in the field of membrane computing. One of the promising discrete structures to represent P systems is matrix. Models with matrices as their representation have been helpful to physical scientists – biologist, chemists, physicists, engineers, statisticians, and economists – solving real world problems. Recently, matrix representation was introduced to represent a restricted form of cell-like P systems without dissolution (where only non-cooperate rules are used) [2]. It was proved that with algebraic representation P systems can be easily simulated and computed backward (that is, to find all the configurations that produce a given one in one computational step).

In this work, a matrix representation of SN P systems without delay is proposed, where configuration vectors are defined to represent the number of spikes in neurons; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of system; transition net gain vectors are also introduced to quantify the total amount of spikes consumed and produced after the chosen rules are applied. With these algebraic representation, matrix transition can be used to compute the next configuration from a given configuration.

The rest of this paper is organized as follows. In the next section, we introduce the definition of SN P systems. Section 3 presents the matrix representation of SN P systems. Section 4 illustrates how to represent the computation of SN P system by matrix transition. Conclusions and remarks are given in Section 5.

## 2 Spiking Neural P Systems

In this section, a restricted variant of SN P systems, SN P systems without delay, is introduced.

**Definition 1.** *An SN P system without delay, of degree $m \geq 1$, is a construct of the form*

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

*where:*

*1. $O = \{a\}$ is the singleton alphabet (a is called spike);*
*2. $\sigma_1, \dots, \sigma_m$ are neurons, of the form*

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

*where:*
*a) $n_i \geq 0$ is the initial number of spikes contained in $\sigma_i$;*
*b) $R_i$ is a finite set of rules of the following two forms:*
*(1) $E/a^c \to a^p$, where $E$ is a regular expression over $a$, and $c \geq 1$, $p \geq 1$, with the restriction $c \geq p$;*
*(2) $a^s \to \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \to a^p$ of type (1) from $R_i$, $a^s \notin L(E)$;*
*3. $syn = \{(i, j) \mid 1 \leq i, j \leq m, \, i \neq j\}$ (synapses between neurons);*
*4. $in, out \in \{1, 2, \dots, m\}$ indicate the input and output neurons, respectively.*

The rules of type (1) are spiking (or called firing) rules, which are applied as follows. If the neuron $\sigma_i$ contains $k$ spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a^p \in R_i$ can be applied. This means that consuming (removing) $c$ spikes (thus only $k - c$ spikes remain in $\sigma_i$), the neuron is fired, and it produces $p$ spikes; these spikes are transported to all neighbor neurons by outgoing synapses.

The rules of type (2) are forgetting rules; they are applied as follows: if the neuron $\sigma_i$ contains exactly $s$ spikes, then the rule $a^s \to \lambda$ from $R_i$ can be used, meaning that all $s$ spikes are removed from $\sigma_i$.

If a rule $E/a^c \to a$ has $E = a^c$, then it is written in the simplified form $a^c \to a$.

In each time unit, if a neuron $\sigma_i$ can apply one of its rules, then a rule from $R_i$ must be applied. Since two spiking rules, $E_1/a^{c_1} \to a^{p_1}$ and $E_2/a^{c_2} \to a^{p_2}$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules are applicable in a neuron, and in that case, only one of them is chosen and applied non-deterministically. However, note that, by definition, if a spiking rule is applicable, then no forgetting rule is applicable, and vice versa.

Thus, the rules are applied in the sequential manner in each neuron, at most one in each step, but neurons function in parallel with each other. It is important to notice that the applicability of a rule is established based on the total number of spikes contained in the neuron.

A configuration of the system is described by the number of spikes present in each neuron. 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 applied. The result of a computation is the number of steps elapsed between the first two spikes sent by the output neuron during the computation.

## 3 Matrix Representation of SN P System

In this section, a matrix representation of SN P system is given.

As mentioned in the above section, a configuration of the system is described by the number of spikes present in each neuron. Here, vectors are used to represent configurations.

**Definition 2 (Configuration Vectors).** *Let $\Pi$ be an SN P system with $m$ neurons, the vector $C_0 = (n_1, n_2, \dots, n_m)$ is called the* **initial configuration vector** *of $\Pi$, where $n_i$ is the amount of the initial spikes present in neuron $\sigma_i$, $i = 1, 2, \dots, m$ before the computation starts.*

*For any $k \in \mathbb{N}$, the vector $C_k = (n_1^{(k)}, n_2^{(k)}, \dots, n_m^{(k)})$ is called the* **$k$th configuration vector** *of the system, where $n_i^{(k)}$ is the amount of spikes in neuron $\sigma_i$, $i = 1, 2, \dots, m$ after the $k$th step in the computation.*

In order to describe which rules are chosen and applied in each configuration, *spiking vector* is defined.

**Definition 3 (Spiking Vectors).** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \le n < \infty$. Assume a total order $d : 1, \dots, n$ is given for all the $n$ rules, so the rules can be referred as $r_1, \dots, r_n$. A* **spiking vector** *$s^{(k)}$ is defined as follows:*

$$s^{(k)} = (r_1^{(k)}, r_2^{(k)}, \dots, r_n^{(k)}),$$

*where:*

$$r_i^{(k)} = \begin{cases} 1, \textit{ if the regular expression } E_i \textit{ of the rule } r_i \textit{ is satisfied} \\ \quad \textit{and the rule } r_i \textit{ is chosen and applied}; \\ 0, \textit{ otherwise.} \end{cases}$$

*In particular, $s^{(0)} = (r_1^{(0)}, r_2^{(0)}, \dots, r_n^{(0)})$ is called the* **initial spiking vector**.

In each configuration, when the chosen rules are applied, the change of the number of spikes in each neuron is represented by *spiking transition matrix.*

**Definition 4 (Spiking Transition Matrix).** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $d : 1, \dots, n$ a total order given for all the $n$ rules. The* **spiking transition matrix** *of the system $\Pi$, $M_\Pi$, is defined as follows:*

$$M_\Pi = [a_{ij}]_{n \times m},$$

*where:*

$$a_{ij} = \begin{cases} -c, \textit{ if rule } r_i \textit{ is in neuron } \sigma_j \textit{ and it is applied consuming } c \textit{ spikes;} \\ \ \ p, \textit{ if rule } r_i \textit{ is in neuron } \sigma_s\ (s \neq j \textit{ and } (s, j) \in syn) \\ \quad \textit{and it is applied producing } p \textit{ spikes;} \\ \ \ 0, \textit{ if rule } r_i \textit{ is in neuron } \sigma_s\ (s \neq j \textit{ and } (s, j) \notin syn). \end{cases}$$

In a spiking transition matrix, the row $i$ is associated with the rule $r_i : E/a^c \to a^p$. Assume that the rule $r_i$ is in neuron $\sigma_j$. When the rule $r_i$ is applied, it consumes $c$ spikes in neuron $\sigma_j$; neuron $\sigma_s$ ($s \neq j$ and $(j, s) \in syn$) receives $p$ spikes from neuron $\sigma_j$; neuron $\sigma_s$ ($s \neq j$ and $(j, s) \notin syn$) receives no spike from neuron $\sigma_j$. By the definition of spiking transition matrix, the entry in the position $(i, j)$ is a negative number; the other entries in the row $i$ are non-negative numbers. So the following observation holds:

**Observation 1** *Each row of a spiking transition matrix has exactly one negative entry.*

In a spiking transition matrix, the column $i$ is associated with neuron $\sigma_i$. For an SN P system, without loss of generality, it can be assumed that each neuron has at least one rule inside (if a neuron has no rule inside, it just stores spikes, sending no spikes to other neurons or environment, so it can be deleted without any influence to the computational result of the system). Assume the rules to be $r_m, r_n, \ldots$. The rules consume spikes in neuron $\sigma_i$ when they are applied. So the corresponding entries $(m, i), (n, i), \ldots$ in the spiking transition matrix are negative numbers, and the following observation holds:

**Observation 2** *Each column of a spiking transition matrix has at least one negative entry.*

## 4 Computing via Matrices

In this section, it is shown how the computation of SN P system can be represented by operations on matrices, by using the matrix representation defined in the above section.

First, we provide an example before we define formally the matrix operations for an SN P system.

*Example 1.* Let us consider an SN P system $\Pi = (\{a\}, \sigma_1, \sigma_1, \sigma_3, syn, out)$ that generates the set $\mathbb{N}$ of natural numbers excluding 1, where $\sigma_1 = (2, R_1)$, with $R_1 = \{a^2/a \rightarrow a, a^2 \rightarrow a\}$; $\sigma_2 = (1, R_2)$, with $R_2 = \{a \rightarrow a\}$; $\sigma_3 = (1, R_3)$, with $R_3 = \{a \rightarrow a, a^2 \rightarrow \lambda\}$; $syn = \{(1, 2), (1, 3), (2, 1), (2, 3)\}$; $out = 3$. $\Pi$ is also represented graphically in Figure 1, which may be easier to understand .

In order to represent the above SN P system $\Pi$ in a matrix, firstly, we set a total order for all the rules in the system, which can be seen in Figure 1. With this order, the rules can be denoted by $r_1, \ldots, r_5$.

Let $M_{\Pi_1} = [a_{ij}]_{5\times 3}$ be the spiking transition matrix for $\Pi$. As defined in Section 3, row $i$ of $M_\Pi$ is associated with rule $r_i : E/a^c \rightarrow a^p, c \geq 1, p \geq 0$ in system $\Pi$. The entries $a_{i1}, a_{i2}, a_{i3}$ are the amount of spikes which neurons $\sigma_1, \sigma_2, \sigma_3$ will get (or consume) when rule $r_i$ is applied.

We have the following spiking transition matrix for the SN P system $\Pi$ in Figure 1.

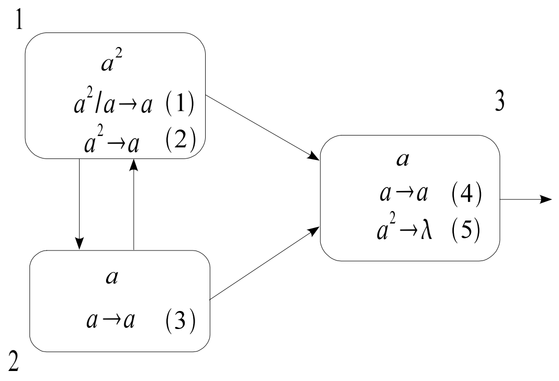

**Fig. 1.** An SN P system $\Pi$ that generates the set $\mathbb{N} - \{1\}$

$$M_\Pi = \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} \tag{1}$$

Initially, neurons $\sigma_1$, $\sigma_2$, $\sigma_3$ have 2, 1, 1 spikes, respectively. According to Definition 2, the initial configuration vector for system $\Pi$ would be $C_0 = (2, 1, 1)$. We also get the initial spiking vector by Definition 3: since neuron $\sigma_1$ has two rules $r_1$ and $r_2$ that could possibly be applied in the initial transition, one of the rules could be chosen. The initial spiking transition vector would be $(1, 0, 1, 1, 0)$ or $(0, 1, 1, 1, 0)$. Note here that we cannot use rule $r_5$ because the regular expression $a^2$ is not satisfied in neuron $\sigma_3$.

If rule $r_1 : a^2/a \to a$ is applied, it consumes one spike in neuron $\sigma_1$ and sends 1 spike to neurons $\sigma_2$ and $\sigma_3$, respectively; at the same time, neuron $\sigma_2$ sends 1 spike to neurons $\sigma_1$ and $\sigma_3$. In this step, the net gain of neuron $\sigma_1$ is 0 spike (it consumes 1 spike by $r_1$ and receives 1 spike from neuron $\sigma_2$); the net gain of neuron $\sigma_2$ is 0 spike (it consumes 1 spike by $r_3$ and receives 1 spike from neuron $\sigma_1$); the net gain of neuron $\sigma_3$ is 1 spike (it consumes 1 spike by rule $r_5$ and receives 2 spikes from neurons $\sigma_1$ and $\sigma_2$, respectively). After this step, the numbers of spikes in neurons $\sigma_1$, $\sigma_2$, $\sigma_3$ are 2, 1, 2, respectively.

Our illustration above served as witness to the following results.

**Definition 5 (Transition Net Gain Vector).** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \leq n < \infty$, $C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)})$ the kth configuration vector of an SN P system $\Pi$. We define*

$$NG^{(k)} = C_{k+1} - C_k,$$

*as the* **transition net gain vector** *at step $k$.*

**Lemma 1.** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the $n$ rules, $M_{\Pi}$ the spiking transition matrix of $\Pi$, $s^{(k)}$ the spiking vector at step $k$. Then the transition net gain vector at step $k$ can be obtained by*

$$NG^{(k)} = s^{(k)} \cdot M_{\Pi}. \tag{2}$$

*Proof.* Equation (2) implies that $NG^{(k)} = (g_1, g_2, \ldots, g_m)$, such that $g_j = \Sigma_{i=1}^{n} r_i^{(k)} a_{ij}$, for all $j = 1, 2, \ldots, m$. Note that the spiking vector $s^{(k)}$ is a $\{0, 1\}$-vector that identifies the rules in each neuron that would be applied at step $k$. Thus, $g_j$ represents the total amount of spikes obtained and consumed by neuron $\sigma_j$ after applying the rules identified by $s^{(k)}$. Therefore, we have $(n_1^{(k+1)}, n_2^{(k+1)}, \ldots, n_m^{(k+1)}) = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)}) + (g_1, g_2, \ldots, g_m)$, that is, $C_{k+1} = C_k + NG^{(k)}$.

**Theorem 1.** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the $n$ rules, $M_{\Pi}$ the spiking transition matrix of $\Pi$, $C^{(k)}$ the kth configuration vector, $s^{(k)}$ the spiking vector at step $k$, then every configuration $C_k$ of $\Pi$ can be obtained by*

$$C_k = C_{k-1} \ + \ s^{(k-1)} \cdot M_{SNP}. \tag{3}$$

*Proof.* This results follows directly from the preceding Lemma.

Let us go back to the example show in Figure 1. Given the initial configuration vector $C_0 = (2, 1, 1)$, the next configuration of system $\Pi$ can be computed as follows:

If we choose the rules $r_1, r_3, r_4$ to apply, the spiking vector will be $s^{(0)} = (1, 0, 1, 1, 0)$, then the next configuration can be obtained by:

$$C_1 = (2 \quad 1 \quad 1) \ + \ (1 \quad 0 \quad 1 \quad 1 \quad 0) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2 \quad 1 \quad 2) \tag{4}$$

In the next step, we choose $r_1, r_3, r_5$ to apply, the spiking vector is $(1, 0, 1, 0, 1)$, then the next configuration is:

$$C_2 = (2 \quad 1 \quad 2) \ + \ (1 \quad 0 \quad 1 \quad 0 \quad 1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (2 \quad 1 \quad 2) \tag{5}$$

So, if we choose the spiking vector to be $(1, 0, 1, 0, 1)$, the transition net gain vector will be:

$$NG = (1 \quad 0 \quad 1 \quad 0 \quad 1) \begin{pmatrix} -1 & 1 & 1 \\ -2 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -2 \end{pmatrix} = (0 \quad 0 \quad 0) \tag{6}$$

Equation (6) means that the configuration of the system will remain unchanged as long as we choose rules $r_1$, $r_3$ and $r_5$ to apply. However, at any moment, starting from the first step of computation, neuron $\sigma_1$ can choose to use the rule $r_2 : a^2 \to a$, in this case system will go to another configuration, we do not want to check the computation in detail here but leave this task to the readers.

Finally, the following Corollary is a direct consequence of the preceding Theorem.

**Corollary 1.** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the $n$ rules, $M_\Pi$ the spiking transition matrix of $\Pi$, $C^{(k)}$ the kth configuration vector, $s^{(k)}$ the spiking vector at step $k$, the previous configuration $C_{k-1}$ can be obtained by*

$$C_{k-1} = C_k \ - \ s^{(k-1)} \cdot M_\Pi. \tag{7}$$

In our matrix representation, we do not include the environment. This idea can be incorporated in the definition of a spiking transition matrix by introducing a so-called **augmented spiking transition matrix.**

**Definition 6 (Augmented Transition Spiking Matrix).** *Let $\Pi$ be an SN P system with $m$ neurons and $n$ rules, $m \leq n < \infty$, $d : 1, \ldots, n$ a total order for the $n$ rules, $M_\Pi$ the $n \times m$ spiking transition matrix of $\Pi$. We define an* **augmented spiking transition matrix** *as follows:*

$$[M_\Pi \mid e]_{n \times (m+1)},$$

*where $e = (e_1, e_2, \ldots, e_n)^T$ is a column representing environment, where:*

$$e_i = \begin{cases} p, & \text{if rule } r_i \text{ is in the output neuron and it is applied producing } p \text{ spikes;} \\ 0, & \text{if rule } r_i \text{ is not in the output neuron.} \end{cases}$$

Correspondingly, we define

$$C_k = (n_1^{(k)}, n_2^{(k)}, \ldots, n_m^{(k)}, n_e^{(k)}),$$

to be **augmented configuration vector** after the $k$th step in the computation, where $n_i^{(k)}$ is the amount of spikes in neuron $\sigma_i$, for all $i = 1, 2, \ldots, m$, $n_e^{(k)}$ is the amount of spikes collected by environment. Using this vector instead of the configuration vector defined in Section 3 simply allows us to monitor the output of a system.

## 5 Conclusions and Remarks

In this paper, we have found a universal algebraic representation for SN P systems: for every SN P system without delay, configuration vectors are defined to represent the number of spikes in neurons; spiking vectors are used to denote which rules will be applied; a spiking transition matrix is used to describe the skeleton of system. Such algebraic representation turns out to be a reasonable representation of SN P systems. It is shown that matrix computation is convenient for deciding the next configuration of our systems.

It is not difficult to see that such matrix representation is also suitable for other variants of SN P systems, such as asynchronous SN P systems [1], SN P systems with exhaustive use of rules [4], and so on. The spiking transition matrix is related to the structure of system only, so that the elements of the matrix are determined initially. During the computation of a system, it is only necessary to decide the spiking vector by checking the current configuration vector and the regular expressions of rules. Thus, it is easy to program for computer application, which can offer as a powerful tool for the simulation and analysis of these systems.

Anyway, there are many issues still worth investigating. Another research line would be of interest to see whether matrix can be used for biological neural networks, for example, human brain. A problem is how to capture the feature that neurons have refractory time (in the present paper, we have not considered this feature in our model).

**Acknowledgements**

The work of X. Zeng and L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60533010), 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).

## References

1. M. Cavaliere, O. Egecioglu, O. H. Ibarra, S. Woodworth, M. Ionescu, and Gh. Păun, Asynchronous spiking neural P systems, *Theoretical Computer Science*, 2009, 410, 2352–2364.
2. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, Computing backwards with P systems, *WMC10*, Curtea de Arges, Romania, 2009, 282–295.
3. M. Ionescu, Gh. Păun, and T. Yokomori, Spiking neural P systems. *Fundamenta Informaticae*, 2006, 71(2–3), 279–308.
4. M. Ionescu, Gh. Păun, and T. Yokomori, Spiking neural P systems with exhaustive use of rules, *International Journal of Unconventional Computing*, 2007, 3, 135–154.
5. J. K. Nelson and J. C. McCormac, Structural Analysis: Using Classical and Matrix Methods, 3rd Edition, Wiley, 2003.

6. Gh. Păun, Computing with membranes, *Journal of Computer and System Science*, 2000, 61(1), 108–143.
7. Gh. Păun, Membrane Computing – An Introduction, Springer-Verlag, Berlin, 2002.
8. The P System Web Page: http://ppage.psystems.eu.