Spiking Neural P systems with colored spikes and multiple channels

José M. Sempere[1,2] and Rodrigo Llanes[1]

[1] Valencian Research Institute for Artificial Intelligence (VRAIN)
[2] Valencian Graduate School and Research Network of Artificial Intelligence (VALGRAI)
Universitat Politècnica de València, Spain
jsempere@dsic.upv.es

Abstract. In this paper, we combine two variants of Spiking Neural (SN) P systems: we use a non-singleton alphabet of spikes (colored spikes), and we use multiple channels that can be used in the rules of the neurons. The completeness of the proposed model is shown by the simulation of register machines. We also show the equivalence between the proposed model and other different ones, such as virus machines and neural-like P systems with plasmids. We show the simulations of the latter models by our proposed one. Hence, we can consider that our proposed model is a unifying one that can achieve two main properties: different encodings for the information (spikes), and different communication topologies (channels).

Keywords: SN P systems, colored spikes, multiple channels, completeness, virus machines, neural-like P systems with plasmids

1 Introduction

Membrane computing is a research area founded by Gheorghe Păun at the end of the last century [24], and it is a significant part of the natural computing paradigm [11]. The source of inspiration for membrane computing is the eukaryotic living cell whose internal structure allows regions separated by membranes, where different molecular reactions are carried out by the biomolecular machinery of the cell. Membrane computing models are known as P systems. Initially, P systems were proposed as cell-like transition P systems [23]. Then, tissue P systems were proposed through the association and abstraction of cell-like P systems [15]. A very few years later, the spiking neural (SN) P system was proposed [10] that allows rate and pulse encoding [8]. It was proposed by introducing artificial spiking neural networks in the framework of membrane computing.

Interest on SN P systems has increased in the last years, both when it comes to proposing new variants or new features to the basic model [12, 2], and when it comes to applying it to solve problems in many different areas [6] . Among the proposals that have been made during the last years, we could mention fuzzy reasoning spiking neural P systems (FRSNP) [28], Optimization Spiking neural

P Systems (OSNPS) [39], Spiking neural P systems with structural plasticity [3], electrical synaptic transmission-based spiking neural P systems (ESTSNPS) [36], Spiking Neural P Systems with budding rules [9], Spiking neural P systems with neuron division and budding [19], time-free Spiking Neural P Systems [21], and Spiking Neural P Systems with Astrocytes [20]. Two recent references on spiking neural P systems, and its variants and applications are [26, 40].

In this work, we use SN P systems that allows the sending/receiving of different spikes through different channels. The use of different channels in the rules is not novel: it was first proposed by Alhazov *et al.* [1], and referred again in [38]. It has been proposed again in [27], where the authors use different spiking channels to connect the neurons in the network. An increasing interest on the use of multiple channels has been shown in the last years [14]. So, for example, Miao *et al.* proposed the use of multiple channels with dynamic adjustment of the rules [16], while in [18], Ning *et al.* propose a combination of multiple channels with autapses [35].

The use of a non-singleton alphabet to define the spikes of the system (colored spikes) is not novel given that it was proposed in [34], and it has been recently used to solve NP-complete problems [22].

In this work, we combine the two previously referred features: multiple channels and colored spikes. We integrate them in the rules of the SN P system. So, the rules can manage different spikes at every moment, and the spikes can be sent to other neurons through different channels. These characteristics, when combined appropriately, provide us with an efficient way to simulate other models proposed in the field of membrane computing, such as virus machines [5], and neural-like P systems with plasmids [4]. As far as we are aware, the use of SN P systems to simulate these models is non-existent or has never even been considered. Furthermore, we believe that our proposal could become a general computing framework in which several networks operate simultaneously, influencing one another, with nodes (neurons) specialising in different types of information.

The structure of this work is as follows: In section 2, we provide the basic concepts and definition for membrane computing and spiking neural P systems. In section 3, we propose our model that combines a non-singleton alphabet for spikes, and multiple channels in the rules. We establish the completeness of the proposed model by simulating register machines. In section 4, we propose efficient simulations of other computational models by our model. We propose a simulation for virus machines, and a simulation for bacterial computing with plasmids. Finally, in section 5, we outline some conclusions of our work, and we point out to future research works.

2 Basic concepts and definitions

In the following, we assume that the reader is familiar with the basic concepts of formal language theory [32]. Nevertheless, we will introduce additional concepts on multisets, strings and languages.

Let $\mathbb{N}$ denotes the set of natural numbers. Given a set V, a *multiset* M over V is defined as a mapping $M : V \to \mathbb{N}$, where for every $a \in V$, $M(a)$ denotes the multiplicity of the element a in M. Given two multisets over V, M_1 and M_2, we will say that $M_1 \subseteq M_2$ iff $(\forall a \in V)\ M_1(a) \leq M_2(a)$. Given an alphabet Σ, and the string $x \in \Sigma^*$, we denote the number of symbols $a \in \Sigma$ in x by $|x|_a$. Given the alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$, the Parikh projection of a string $x \in \Sigma^*$, is the vector $\Psi_\Sigma(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. Given any language $L \subseteq \Sigma^*$, the Parikh projection of L in Σ is the set $\Psi_\Sigma(L) = \{\Psi_\Sigma(x) : x \in L\}$. For any set $V = \{a_1, a_2, \ldots, a_n\}$, any multiset M over V can be represented by the string $x = {a_1}^{M(a_1)}{a_2}^{M(a_2)} \ldots {a_n}^{M(a_n)}$ (observe that any permutation in the symbols of x represents the multiset M).

Now, we introduce the definition of a register machine, that is a universal model of computation proposed in [17].

Definition 1. A register machine is defined by $M = (m, H, l_0, l_h, I)$, where:

1. $m \geq 1$ indicates the number of machine registers. Every register holds a positive integer value, and all the registers are initialized to zero.
2. H is a set of instruction labels.
3. l_0 is the label corresponding to the initial instruction.
4. l_h is the label corresponding to the halt instruction.
5. I is the set of instructions such that $|I| = |H|$ and each label from H refers to an instruction in I. Each of the instructions of I is in one of the following forms:
   - $l_i : (\text{ADD}(r), l_j, l_k)$ add one to the register r, and then non-deterministically jumps to instruction l_j or l_k.
   - $l_i : (\text{SUB}(r), l_j, l_k)$ subtract one if the contents of register r is greater than zero, and jump to l_j. Otherwise, jump to l_k.
   - HALT finishes the computation.

□

The register machine begins its execution by the instruction labeled l_0, then it executes the instruction and it jumps to the next one according to the type of instruction and the value of the registers. This process is repeated until it reaches the halt instruction l_h. There is a predefined register where the result of the computation is stored when the machine halts. Usually, the first register is taken as the output register.

Basic concepts and results about P systems and membrane computing can be consulted at [25], while basic definitions related to Spiking Neural P systems can be taken from [10].

Definition 2. A spiking neural P system (SN P system, for short) of degree $m \geq 1$ is defined by $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where:

1. $O = \{a\}$ is a singleton alphabet of spikes
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ 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 in σ_i.

(b) R_i is a finite set of rules in one of the following two forms:
- firing or spiking rules: $E/a^c \rightarrow a; d$ where E is a regular expression over O, and $c \geq 1,\ d \geq 0$ are integer numbers. We omit E whenever it be equal to a^c, and we omit d whenever it be equal to 0.
- forgetting rules: $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each spiking rule $E/a^c \rightarrow a; d$ then $a^s \notin L(E)$ ($L(E)$ is the regular language defined by E)

3. $syn \subseteq \{1, 2, \ldots m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of synapses between neurons
4. $in,\ out\ \in \{1, \ldots, m\}$ indicate the input and the output neurons of Π.

□

Observe that we can omit the definition of the input neuron in order to define the SN P system as a generator model.

At neuron σ_i, the firing rules $E/a^c \rightarrow a; d$ are applied as follows: if the neuron contains $k \geq c$ spikes and $a^k \in L_E$ (the language denoted by the expression E) then c spikes are removed from σ_i and one spike is delivered to all the neurons σ_j connected to σ_i with $(i, j) \in syn$. If $d = 0$ the spike is immediately emitted, otherwise it is emitted after d computation steps (during these computation steps, the neuron is closed, so it cannot receive spikes, it cannot apply the rules and, subsequently, it cannot send new spikes). At neuron σ_i, the forgetting rule $a^s \rightarrow \lambda$ is applied as follows: if the neuron σ_i contains exactly s spikes and no firing rule can be applied then all the spikes of the neuron are removed.

A configuration of the system at time t during a computation is defined by the tuple $(i_1/t_1, \ldots, i_m/t_m)$ that denotes the number of spikes that are at every neuron together with the computation time needed to open the neuron. The initial configuration of the SN P system is $(n_1/0, \ldots, n_m/0)$. During a computation, the step when a spike is emitted by the output neuron can be marked by '1' while the other steps are marked by '0'. The binary sequence that is obtained in such a way during the computation is called the spike train of the system. It should be noted that there are multiple ways to read the system output. For example, in generator systems, if the computation does not finish, the output is the number of spikes in the neuron *out* at each computation step. The total number of spikes that the system has sent out to the environment during a computation (which must be finite) can also be considered as output.

Example 1. In the figure 1 we show an example of a SN P system with three neurons. The output neuron is neuron 3. The input neuron is neuron 1. Neuron 1 has n initial spikes and one firing rule. Neurons 2 and 3 have no initial spikes and one firing rule.

In the first iteration, only the rule $a^+/a \rightarrow a$ of neuron 1 can be applied. It consumes one spike and sends a spike to neurons 2 and 3. In the following $n-1$ iterations, the three neurons will be active. Observe that along the time, neuron 3 receives $2 \times n$ spikes that will be sent to the environment. Once all these spikes have been consumed the system halts and it has generated $2 \times n$ spikes.

□

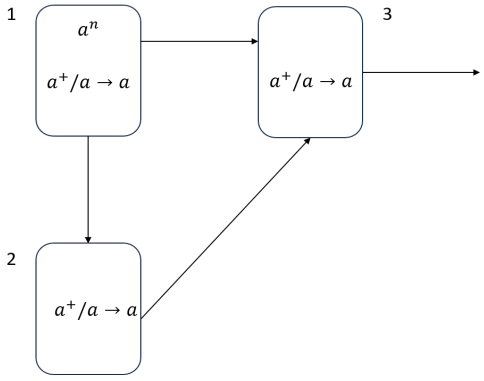

Fig. 1. An example of an SN P system that generates $2 \times n$.

Once we have defined the SN P system, we introduce two variants of it: multiple channels [27], and a spike alphabet with more than one symbol (colored spikes) [34].

Definition 3. [27] A spiking neural P system with multiple channels (SNP-MC system, for short) of degree $m \geq 1$ is defined by $\Pi = (O, L, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, out)$, where:

1. $O = \{a\}$ is a singleton alphabet of spikes
2. $L = \{1, 2, \ldots, N\}$ is the alphabet of channel labels
3. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, L_i, R_i)$, $1 \leq i \leq m$, where:
   (a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
   (b) $L_i \subseteq L$ is a finite set of channel labels used in the neuron
   (c) R_i is a finite set of extended rules in the form $E/a^c \to a^p(l)$ where E is a regular expression over O, and $c \geq 1,\ p \geq 0, l \in L_i$.
4. $syn = \{(i, j, l)\} \subseteq \{1, 2, \ldots m\} \times \{1, 2, \ldots, m\} \times L$ with $(i, i, l) \notin syn$ for $1 \leq i \leq m$, and $l \in L$ (synapse connections);
5. $out \in \{1, \ldots, m\}$ is the output neuron.

□

Observe that the rules of the system are defined in a similar way as they have been described in the definition 1. The forgetting rules can be established as those extended rules with $p = 0$ and no channel reference. Every rule establishes the channels that it uses to send the spikes. The spikes are only sent to the connected neurons by the channel established in the rule. In addition, in each computation step there can be some competition between different rules at the neuron. Provided that more than one rule can be applied, then the system selects only one of them non-deterministically. So, the neurons work in sequential manner, that is they apply only one rule at every computation step.

It has been proved that SNP-MC systems are universal, even if they work in a sequential manner [14].

Definition 4. [34] A spiking neural P system with colored spikes (SNP-CS system, for short) of degree $m \geq 1$ is defined by $\Pi = (C, O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where:

1. $C = \{1, 2, \ldots g\}$ is a finite set of colors to mark the color of a spike. Every spike is associated with a unique color
2. $O = \{a_1, a_2, \ldots, a_g\}$ is an alphabet of g colored spikes
3. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (< n_1^i, n_2^i, \ldots, n_g^i >, R_i)$, $1 \leq i \leq m$, where:
   (a) $n_h^i \in \mathbb{N}$ is the number of spikes $a_h \in O$ initially placed in neuron σ_i.
   (b) R_i is a finite set of spiking and forgetting rules in the form:
      – Spiking rule: $E/a_1^{c_1} a_2^{c_2} \ldots a_g^{c_g} \rightarrow a_1^{p_1} a_2^{p_2} \ldots a_g^{c_g}; d$ where E is a regular expression over O, and $c_i, p_i \geq 0$;
      – Forgetting rule: $a_1^{s_1} a_2^{s_2} \ldots a_g^{s_g} \rightarrow \lambda$ where $a_1^{s_1} a_2^{s_2} \ldots a_g^{s_g} \notin L(E)$ for any regular expression E associated with a spiking rule;
4. $syn \subseteq \{1, 2, \ldots m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$;
5. $in, out \in \{1, \ldots m\}$ are the input and the output neurons

□

Observe that the SNP-CS systems are a generalization of the SN P systems. The functioning of the rules in the neurons, the behavior of the SNP-CS system and its output are generalizations of the SN P systems. Hence, the model is computationally complete. Note that, in this case, the membership problem associated with any regular expressions E in the spiking rules should be replaced by a membersip problem within the Parikh projection $\Psi_O(L_E)$.

3 Spiking neural P systems with multiple channels and colored spikes

We introduce a new model of SN P systems that combines the use of multiple channels, together with a non-singleton alphabet of spikes. We define different ways of obtaining the system outputs, and we will prove the completeness of the model by proposing a simulation of any register machine program.

Definition 5. A spiking neural P system with multiple channels and colored spikes (SNP-MC-CS system, for short) of degree $m \geq 1$ is defined by $\Pi = (O, L, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in)$, where:

1. $O = \{a_1, a_2, \ldots, a_g\}$ is an alphabet with g colored spikes
2. $L = \{1, 2, \ldots, k\}$ is an alphabet of channel labels
3. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (w_i, R_i)$, $1 \leq i \leq m$, where:
   (a) $w_i \in O^*$ is a string that denotes the initial multiset of spikes in the neuron.
   (b) R_i is a finite set of rules in the forms:
      – Firing rules: $E/w_c \rightarrow w_1\,(l_1)\, w_2\,(l_2) \ldots w_n\,(l_n)\,; d$ where E is a regular expression over O, and $w_c, w_i \in O^*$ $1 \leq i \leq n, l_i \in L$, $d \geq 0$.

- Forgetting rules: $w_c \to \lambda$, where $w_c \in O^*$.

4. $syn \subseteq \{1, 2, \dots m\} \times \{1, 2, \dots, m, out\} \times L$ is the synapse connections with channels where $(i, i, l) \notin syn$ for $1 \leq i \leq m$ and $\forall\ l\ \in L$. Observe that (i, out, l) denotes that the neuron σ_i sends the spikes out to the environment by the channel l.
5. $in\ \in \{1, \dots, m\}$ is the input neuron. Observe that the input neuron can be omitted whenever the system is a generator.

□

The firing rules $E/w_c \to w_1\,(l_1)\,w_2\,(l_2) \dots w_n\,(l_n)\,;d$ of the neuron σ_i can be applied whenever $\psi_O(x_i) \in \Psi_O(L_E)$ and $w_c \subseteq x_i$, where x_i denotes the spikes in the neuron σ_i and L_E is the language denoted by the regular expression E. In such a case, the spikes denoted by w_c are removed from the neuron, and the spikes denoted by w_i are sent to the connected neurons (or to the environment) by the channel l_i according to the synaptic connections of the neuron. If the delay $d = t > 0$, then the neuron is blocked, and it can neither send nor receive spikes after t computation steps.

The forgetting rules of the neuron σ_i in the form $w_c \to \lambda$, can be applied whenever no firing rule is applicable and $w_c \subseteq x_i$, where x_i denotes the spikes in the neuron σ_i. In such a case, the spikes denoted by w_c are removed from the neuron.

The system runs in a maximally parallel manner, that is, at each computational step, every neuron that has some applicable rule applies it. A rule is applicable iff the spikes in the neuron meet the regular expression in the rule, and the number of spikes are greater or equal to the number of spikes to be removed. In the case of forgetting rules, the regular expression condition is not considered. If there are more than one applicable rule, the firing rules will have priority over those of forgetting. If there are more than one applicable firing rule in conflict, it means that they require common spikes, then, only one of them is non-deterministically selected to be applied. If a rule with a delay is applied, the neuron will be blocked just as in SN P systems. If several rules with delays are applied at the same computational step, the neuron will be blocked for the maximum delay value of the applied rules. Finally, it should be noted that the computation finishes whenever there is no blocked neuron, and no rule can be applied in any neuron.

Regarding the output of the system, we will propose four different ways to obtain it, as follows:

- *halting mode*: The result of the computation is the number of spikes that have been sent to the environment during the computation, no matter their colors. The output is defined only when the system halts.
- *multi-channel halting mode*: As in the halting mode, the output of the model is the set of spikes sent to the environment during the computation. The output spikes are defined for the channels through which they have been sent. In this case, the output is a vector of natural numbers.

- *temporary mode*: This mode is the one used for generating systems, since it is not necessary reaching a halting configuration, the system output will be read as the sequence of spikes sent to the environment at every computation step (a spike train).
- *multi-channel temporary mode*: It can be used for signal processing, as it allows reading the output as the sequence of spikes (a spike train) sent to the environment at every computation step for each of the channels.

Example 2. In the figure 2, we show an example of an SNP-MC-CS system with two neurons, 1 (left) and 2 (right). The system uses three channels, and it does not use any input neuron, so it is a generative model. In the first computation step, the system can only execute the rule in the neuron 2, sending a spike a to the neuron 1 through channel (1). In the second computation step, the following rules can be chosen non-deterministically: $a \to a(0), ab(2)$, so spikes a and b will be sent to the environment through the channel (2), and a spike a is sent to the neuron 2 through the channel (0). So in the next computation step the system will repeat the initial configuration, the second rule, $a \to ab(2)$ sends the spikes a and b to the environment and the system halts, given that no rule can be applied again. The system outputs in a halting mode the multiset defined by $\Psi_{\{a,b\}}(\{a^n b^n : n \geq 1\})$.

□

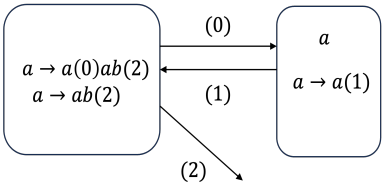

Fig. 2. An example of an SNP-MC-CS system.

3.1 Completeness

In the following, we show that the SNP-MC-CS systems are complete models of computation. It is important to highlight that SNP-CS systems are universal [34], and they are a particular case of SNP-MC-CS with a single channel. On the other hand, SNP-MC are also universal [27], and they are a particular case of SNP-MC-CS with a single color. So, the completeness of the SNP-MC-CS system is trivially deduced from the referred previous works, as established in the following theorem.

Theorem 1. SNP-MC-CS systems are universal models of computation.

Proof. Trivially deduced from [27] and [34]. □

Nevertheless, we propose the simulations of the basic instructions of register machines by SNP-MC-CS systems. In the following, we associate a neuron with label l_i with each instruction with a label l_i, and a neuron with label r with the register r in the machine. If the contents of any register is a natural number p, the multiset c^p will be stored in its associated neuron. The activation of any instruction will be triggered by the spike a. In addition, other spikes are used to simulate the SUB instructions in the registers. The HALT instruction will be managed by using an additional spike h.

Now, we will define a scheme to implement every basic instruction of the register machine as a SNP-MC-CS sub-system.

– Simulation of the instruction $l_i : (\text{ADD}(r), l_j, l_k)$
The ADD instruction can be simulated by the system proposed in the figure 3. The simulation works as follows: upon receiving a spike a in neuron l_i (which represents the instruction with the same label), one of the two rules is executed non-deterministically. Observe that in both rules a spike c is sent to the neuron r through the channel (1). It adds one unit to the contents of the register r. A spike a is sent to the neurons to l_j or l_k in order to activate the following instruction in a non-deterministic manner.

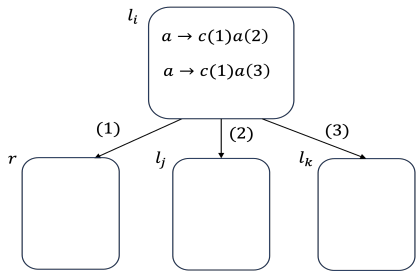

Fig. 3. An SNP-MC-CS system to simulate $l_i : (\text{ADD}(r), l_j, l_k)$.

– Simulation of the instruction $l_i : (\text{SUB}(r), l_j, l_k)$
The SUB instruction can be simulated by following the scheme of the figure 4. Upon receiving a spike a, the neuron l_i consumes it and it sends a spike d to the neuron r that represents the register r. In the neuron r, the subtraction is carried out by the proposed rules. Observe that the first rule is applied only if the contents of the register is greater than zero, while the second rule is reserved for the zero case. A spike a is sent to activate the following instruction l_j, or l_k.
Now, we must analize the case where at least two SUB instructions operate over the same register r. In such a case, additional rules and colored spikes are needed. For example, let us consider that, in the register machine program, the instructions $l_i : (\text{SUB}(r), l_j, l_k)$ and $l_m : (\text{SUB}(r), l_n, l_p)$ are used. Then,

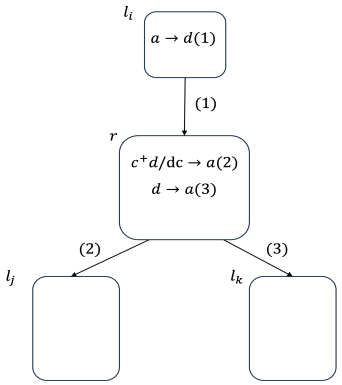

Fig. 4. An SNP-MC-CS system to simulate the instruction $l_i : (\mathrm{SUB}(r), l_j, l_k)$.

we can apply the scheme proposed in the figure 5. In such a case, we use different spikes to differentiate the two instructions: the spike d is used to activate the first SUB instruction, while the spike e is used to activate the other SUB instruction.
The scheme of the figure 5 can be enlarged as many times as necessary to simulate different SUB instructions applied to the same register.

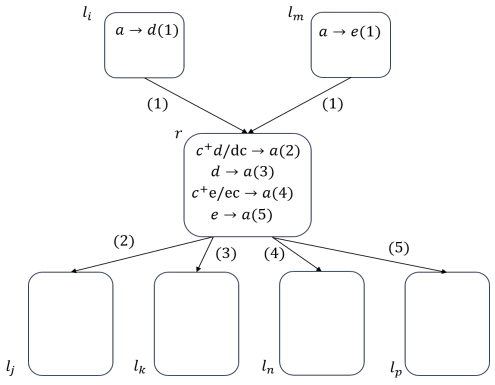

Fig. 5. An SNP-MC-CS system to simulate two instructions $l_i : (\mathrm{SUB}(r), l_j, l_k)$ and $l_m : (\mathrm{SUB}(r), l_n, l_p)$.

– Simulation of a HALT instruction
The simulation of a HALT instruction can be carried out by following the scheme of the figure 6. Upon receiving a spike a, the neuron l_h sends a spike h to the output register r whose contents is the output of the simulated

register machine. The neuron r receives the activating spike, and it consumes sequentially all the spikes c that are used to represent the value of the register. For every consumed spike, a copy is sent out to the environment. Finally, the forgetting rule consumes the remaining spike h and the computation halts since there is no spike that could activate another instruction. Observe that the output is obtained in a halting mode

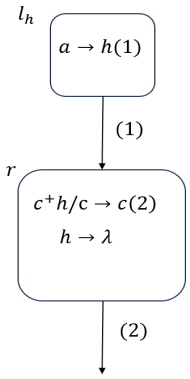

Fig. 6. An SNP-MC-CS system to simulate the HALT instruction.

4 Simulating other computation models by SNP-MC-CS systems.

Once we have proposed a new SN P system model, and we have shown its completeness, we will see in this section how the proposed model is able to simulate other ones that have been proposed in the field of membrane computing. In this way, the SNP-MC-CS model aims to be a unifying framework for other alternative models.

In the previous section, we have already seen that the model generalizes to models with multiple channels (SNP-MC systems), and, also, to models with multiple types of spikes (SNP-CS systems). Now, we will see how the model simulates virus machines [5] and neural-like P systems with plasmids [4].

4.1 Virus machines

Virus machines were first introduced in [37], and they have been used to compute recursive functions [31]. They have been proposed to generate, compute or recognize complex sets [29] and, recently, some characterizations of the number sets generated by the model have been studied [30].

In order to propose a simulation for virus machines by SNP-MC-CS systems, we provide a basic definition about the model as follows.

Definition 6. [5] A virus machine (VM, for short) of degree (p, q), with $p, q \geq 1$, is a tuple $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ where:

1. $\Gamma = \{v\}$ is a singleton alphabet;
2. $H = \{h_1, \ldots, h_p\}$ and $I = \{i_1, \ldots, i_q\}$ are ordered sets such that $v \notin H \cup I$ and $H \cap I = \emptyset$;
3. $D_H = (H \cup \{h_{out}\}, E_H, w_H)$ is a weighted directed graph, where $E_H \subseteq H \times (H \cup \{h_{out}\})$, $(h, h) \notin E_h$ for each $h \in H$, *out-degree*$(h_{out}) = 0$, and w_H is a mapping from E_H onto $\mathbb{N}\backslash\{0\}$ (the set of positive integer numbers);
4. $D_I = (I, E_I, w_I)$ is a weighted directed graph, where $E_I \subseteq I \times I$, w_I is a mapping from E_I onto $\mathbb{N}\backslash\{0\}$ and, for each vertex $i_j \in I$, the out-degree of i_j is less than or equal to 2;
5. $G_C = (V_C, E_C)$ is an undirected bipartite graph, where $V_C = I \cup E_H$, being $\{I, E_H\}$ the partition associated with it. In addition, for each vertex $i_j \in I$, the degree of i_j in G_C is less than or equal to 1.
6. $n_j \in \mathbb{N}$ $(1 \leq j \leq p)$ and $i_1 \in I$
7. $h_{out} \notin I \cup \{v\}$ and h_{out} is denoted by h_0 in the case that $h_{out} \notin H$.

□

In the figure 7, we can see the structure of a virus machine.

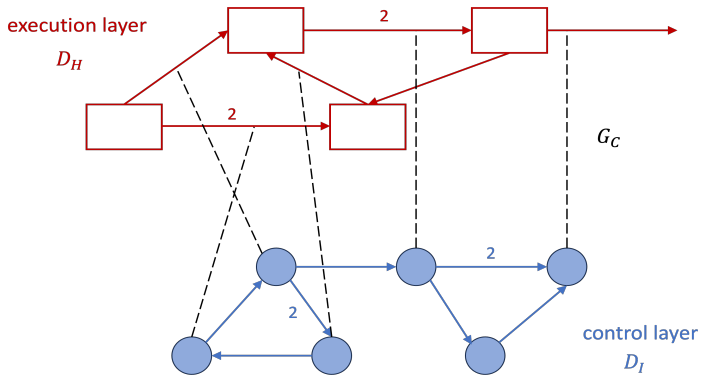

Fig. 7. The structure of a virus machine. In the top side we have an execution layer, in the bottom side we have a control layer, and both layers are connected by control executions.

We explain how a virus machine works: The nodes of the upper graph D_H represent the hosts that contain viruses. The transmission and replication of viruses is done through the connections between these nodes. The weights of these connections represent how many copies of the virus are transmitted to the

receiving host (i.e., the weight $w_{s,t}$ means that whenever the host s sends a copy of the virus to the host t, it replicates $w_{s,t}$ viruses in t). The connections between the nodes from D_H are transmission channels that can be activated depending on the configuration in the D_I graph, which is a graph of instructions. Each node in the D_I graph activates the transmission channel of the D_H graph depending on the connections that are specified in the G_C graph. Once an instruction from D_I has been executed, the next instruction to be executed will depend on whether viruses have been transmitted in the D_H graph, and it will also depend on the out-degree of the last executed instruction. If there is more than one possible following instruction, provided that weights that connect the instructions are equal, the next instruction will be chosen non-deterministically. Otherwise, the next instruction will be selected according to the highest weight (if the viruses have been transmitted) or the smallest weight (if the viruses have not been transmitted). If there is no following instruction, then the next instruction will be interpreted as a halting instruction. The output of the system will be the number of viruses collected in the output host of the D_H graph at the halting configuration, or in the environment if this had been predefined. For a greater level of detail on virus machines, we suggest the reader consult [5].

It has been proved that virus machines are a universal model of computation given that they can simulate the register machines [5].

Now, we provide a result that allows VM simulation by SNP-MC-CS systems.

Theorem 2. For every virus machine there exists an equivalent SNP-MC-CS system.

Proof. Let $\Pi = (\Gamma, H, I, D_H, D_I, G_C, n_1, \ldots, n_p, i_1, h_{out})$ be a VM. We can apply the following steps to obtain an equivalent SNP-MC-CS system:

1. Create a spike v that represents a virus and three auxiliary spikes s, f and a_i.
2. For all $h_i \in H$ create a neuron h_i
3. For all $i_j \in I$ create a neuron i_j
4. For each edge $(h_s, h_t) \in E_H$ associated with an instruction i_k:
   (a) Create a channel c_{h_s,h_t} that represents the communication channel between hosts h_s and h_t.
   (b) Create a channel c_{i_k} that represents the control relationship between (h_s, h_t) and the instruction i_k.
   (c) Add a synapse from neuron h_s to neuron h_t within a channel c_{h_s,h_t}.
   (d) Add a synapse from neuron h_s to neuron i_k within channel c_{i_k}, and another one in the opposite direction.
   (e) Create a spike a_{h_s,h_t} that represents the activation signal for the channel c_{h_s,h_t}.
   (f) Add to the neuron h_s the firing rules of the form:

$$a_{h_s,h_t} v^+ / v a_{h_s,h_t} \rightarrow v^{w_{h_s,h_t}}(c_{h_s,h_t}) s(c_{i_k})$$

This rule means that if an activation signal is received and there are viruses in the host, it consumes a virus and the activation signal, and it sends w_{h_s,h_t} virus spikes to neuron h_t and a spike s to the instruction that triggered the transmission channel.

$$a_{h_s,h_t}/a_{h_s,h_t} \rightarrow f(c_{i_k})$$

This rule means that if an activation spike is received and there are no viruses in the host, it consumes the activation signal spike and it sends a spike f to the instruction that triggered the transmission channel.

5. For each instruction i_k:
   (a) If the output degree of i_k is equal to 2 then there are two different instructions $i_{k_1}$and i_{k_2} such that $(i_k, i_{k_1}) \in E_I$ and $(i_k, i_{k_2}) \in E_I$.
   If the instruction i_k is connected to a channel (h_s, h_t):
   - Create a rule $a_i/a_i \rightarrow a_{h_s,h_t}(c_{i_k})$, which upon receiving an activation spike instruction a_i, consumes it and sends an activation spike to the channel.
   - Create two channels $c_{i_k,i_{k_1}}$ and $c_{i_k,i_{k_2}}$
   - If $w_I(i_{k_1}) = w_I(i_{k_2})$ then create the following firing rules:

$$s/s \rightarrow a_i(c_{i_k,i_{k_1}})$$

$$f/f \rightarrow a_i(c_{i_k,i_{k_1}})$$

$$s/s \rightarrow a_i\left(c_{i_k,i_{k_2}}\right)$$

$$f/f \rightarrow a_i(c_{i_k,i_{k_2}})$$

According to the previous rules, the following instruction is non-deterministically selected.
Otherwise, we take

$$i_{max} = \text{argmax}_{i\in\{i_{k_1},i_{k_2}\}} w_I((i_k, i))$$

$$i_{min} = \text{argmin}_{i\in\{i_{k_1},i_{k_2}\}} w_I((i_k, i)).$$

The following two firing rules are created:

$$s/s \rightarrow a_i(c_{i_k,i_{max}})$$

$$f/f \rightarrow a_i(c_{i_k,i_{min}})$$

So, if there was a virus in the host, the next instruction is selected according to the maximal weight channel and, otherwise, it is selected according to the minimal weight channel.
If the instruction i_k is not connected to any channel:
- Create two channels $c_{i_k,i_{k_1}}$ and $c_{i_k,i_{k_2}}$

– Create the following firing rules:

$$a_i/a_i \to a_i(c_{i_k,i_{k_1}})$$
$$a_i/a_i \to a_i(c_{i_k,i_{k_2}})$$

So, the next instruction is non-deterministically selected.

(b) If the output degree of i_k equals to one, and $(i_k, i_{k'}) \in E_I$, a rule of the form $a_i/a_i \to a_i(c_{i_k,i_{k'}})$ is created.

(c) If the output degree of i_k is equal to 0, a forgetting rule $a_i \to \lambda$ is created.

In the proposed construction, the structure of the graphs in the virus machine is virtually simulated through the different communication channels. Furthermore, the dynamics of virus transmission, the activation of control instructions, and the opening of transmission channels based on those instructions are modeled by the different firing rules that we have defined.

We can formally prove that the complete simulation of a virus machine computation can be carried out on the SNP-MC-CS system.

□

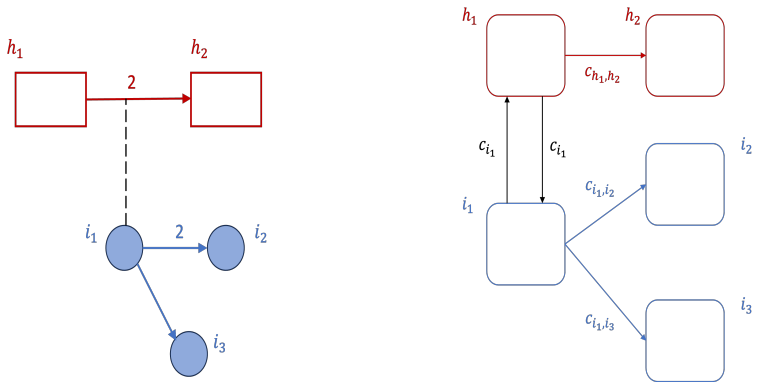

Fig. 8. A virus machine (left) and its equivalent SNP-MC-CS system (right).

Example 1. In the figure 8, we can see a very simple virus machine with two host nodes, and three instruction nodes. In the same figure, we can see the equivalent SNP-MC-CS. The rules in every neuron are the following:

– Neuron h_1
  - $a_{h_1,h_2}V^+/v\ a_{h_1,h_2} \to v^2(c_{h_1,h_2}), s(c_{i_1})$
  - $a_{h_1,h_2} \to f(c_{i_1})$

– Neuron i_1
  - $a_i \to a_{h_1,h_2}(c_{i_1})$
  - $s \to a_i(c_{i_1,i_2})$
  - $f \to a_i(c_{i_1,i_3})$

4.2 Neural-like P systems with plasmids

Neural-like P systems with plasmids were proposed in [4] inspired by the bacterial conjugation of genetic plasmids. In that work, the plasmids are objects of the system. This is a different approach to the recently computing by plasmids model [33], where the plasmids are finite sets of rules that can be moved in the region space of a P system. The use of multiple channels within neural-like P systems with plasmids has also been proposed [13].

In this section, we propose a simulation of neural-like P systems with plasmids by SNP-MC-CS systems. First we provide a basic definition of the model as follows.

Definition 7.[4] A neural-like P systems with plasmids (NP P system, for short) of degree $m \geq 1$, is a construct of the form $\Pi = (O, b_1, \ldots, b_m, link, in, out)$ where:

1. $O = \{p_1, \ldots, p_w\}$ is an alphabet of *plasmids*
2. $b_1, \ldots, b_m$ are bacteria of the form $b_i = (N_i, R_i)$ for $1 \leq i \leq m$ where $N_i = \langle n_1, n_2, \ldots, n_w \rangle$ is a vector and $n_j \geq 0$ is the initial number of plasmids p_j inside bacterium b_i for $1 \leq j \leq w$; and R_i is a finite non-empty set of plasmid rules of the form $C/r_1, r_2, \ldots, r_n$ where condition C is a multiset over O, and, each r_k for $1 \leq k \leq n$ is either one of two forms:
   (a) A *transmit* operation: $P \rightarrow out$, where P is a submultiset of C;
   (b) A *kill* operation: $P \rightarrow \lambda$, where P is a submultiset of C;
3. $link \subset \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin link$ for $1 \leq i \leq m$ (the links between bacteria);
4. in, out indicates the input and output bacterium of the system, respectively. They can be omitted, depending on whether the system is generating outputs or accepting inputs.

□

We explain how a NP P system works. At every time, the system executes a rule in each bacterium (whenever possible), this rule will be chosen in a non-deterministic way among the applicable rules. A rule $C/r_1, r_2, \ldots, r_n$ in bacteria b_i is applicable if at least each plasmid in C is in b_i. When a rule is executed, all its operations are executed sequentially starting from r_1 up to r_n. The transmit operations ($P \rightarrow out$) send a copy of every plasmid in P to every linked bacterium b_j, that is $(i, j) \in link$. On the other hand, the kill operations ($P \rightarrow \lambda$) remove all the plasmids defined in P. Observe that the transmit operations do not remove any plasmid in the bacteria. The computation halts whenever no bacteria can apply a rule.

It has been proved that NP P systems are a universal model of computation given that they can simulate the register machines [4].

The NP P systems with multiple channels (NPMC P systems) [13] have been proved to be universal, and they can achieve universality by using less computational resources than NP P systems (the number of plasmids and rules decreases).

In the following, we will prove the equivalence between NP P systems and SNP-MC-CS systems. The proof can be easily adapted to NPMC P systems.

Theorem 3. For every NP P system there exists an equivalent SNP-MC-CS system.

Proof. Let $\Pi = (O, b_1, \ldots, b_m, link, in, out)$ be an NP P system. We can apply the following steps to obtain an equivalent SNP-MC-CS system:

1. Create three spikes a_p, s_p and h_p.
2. Create three channels c_h, c_p and c_c.
3. Create the neuron n_{halt} with the rules $h_p^m \to h_p(c_h)$ and $h_p \to \lambda$.
4. For each $p_i \in O$ define a spike p_i.
5. For each bacterium $b_i = (N_i, R_i)$ $(1 \leq i \leq m)$:
   (a) Create three neurons: b_i, c_{b_i} and g_{b_i}. Initially, every neuron has a spike a_p. The neuron c_{b_i} also has a spike s_p.
   (b) Add the synapses (b_i, c_{b_i}), (c_{b_i}, b_i), (c_{b_i}, g_{b_i}) and (g_{b_i}, c_{b_i}) connected by the channel c_c.
   (c) Add the synapse (c_{b_i}, n_{halt}) connected by the channel c_h.
   (d) For all $n_j \in N_i$ add n_j spikes of type p_j to the neuron b_i.
   (e) Add the rule $a_p \to a_p(c_c)$ to the neuron g_{b_i}.
   (f) Add the rules $a_p s_p \to a_p(c_c)$, and $a_p \to a_p(c_c)h_p(c_h)$ to the neuron c_{b_i}.
   (g) For every rule of the form $C/r_1, r_2, \ldots, r_n \in R_i$ define a rule such that:
      i. The regular expression is defined as $c_1 c_2 \ldots c_n p_1^* p_2^* \ldots p_w^*$ where $c_1 c_2 \ldots c_n$ is the multiset C.
      ii. The spikes that consume the rule is the sum of all the multisets of plasmids consumed by the kill operations plus the spike a_p.
      iii. The right-hand side of the rule is defined by the multiset resulting from adding all the multisets of the transmit operations by the channel c_p, and the spike s_p by the channel c_c.
   (h) Add the rule $a_p \to \lambda$ to the neuron b_i.
6. For each $(i, j) \in link$ add a synapse (b_i, b_j) connected by the channel c_p.
7. The input neuron is defined by in.
8. Given the output bacteria out, add the synapses (n_{halt}, b_{out}) and (b_{out}, out) connected by the channel c_h.
9. Add to the neuron out the rules $h_p p_1^* p_2^* \ldots p_w^* / p_i \to p_i(c_h)$ for all $1 \leq i \leq w$ and $h_p \to \lambda$

The system works as follows: As only one rule can be applied to each bacterium at a time, there is a single spike a_p in the neuron representing the bacterium, and all the rules consume it; therefore, once one rule is applied, the others can no longer be applied.

To maintain a constant number of a_p spikes per bacterium, each bacterium has a control neuron c_{b_i} and a generation neuron g_{b_i} which, at each execution step, send a spike a_p to the bacterium b_i. To prevent a_p spikes from accumulating in the bacterium if it does not execute a rule, a forgetting rule $a_p \to \lambda$ is added. However, this presents a new problem: when the NP P system reaches a halting

state (no bacterium is applying any rules), the system as a whole will not reach a halting state, because the generators continue to apply rules.

To solve this, we have the spikes s_p, h_p and the neuron n_{halt}. All control neurons are connected to the neuron n_{halt}; when a rule is applied to a bacterium, a spike s_p is sent to its control neuron. Therefore, if that neuron does not receive a spike s_p at a specific point in time, this means that its bacterium has not applied any rules, so it sends a spike h_p to the neuron n_{halt}.

If, at a given moment, the neuron n_{halt} receives a number of h_p spikes smaller than the number of bacteria in the system, this means that not all of them have stopped, so it applies a forgetting rule and everything continues to function normally. However, if, on the other hand, the number of spikes h_p received is equal to the number of bacteria in the system, this means that all of them have stopped; consequently, it sends a spike h_p to all the control neurons, which blocks them and causes them to stop generating spikes, thereby reaching the halting state.

We can formally prove that the complete simulation of a NP P system with plasmids computation can be carried out by the SNP-MC-CS system.

□

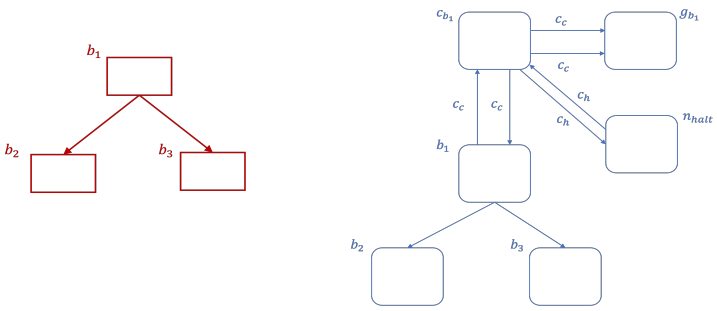

Fig. 9. A neural-like P system with plasmids (left) and its equivalent SNP-MC-CS system (right).

Example 2. In the figure 9, we can see a very simple neural-like P system with three bacteria. In the same figure, we can see the equivalent SNP-MC-CS.

The rules in the bacteria b_1 are the following:

- $p_1p_2/p_2 \to out,\ p_1 \to \lambda$
- $p_1p_3/p_3 \to out,\ p_1 \to \lambda$

The rules in the main neurons of the SNP-MC-CS system are the followings:

- Neuron c_{b_1} with the spikes $a_p s_p$

- $a_p s_p \to a_p(c_c)$
- $a_p \to a_p(c_c), h_p(c_h)$

– Neuron g_{b_1} with the spike a_p

- $a_p \to a_p(c_c)$

– Neuron b_1 with the spike a_p

- $p_1 p_2 p_1^* p_2^* p_3^* / p_1 a_p \to p_2(c_p), s_p(c_c)$
- $p_1 p_3 p_1^* p_2^* p_3^* / p_1 a_p \to p_3(c_p), s_p(c_c)$
- $a_p \to \lambda$

– Neuron n_{halt}

- $h_p^3 \to h_p(c_h)$
- $h_p \to \lambda$

5 Conclusions

In this work we have presented a new model that combines two previously proposed features of Spiking Neural P systems: the use of multiple channels in the rules and in the network topology, and the use of a multi-spike alphabet. Initially, we demonstrated the completeness of the new model through simulations of register machines. This is not a novel aspect, since each variant used separately already exhibited this universality property. However, in our proposal, we present new simulations that incorporate both characteristics of the model.

Subsequently, we have demonstrated that the proposed model is capable of directly simulating other computational models of interest in the field of membrane computing: virus machines and neural-like P systems with plasmids. In both cases, we have proposed a direct construction of the model from the definitions of VM and NP P systems.

In section 3.1, we have provided different schemes for simulating register machine programs. However, the simulation is fundamentally sequential. In future works, we will explore the ability of this model to simulate other intrinsically parallel models, such as parallel random access machines (PRAM) [7].

Furthermore, as we mentioned in the introduction to this work, our proposal can be formulated as a general-purpose computing environment in which different neural networks must coexist, cooperate and influence one another, and in which neurons can specialise in different types of information. This general framework will be explored in future works.

Acknowledgments. This work has received funding from the Generalitat Valenciana within the Prometeo program in the project *"A disruptive way to ameliorate the diagnosis and treatment of sensorineural diseases."* CIPROM/2023/026.

References

1. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik (2006) Extended Spiking Neural P Systems. In: H.J. Hoogeboom, G. Păun, G. Rozenberg, A. Salomaa (eds). Membrane Computing. WMC 2006. LNCS vol, 4361. pp 123-134 Springer.
2. F.G.C. Cabarle (2024) Thinking about spiking neural P systems: some theories, tools, and research topics *Journal of Membrane Computing* 6 pp 148-167.
3. F. G. C. Cabarle, H. N. Adorna, M. J. Pérez-Jiménez, T. Song (2015) Spiking neural P systems with structural plasticity. *Neural Computing and Applications* 26 pp 1905–1917.
4. F.G.C. Cabarle, X. Zeng, N. Murphy, T. Song, A. Rodríguez-Patón, X. Liu. (2021) Neural-like P systems with plasmids. *Information and Computation*, Vol. 281, 194776.
5. X. Chen, M.J. Pérez-Jiménez, L. Valencia-Cabrera, B. Wang, X. Zeng (2016) Computing with viruses. *Theoretical Computer Science* 623 pp 146-159.
6. S. Fan, P Paul, T. Wu, H. Rong, G. Zhang (2020) On Applications of Spiking Neural P systems. *Applied Sciences* 10(20) 7011.
7. S. Fortune, J. Wyllie (1978) Parallelism in random access machines In Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, pp. 114-118
8. S. Ghosh-Dastidar, H. Adeli (2009) Spiking Neural Networks. *International Journal of Neural Systems* 19(4) pp 295-308.
9. T.O. Ishdorj, A. Leporati, L. Pan, J. Wang (2009) Solving NP-Complete Problems by Spiking Neural P Systems with Budding Rules. Proceedings of the *10th International Workshop on Membrane Computing*, LNCS Vol. 5957 pp 335–353. Springer.
10. M. Ionescu, Gh. Păun, T. Yokomori (2006) Spiking neural P systems. *Fundamenta Informaticae*, 71 pp 279-308
11. L. Kari, G. Rozenberg (2008) The Many Facets of Natural Computing. *Communications of the ACM*, 51(10) pp 72-83.
12. A. Leporati, G. Mauri, C. Zandron (2022) Spiking neural P systems: main ideas and results. *Natural Computing* Vol. 21, pp 629-649.
13. Y. Li, B. Song , X. Zeng (2023) Neural-Like P Systems With Plasmids and Multiple Channels. *IEEE Transactions on Nanobioscience*, Vol. 22, No, 2 pp 420-429
14. Z. Lv, Q.Yang, H. Peng, X. Song and J. Wang Z. (2021) Computational power of sequential spiking neural P systems with multiple channels. *Journal of Membrane Computing* Vol. 3, Iss. 4 pp 270–283
15. C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón (2003) Tissue P systems. *Theoretical Computer Science* 296(2) pp 295-326.
16. Q. Miao, Y. Shen, W. Zang, Y. Zhao (2025) Multiple channels spiking neural P systems with lateral inhibition, rules dynamic generation and removal. *Theoretical Computer Science* Vol. 1042, 115236.
17. M.L. Minsky (1967) *Computation: Finite and Infinite Machines* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
18. X. Ning. G. Yang, Z. Sun, X. Song (2024) On the Universality of Spiking Neural P Systems With Multiple Channels and Autapses *IEEE Access* Vol. 12, pp 8773-8779.
19. L. Pan, Gh. Păun, M.J. Pérez-Jiménez (2011) Spiking neural P systems with neuron division and budding. *Science China Information Sciences* 54 pp 1596–1607.
20. L. Pan, J. Wang, H.J. Hoogeboom (2012) Spiking Neural P Systems with Astrocytes. *Neural Computation* 24 (2012) pp 805-825.
21. L. Pan, X. Zeng, X. Zhang (2011) Time-free Spiking Neural P Systems. *Neural Computation* 23(5) pp 1320–1342.

22. P. Paul, P. Sosík (2024) Solving the SAT problem using spiking neural P systems with coloured spikes and division rules *Journal of Membrane Computing* Vol. 6, Issue 3, pp 222-233.
23. Gh. Păun (2000) Computing with Membranes. *Journal of Computer and Systems Sciences* 61, pp 108-143.
24. Gh. Păun (2002) *Membrane Computing. An Introduction.* Springer.
25. Gh. Păun, Gr., Rozenberg, A. Salomaa, A. (eds.) (2010) The Oxford Handbook of Membrane Computing. Oxford University Press, NY
26. H. Peng, J. Wang (2024) Advanced Spiking Neural P Systems. Models and Applications. Springer.
27. H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Lou. (2017) Spiking neural P systems with multiple channels. *Neural Networks,* Vol. 95, pp 66-71.
28. H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang (2013) Fuzzy reasoning spiking neural P system for fault diagnosis. *Information Sciences* 235 pp 106–116.
29. A. Ramírez de Arellano, D. Orellana Martín M.J. Pérez Jiménez (2023) Generating, computing and recognizing with virus machines. *Theoretical Computer Science* Vol. 972 114077.
30. A. Ramírez de Arellano, F.G.C. Cabarle, D. Orellana Martín M.J. Pérez Jiménez (2026) On the normal forms and computational power of virus machines *Theoretical Computer Science* Vol. 1083 116139.
31. Á. Romero-Jiménez, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez (2015) Computing partial recursive functions by virus machines Proceedings of the 16th International Conference on Membrane Computing (CMC16). Gr. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (eds.). LNCS Vol. 9504 pp. 353-384 Springer.
32. G. Rozenberg, A. Salomaa, Eds. (1997) *Handbook of Formal Languages.* Vols. I, II and III. Springer-Verlag.
33. J.M. Sempere (2026): Computing by plasmids. *Journal of Membrane Computing*, Online first.
34. T. Song, A. Rodríguez-Patón, P. Zheng, X. Zeng (2018) Spiking Neural P Systems With Colored Spikes. *IEEE Transactions on Cognitive and Developmental Systems* 10(4) pp 1106-1115.
35. X. Song, L. Valencia-Cabrera, H. Peng, J. Wang (2021) Spiking neural P systems with autapses, *Information Sciences*, Vol. 570, pp 383-402,.
36. Z. Sun, Q. Wang, Z. Wei (2019) Fault location of distribution network with distributed generations using electrical synaptic transmission-based spiking neural P systems. *International Journal of Parallel, Emergent and Distributed Systems* 36(1) pp 11-27.
37. L Valencia-Cabrera, MJ Pérez-Jiménez, X Chen, B Wang, X Zeng (2015) Basic Virus Machines. In Proceedings of the 16th International Conference on Membrane Computing (CMC16), José M. Sempere and Claudio Zandron (eds.) pp 323-342.
38. S. Verlan, R. Freund, A. Alhazov, S. Ivanov, L. Pan (2020) A formal framework for spiking neural P systems. *Journal of Membrane Computing* Vol. 2, pp 355–368.
39. G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez (2014) An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems* 24(5) 1440006.
40. G. Zhang, S. Verlan, T. Wu, F.G.C. Cabarle, J. Xue. D. Orellana-Martín, J. Dong, L. Valencia-Cabrera, M.J. Pérez-Jiménez (2024) Spiking Neural P Systems. Theory, Applications and Implementations. Springer.