Dendrite P systems☆

Hong Peng [a], Tingting Bao [a], Xiaohui Luo [a], Jun Wang [b,*], Xiaoxiao Song [b], Agustín Riscos-Núñez [c], Mario J. Pérez-Jiménez [c]

[a] School of Computer and Software Engineering, Xihua University, Chengdu 610039, China
[b] School of Electrical Engineering and Electronic Information, Xihua University, Chengdu 610039, China
[c] Research Group of Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Sevilla 41012, Spain

Keywords:
P systems
Neural-like P systems
Dendrite P systems
Computational power

A B S T R A C T

It was recently found that dendrites are not just a passive channel. They can perform mixed computation of analog and digital signals, and therefore can be abstracted as information processors. Moreover, dendrites possess a feedback mechanism. Motivated by these computational and feedback characteristics, this article proposes a new variant of neural-like P systems, dendrite P (DeP) systems, where neurons simulate the computational function of dendrites and perform a firing–storing process instead of the storing–firing process in spiking neural P (SNP) systems. Moreover, the behavior of the neurons is characterized by dendrite rules that are abstracted by two characteristics of dendrites. Different from the usual firing rules in SNP systems, the firing of a dendrite rule is controlled by the states of the corresponding source neurons. Therefore, DeP systems can provide a collaborative control capability for neurons. We discuss the computational power of DeP systems. In particular, it is proven that DeP systems are Turing-universal number generating/accepting devices. Moreover, we construct a small universal DeP system consisting of 115 neurons for computing functions.

1. Introduction

Neural-like P systems are one of the main types of membrane computing models (Bernardini & Gheorghe, 2004; Cienciálová, Csuhaj-Varjú, Kelemenová, & Vaszil, 2009; Freund, Păun, & Pérez-Jiménez, 2005; Gheorghe et al., 2013; Martínez-Del-Amor, Macías-Ramos, Valencia-Cabrera, & Pérez-Jiménez, 2016; Pavel & Buiu, 2012; Peng, Shi, Wang, Riscos-Núñez, & Pérez-Jiménez, 2017; Peng, Wang, Pérez-Jiménez, & Riscos-Núñez, 2015; Peng, Wang, Shi, Pérez-Jiménez, & Riscos-Núñez, 2016; Păun, 2000; Păun & Pérez-Jiménez, 2010; Păun & Păun, 2006; Song, Zhang, & Pan, 2016; Wang, Shi, & Peng, 2016; Xue et al., 2018; Zhang, Gheorghe, Pan, & Pérez-Jiménez, 2014; Zhang, Wu, Păun, & Pan, 2016; Zhao, Liu, & Qu, 2012). They generally refer to a class of distributed and parallel computing models, motivated by the mechanisms of biological neurons and nervous systems. Topologically, neural-like P systems are expressed by a directed graph or a more complex network. They address three main topics: (i) models and computational theoretical problems (universality, effectiveness, and complexity); (ii) application of models; and (iii) implementation of models. Much effort has been made in the realm, as found in the Handbook of Membrane Computing (Păun, Rozenberg, & Salomaa, 2010).

Spiking neural P (SNP) systems (Ionescu, Păun, & Yokomori, 2006) are a widely studied type of neural-like P system, inspired by the way that neurons work with and exchange information by sending spikes along synapses. An SNP system is a distributed and parallel computing model. Except a directed graph, SNP systems have two components: data and rules. Data can usually be denoted by the number of spikes in each neuron, and the collection of the states of all neurons, known as a configuration, describes the state of the entire system. The behavior of the system is controlled by firing rules and forgetting rules. Firing rules are expressed by the form $E/a^c \rightarrow a^p$, where E is a regular expression. To distinguish the dendrite rules presented below, $E/a^c \rightarrow a^p$ are called "usual firing rules" in this work, and their semantics can be illustrated as follows. Suppose that a neuron σ has a firing rule $E/a^c \rightarrow a^p$ and n spikes. If $a^n \in L(E)$ and $n \geq c$, then the neuron can fire. When the neuron fires, it consumes c spikes ($n - c$ spikes are retained) and generates p spikes. The generated p spikes will be sent to its succeeding neurons. If $p = 0$, then the rules can be written in the form $a^c \rightarrow \lambda$. A forgetting rule indicates that c spikes are consumed, but no spike is generated. From the perspective of working mechanism, the action of a neuron contains the two steps of integration and excitation. In

☆ This work was partially supported by Research Fund of Sichuan Science and Technology Project China (No. 2018JY0083), and Research Foundation of the Education Department of Sichuan province China (No. 17TD0034), China.

* Corresponding author.
E-mail addresses: ph.xhu@hotmail.com (H. Peng), wj.xhu@hotmail.com (J. Wang).

the integration step, a neuron receives, accumulates, and stores the spikes that its prepositive (source) neurons send out. The excitation step accomplishes a neuron's firing procedure, as described above. Therefore, the working mechanism of neurons can be simply summarized as a storing–firing process: the spikes are first stored and then are excited by the firing rules.

Many variants have appeared since SNP systems were first proposed. Abstracted from the inhibitory and excitatory influence of astrocytes on synapses, Păun (2007) discussed SNP systems with astrocytes. Pan, Wang, and Hoogeboom (2012) presented SNP systems with anti-spikes inspired by inhibitory impulses/spikes. Peng, Yang, et al. (2017) and Song et al. (2018) discussed SNP systems with multiple channels. SNP systems with polarizations were discussed in Wu, Păun, Zhang, and Pan (2018). The structural dynamism of biological synapses inspired Cabarle, Adorna, Jiang, and Zeng (2016) to propose an SNP system with scheduled synapses. Considering a new communication strategy among neurons, Pan, Păun, Zhang, and Neri (2017) investigated SNP systems with communication on request. Wang, Hoogeboom, Pan, Păun, and Pérez-Jiménez (2010) discussed an SNP system with weights, while Zeng, Zhang, Song, and Pan (2014) presented SNP systems with thresholds. Abstracted from Eckhorn's neuron model and intersecting cortical model (ICM), coupled neural P systems and dynamic threshold neural P systems were investigated by Peng and Wang (2018), Peng, Wang, Pérez-Jiménez, and Riscos-Núñez (2019), respectively. Moreover, spiking neural P systems with inhibitory rules and nonlinear spiking neural P systems were discussed in Peng, Li, et al. (2020), Peng, Lv, et al. (2020). Ionescu, Păun, Pérez-Jiménez, and Yokomori (2011) discussed spiking neural dP systems (with request rules). With the limitation that at each time at most one neuron works, sequential SNP systems were discussed in Ibarra, Păun, and Rodríguez-Patón (2009) and Zhang, Zeng, Luo, and Pan (2014). A global clock is usually assumed in the above SNP systems, in which case they are synchronized. Cavaliere et al. (2009) and Song, Pan, and Păun (2012) discussed a number of asynchronous SNP systems. Several SNP systems using fuzzy logic were investigated, for example, weighted fuzzy SNP systems (Wang, Shi, Peng, Pérez-Jiménez, & Wang, 2013) and fuzzy reasoning SNP systems (Peng et al., 2013). Computational power (as function computing devices and number or language generating devices) has been discussed. As number generating/accepting devices (Cabarle et al., 2016; Cavaliere et al., 2009; Ionescu et al., 2006; Pan et al., 2017; Peng, Yang, et al., 2017; Păun, 2007; Song et al., 2012), language generators (Chen, Freund, Ionescu, Păun, & Pérez-Jiménez, 2007; Zhang, Zeng, & Pan, 2008), and function computing devices (Păun & Păun, 2007; Păun & Sidoroff, 2012; Wu et al., 2018), it was proven that many variants of SNP systems are Turing universal. Some efforts have been made to apply SNP systems to practical problems, such as image processing (Díaz-Pernil, Gutiérrez-Naranjo, & Peng, 2019; Díaz-Pernil, Peña-Cantillana, & Gutiérrez-Naranjo, 2013), fault diagnosis (Peng, Wang, Ming, et al., 2018; Peng, Wang, Shi, Pérez-Jiménez, & Riscos-Núñez, 2017; Wang et al., 2015), and combinatorial optimization problems (Zhang, Rong, Neri, & Pérez-Jiménez, 2014), and a number of other SNP systems were implemented (Carandang, Villaflores, Cabarle, Adorna, & Martínez-Del-Amor, 2017; Macías-Ramos, Pérez-Jiménez, Song, & Pan, 2015).

Spiking neural P systems with rules on synapses (Song, Pan, , & Păun, 2014) are an interesting form of neural-like P systems where the firing and forgetting rules are moved from neurons to synapses. Thus synapses are regarded as information processing units, while neurons are degenerated into storage units. In addition, Peng et al. (2018) and Song and Pan (2015) discussed two distinct spike consumption strategies, and computational power of the variants as function computing devices and number generating/accepting devices has been discussed.

Chen, Ishdorj, and Păun (2007) discussed another form of neural-like P systems, axon P systems, where nodes are placed in a linear way and each sends spikes only to two neighbors. An axon P system is executed concurrently and can nondeterministically process information in nodes. The universality results of axon P systems as function computing, number generating and language generating devices, have been discussed in Zhang, Pan, and Păun (2015), Zhang, Wang, and Pan (2009), respectively.

A nerve cell consists of dendrites, somata, axons, and synapses. Each has one or more dendrites, which receive the stimulus and transmit excitement to somata, but only one axon that transmits excitement between somata. Therefore, we generally believe that dendrites simply transmit to somata the electrical impulses received from the synapses. Thus, as in SNP systems or spiking neural networks, neurons are viewed as information processors, while dendrites (and axons) are regarded as transmission channels. However, the conclusion that the dendrites only passively transmit the current to somata has not been confirmed by experiments. Moore et al. (2017) recently indicated that the dendrites are not just a passive channel. Their research shows that: (i) dendrites have an electrical activity, and the generated spike is 10 times one produced by soma; and (ii) dendrites perform a mixed computation of analog and digital signals. In our work, dendrites are abstracted as information processors, while neurons (somata) degenerate into storage units.

Some new biological features of dendrites, such as dendritic feedback and delay, have been proven experimentally (London & Hausser, 2005). It has usually been thought that information in the nervous system travels in one direction, from dendrites to somata, and then to axons. However, for many types of neurons, the presence of excitable ionic currents in dendrites supports dendritic action potentials that are transmitted in the opposite direction, from somata to dendrites (Stuart, Spruston, Sakmann, & Hausser, 1997). This backpropagation implies that the neuron possesses an internal feedback mechanism, and the interaction between dendritic responses and somatic spikes can be abstracted computationally. In this work, the dendritic feedback mechanism is abstracted to develop a new kind of firing rule, called a dendrite rule, of the form $(E_1, E_2, \ldots, E_s)/a^p \leftarrow (a^{c_1}, a^{c_2}, \ldots, a^{c_s})$, whose firing condition is controlled by its prepositive (source) neurons. The control idea potentially reflects the feedback mechanism of the dendrites.

We propose a new model of neural-like P systems, dendrite P (DeP) systems, which consist of several neurons in a directed graph to form a distributed and parallel computing system. DeP systems and SNP systems (and SNP systems with rules on synapses) differ in three aspects:

(1) The neurons in DeP systems simulate the information processing mechanism of dendrites instead of that of original neurons or synapses. However, original neurons are used only as storage units in DeP systems. Thus the working mechanism of neurons in DeP systems is a "firing-storing process", i.e., the spikes in prepositive neurons of a neuron can be handled by its dendrite rule, and then the generated spikes are stored in the neuron. Conversely, as mentioned above, SNP systems (and SNP systems with rules on synapses) adopt a "storing–firing process".
(2) In DeP systems, dendrite rules potentially reflect the feedback mechanism of dendrites. Therefore, the firing condition of each dendrite rule only depends on states of prepositive neurons of the neuron where the rule resides, and it is irrelevant to the state of the neuron itself. However, in SNP systems (and SNP systems with rules on synapses), the firing condition of the usual firing rule depends only on the state of the neuron where the rule resides, and is unrelated to the states of other neurons.

(3) Since a dendrite rule has several regular expressions, the firing of the neuron where the rule resides is controlled by multiple prepositive neurons. Therefore, dendrite rules provide a collaborative control capability for DeP systems, i.e., the collaborative firing mechanism of several prepositive neurons. However, SNP systems and those with rules on synapses lack the collaborative mechanism.

Intuitively, DeP systems provide a stronger control condition and processing ability and are potentially suitable for solving some real-life problems. Turing-universality of DeP systems as function computing and number generating/accepting devices is discussed.

The proposed DeP systems are defined in Section 2, and an illustrative example and a comparative example are provided. Universality of DeP systems as number generating/accepting and function computing devices is proven in Section 3. Conclusions and discussion are drawn in Section 4.

2. Dendrite P systems

In this section, DeP systems are introduced, and then illustrative and comparative examples are provided. The notations and terms similar to those used in SNP systems will be adopted for clarity and ease of understanding.

2.1. Definition

Definition 1. A dendrite P (DeP) system with degree $m \geq 1$ is a construct,

$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$,

where:

(1) $O = \{a\}$ is the singleton alphabet (symbol a is known as the spike);
(2) $\sigma_1, \ldots, \sigma_m$ denote m neurons with 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 neuron σ_i;
   (b) R_i denotes the finite set of dendrite rules with the form
   $(E_1, E_2, \ldots, E_s)/a^p \leftarrow (a^{c_1}, a^{c_2}, \ldots, a^{c_s})$
   where E_j is a regular expression, $1 \leq j \leq s$, $p \geq 0$, and s indicates the number of prepositive (source) neurons of neuron σ_i;
(3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$ (synapses);
(4) in and out distinguish input and output neurons, respectively, of the system.

As mentioned above, each neuron in SNP systems has two components, data and rules, as shown in Fig. 1(a), where a^n denotes the data (or state) of a neuron and R_i is the set of firing rules. Since any neuron in an SNP system performs a "storing-firing process" (i.e., the received spikes are first integrated and stored, and then the neuron fires by some rule), it can also be understood in the form shown in Fig. 1(b). In contrast, in DeP systems, dendrites are viewed as information processors (characterized by dendrite rules) and neurons are degraded as storages, while synapses only reflect the connection relationships between neurons. Therefore, any neuron in a DeP system can be illustrated as in Fig. 1(c) and (d). In DeP systems, each neuron performs a "firing-storing process": the spikes in the corresponding prepositive neurons are handled first by dendrite rules, and then the generated spikes are stored in the neuron.

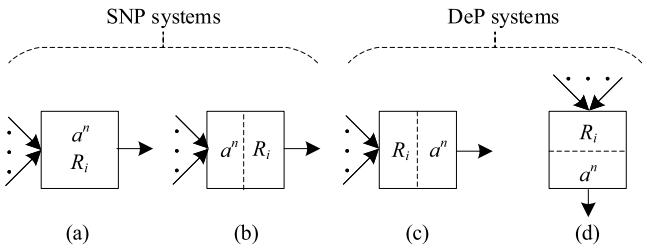

Fig. 1. Comparison of SNP and DeP systems. (a) and (b): neurons in SNP systems; (c) and (d): neurons in DeP systems.

From a topological perspective, a DeP system is expressed by a directed graph, where the nodes are the neurons and the edges correspond to the synapses. DeP systems differ significantly from SNP systems in the use of dendrite rules of the form $(E_1, E_2, \ldots, E_s)/a^p \leftarrow (a^{c_1}, a^{c_2}, \ldots, a^{c_s})$. Note that firing rules in SNP systems are of the form $E/a^c \rightarrow a^p$, called usual firing rules in this work. In usual firing rules, the left part of a firing rule indicates the content (consuming of spikes) in the neuron where the rule resides, while the right part corresponds to other (target) neurons. Since the neuron where a dendrite rule resides is its target, the dendrite rule uses the left arrow "$\leftarrow$" instead of the usual right arrow "$\rightarrow$". In dendrite rules, the right part of the arrow "$\leftarrow$" indicates the consuming of spikes in its source neurons. A neuron connects to several prepositive (source) neurons via its dendrite. Therefore, each dendrite rule has a group of regular expressions, $(E_1, E_2, \ldots, E_s)$. Thus dendrite rules can be written as: $(E_1 \wedge E_2 \wedge \cdots \wedge E_s)/a^p \leftarrow (a^{c_1}, a^{c_2}, \ldots, a^{c_s})$. Note that the firing condition is given in the form of its prepositive (source) neurons. The firing condition can be expressed in a conjunction form, $(a^{n_1} \in L(E_1)) \wedge (a^{n_2} \in L(E_2)) \wedge \cdots \wedge (a^{n_s} \in L(E_s))$, where n_j is the number of spikes in the ith prepositive neuron, $1 \leq j \leq s$. For neuron σ_i, if the firing condition for all of its prepositive (source) neurons is satisfied, then the dendrite rule can be applied, where c_j spikes in the jth prepositive (source) neuron are consumed $(1 \leq j \leq s)$ and p spikes are produced and stored in the neuron where the rule resides. In dendrite rules, if $c_i = 0$ (or $p = 0$), then a^{c_i} (or a^p) can be written as λ.

Note that in any neuron, two dendrite rules such as $(E_{11}, E_{21}, \ldots, E_{s1})/a^{p_1} \leftarrow (a^{c_{11}}, a^{c_{21}}, \ldots, a^{c_{s1}})$ and $(E_{12}, E_{22}, \ldots, E_{s2})/a^{p_2} \leftarrow (a^{c_{12}}, a^{c_{22}}, \ldots, a^{c_{s2}})$ may have that $(a^{n_1} \in L(E_{11})) \wedge (a^{n_2} \in L(E_{21})) \wedge \cdots \wedge (a^{n_s} \in L(E_{s1}))$ and $(a^{n_1} \in L(E_{12})) \wedge (a^{n_2} \in L(E_{22})) \wedge \cdots \wedge (a^{n_s} \in L(E_{s2}))$, i.e., the firing conditions of the two rules are true. In this case, we choose one of them nondeterministically. There is also the interesting case that dendrite rules in different neurons share one or more prepositive (source) neurons. For example, suppose that two rules, $(E_{1i}, E_{2i}, \ldots, E_{si})/a^{p_i} \leftarrow (a^{c_{1i}}, a^{c_{2i}}, \ldots, a^{c_{si}})$ and $(E_{1j}, E_{2j}, \ldots, E_{tj})/a^{p_j} \leftarrow (a^{c_{1j}}, a^{c_{2j}}, \ldots, a^{c_{tj}})$, from neurons σ_i and σ_j, respectively, can be applied simultaneously and share a prepositive (source) neuron σ_k. If the number of spikes n_k in neuron σ_k satisfies $\max\{c_{ki}, c_{kj}\} \leq n_k < c_{ki} + c_{kj}$, then this brings a conflict. In this case, one of the rules is nondeterministically chosen and applied.

As usual, neurons in the system work with each other in parallel, but the dendrite rules in each neuron are applied in sequential way. The configuration of the system at time t is characterized by the number of spikes stored in each neuron, i.e., $C_t = (n_1(t), n_2(t), \ldots, n_m(t))$, so the initial configuration is denoted by $C_0 = (n_1(0), n_2(0), \ldots, n_m(0)) = (n_1, n_2, \ldots, n_m)$. The transition between configurations can be defined using dendrite rules. A computation is any sequence of transitions starts from the initial configuration. The computation halts if a configuration is reached for which no rule can be applied. The output of a DeP system is a spike train exported by the output neuron. However,

each neuron in a DeP system only stores the received spikes and does not send out spikes in a forward direction. To facilitate the discussion of universality, we assume that the spike train in a DeP system is received by the output neuron instead of the environment. Under this assumption, a spike train is expressed by a binary sequence of 1 and 0: we write 1 if the output neuron receives a spike, and 0 if no spike is received. Thus the number of spikes received by the output neuron is the computation result. The set of numbers computed by Π is denoted by $N_{gen}(\Pi)$, where the subscript *gen* means that the system works in the generating mode. The family of all of the sets $N_{gen}(\Pi)$ generated by DeP systems containing at most m neurons and at most n rules in every neuron is denoted by $N_{gen}DeP_m^n$.

A DeP system can work in the accepting mode, where an input neuron is used to receive the spikes from the environment but the system has no output neuron. The system receives a spike train from the environment, and then it introduces the number n in a specified neuron in the form of $2n$ spikes. It is said that the number n is accepted by the system when the computation halts. Denote by $N_{acc}(\Pi)$ the set of numbers accepted by the system Π, where *acc* indicates that the system works in the accepting mode. Denote by $N_{acc}DeP_m^n$ the family of all of the sets $N_{acc}(\Pi)$ accepted by DeP systems containing at most m neurons and at most n rules in every neuron.

Remark 1. In DeP systems, the constraint $p \leq c_1 + c_2 + \cdots + c_s$ is relaxed for dendrite rules. Thus, for a dendrite rule, the number of generated spikes may be greater than the sum of the consumed spikes. The main reason for relaxing the constraint is to ensure that each module in the proof of universality can supply enough spikes. This is different from the usual firing rules in SNP systems, and is limited to the study of universality.

2.2. An illustrative example

Fig. 2 shows a simple dendrite P system consisting of four neurons. Initially, four spikes and two spikes are placed in neurons σ_1 and σ_2, respectively. Thus the initial configuration is $C_0 = (4, 2, 0, 0)$. Note that no rules can be applied in neuron σ_4. Since neuron σ_1 has four spikes and neuron σ_2 has two spikes, two rules, $(a^4, a^2)/a^2 \leftarrow (a^2, a^2)$ and $(a^4, a^2)/a \leftarrow (a^2, a)$, can be applied in neuron σ_3 at the same time, thus one is nondeterministically chosen and applied. There are two cases:

(1) If rule $(a^4, a^2)/a^2 \leftarrow (a^2, a^2)$ is chosen, then two spikes are consumed in neuron σ_1, two spikes are consumed in neuron σ_2, and two spikes are generated by the dendrite rule and stored in neuron σ_3. At this time, $C_1 = (2, 0, 2, 0)$. Since neuron σ_3 has two spikes and neuron σ_2 has none, rule $(\lambda, a^2)/a \leftarrow (\lambda, a)$ can be applied in neuron σ_4. Applying this rule, a spike in neuron σ_3 is removed and a spike is generated and stored in neuron σ_4. Therefore, $C_2 = (2, 0, 1, 1)$. Since no rules can be applied, the system halts.

(2) If rule $(a^4, a^2)/a \leftarrow (a^2, a)$ is chosen, then two spikes are consumed in neuron σ_1, a spike is consumed in neuron σ_2, and a spike is produced and stored in neuron σ_3. Hence $C_1 = (2, 1, 1, 0)$. Because there are two spikes in neuron σ_1 and a spike in neuron σ_2, rule $(a^2, a)/a \leftarrow (a^2, a)$ in neuron σ_3 can be applied. At the same time, a spike in neuron σ_2 and a spike in neuron σ_3 mean that rule $(a, a)/a \leftarrow (a, a)$ in neuron σ_4 can also be applied. As a result, a conflict arises because only one spike is in neuron σ_2. Therefore, one rule is chosen nondeterministically. There are two cases:

   (a) If rule $(a^2, a)/a \leftarrow (a^2, a)$ in neuron σ_3 is chosen, then two spikes are consumed in neuron σ_1, a spike is consumed in neuron σ_2, and a spike is produced and stored in neuron σ_3. Thus, neuron σ_3 contains three spikes. At this time, $C_2 = (0, 0, 3, 0)$ and the system halts.

   (b) If rule $(a, a)/a \leftarrow (a, a)$ in neuron σ_4 is chosen, then a spike is removed from neuron σ_2, a spike is removed from neuron σ_3, and a spike is produced and stored in neuron σ_4. Thus $C_2 = (2, 0, 0, 1)$ and the system halts.

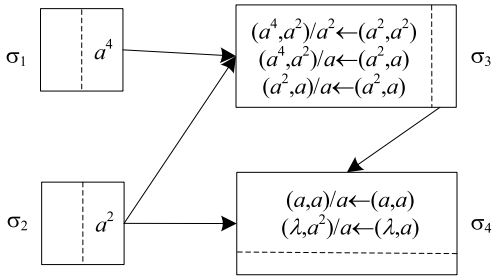

Fig. 2. A dendrite P system.

It can be seen from the example that there are two cases of choosing the rule nondeterministically: (i) as usual in SNP systems, if several rules can be applied in a neuron at the same time, then one is nondeterministically chosen; and (ii) in the conflict case, one of the conflict rules in different neurons is nondeterministically chosen.

2.3. Compared with other variants

To identify the differences between DeP systems and existing variants, an example is provided to compare DeP systems with SNP systems, SNP systems with multiple channels and SNP systems with rules on synapses. Figs. 3 and 4 show the example, where each system has three neurons, σ_1, σ_2 and σ_3.

We consider two cases.

Case 1: Assume that in each system, neurons σ_1 and σ_2 respectively have three and two initial spikes. For a DeP system, with three spikes in neuron σ_1 and two spikes in neuron σ_2, dendrite rule $(a^3, a^2)/a^2 \leftarrow (a^3, a^2)$ can be applied. Thus two spikes are produced and stored in neuron σ_3. For an SNP system with rules on synapses, with three spikes in neuron σ_1, rule $a^3/a^3 \rightarrow a^2$ is applied to send two spikes to neuron σ_3, and with two spikes in neuron σ_2, rule $a^2/a^2 \rightarrow a^2$ is applied to send two spikes to neuron σ_3. Hence, neuron σ_3 receives four spikes. For an SNP system, since neuron σ_1 contains three spikes and neuron σ_2 contains two spikes, rules $a^3/a^3 \rightarrow a^2$ and $a^2/a^2 \rightarrow a^2$ are applied to send two spikes to neuron σ_3. Hence, neuron σ_3 contains four spikes. For an SNP system with multiple channels, neurons σ_1 and σ_2 each send two spikes to neuron σ_3 via channel (1). Hence, neuron σ_3 has four spikes. (See Fig. 3).

Case 2: Assume that in each system, neurons σ_1 and σ_2 respectively have three and one initial spikes. For a DeP system, although neuron σ_1 has three spikes, neuron σ_2 has only one spike, hence dendrite rule $(a^3, a^2)/a^2 \leftarrow (a^3, a^2)$ cannot be applied. Note that if neuron σ_2 later has a chance to receive a spike, then the dendrite rule will be applied. This means that neuron σ_1 waits for neuron σ_2 to work together. This indicates an interesting collaborative firing mechanism of neurons in DeP systems.

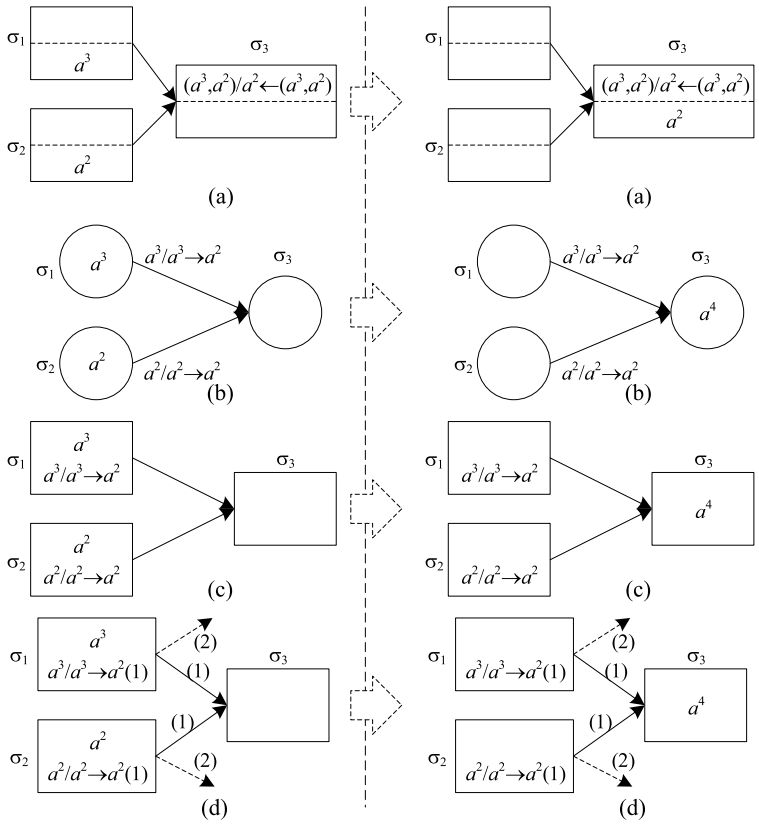

Fig. 3. Four variants in Case 1: (a) DeP system; (b) SNP system with rules on synapses; (c) SNP system; (d) SNP system with multiple channels. Left side is in initial configuration, and right side is in final configuration.

However, in each of the other three systems, with three spikes in neuron σ_1, its rule is applied to send two spikes to neuron σ_3, and since neuron σ_2 contains only one spike, it cannot fire. Therefore, neurons σ_1 and σ_2 cannot work collaboratively in SNP systems, SNP systems with multiple channels, and SNP systems with rules on synapses (see Fig. 4).

From the above two cases, we can distinguish the following three distinguishable characteristics in DeP systems.

(1) In DeP systems, the working mechanism of neurons can be described as a firing–storing process.
(2) In DeP systems, the firing condition of each dendrite rule only depends on states of source neurons of the neuron where the rule resides, and is irrelevant to the state of the neuron where the rule resides.
(3) DeP systems can provide a collaborative firing mechanism between several neurons.

3. Universality results

We now investigate computational power of DeP systems as number generating/accepting devices, and discuss a small universal DeP system for computing functions. By simulating register machines, DeP systems can generate/accept all recursively enumerable sets of numbers (characterized by *NRE*) and compute all of the admissible enumerable sets of the unary partial recursive functions.

A register machine can be denoted by $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, I is the set of instructions, l_0 is the start label, and l_h is the halting label. Each label in H corresponds to an instruction in I. There are three forms of instructions:

(1) l_i : $(ADD(r), l_j, l_k)$ (add 1 to register r and then nondeterministically go to one of the instructions with labels l_j, l_k).
(2) l_i : $(SUB(r), l_j, l_k)$ (if register r is nonzero, then decrease it by 1 and go to the instruction with label l_j; otherwise, go to the instruction with label l_k).
(3) l_h : *HALT* (halting instruction).

3.1. Turing universality of DeP systems as number generating devices

In the generating mode, a register machine can generate the number n in the following way. Initially, all registers are empty; the register machine starts from instruction l_0, and then it applies instructions continuously; when it reaches the halting instruction, the number stored in the first register is regarded as its computation result. As we know, register machines can characterize the family *NRE*.

Theorem 1. $N_{gen}DeP_*^2 = NRE$.

Proof. We must only prove that $NRE \subseteq N_{gen}DeP_*^2$, since the converse is obvious (Păun, 2002). For this purpose, we must simulate register machines in the generating mode based on the characterization of NRE. Without loss of generality, a register machine $M = (m, H, l_0, l_h, I)$ is given, and for a halting configuration, all registers are empty apart from register 1. Moreover, register 1 is never decremented during the computation. A DeP system Π_1 is designed to simulate the register machine, containing three kinds of modules: ADD module simulating ADD instruction, shown in Fig. 5; SUB module simulating SUB instruction, shown in Fig. 6; a FIN module exporting the computation result, shown in Fig. 7.

Suppose that each register r in M corresponds to a neuron σ_r in Π_1, and the number in register r is coded: if register r stores

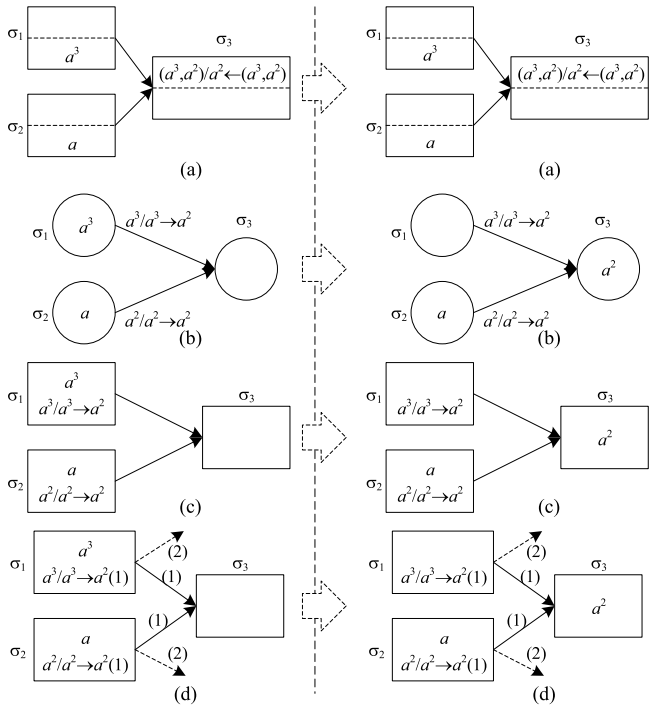

Fig. 4. Four variants in Case 2: (a) DeP system; (b) SNP system with rules on synapses; (c) SNP system; (d) SNP system with multiple channels. Left side is in initial configuration, and right side is in final configuration.

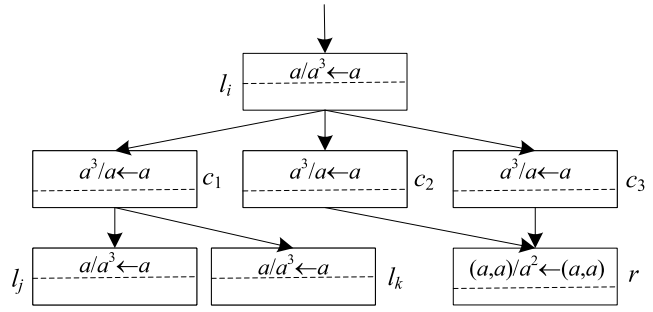

Fig. 5. Module ADD of Π_1, simulating the ADD instruction $l_i : (ADD(r), l_j, l_k)$.

the number $n \geq 0$, then neuron σ_r contains $2n$ spikes, vice versa. A neuron σ_l is associated with each instruction l in H and some auxiliary neurons are considered in the modules. Initially, each auxiliary neuron has no spike, and neuron σ_{l_0} receives a spike.

When neuron σ_{l_i} receives a spike, the system Π_1 starts to simulate the instruction $l_i : (OP(r), l_j, l_k)$ (OP is an ADD or SUB operation). Starting from the activated l_i, the simulation handles neuron σ_r as indicated by OP, and then one of neurons σ_{l_j} and σ_{l_k} receives a spike. The system accomplishes the simulation of register machine M once neuron σ_{l_h} receives a spike. In the simulation, output neuron σ_{out} applies its dendrite rules to receive a series of spikes, and the computation result is the number of the received spikes, i.e., the number in register 1 in M.

To explain that register machine M can be correctly simulated by system Π_1, how the ADD and SUB modules simulate the ADD and SUB instructions respectively and how the FIN module exports the computation result will be discussed in detail.

(1) Module ADD (shown in Fig. 5) — simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$.

To simulate the register machine, the system Π_1 starts from instruction l_0 (an ADD instruction). Suppose that an ADD instruction $l_i : (ADD(r), l_j, l_k)$ is simulated at time t. Neuron σ_{l_i}

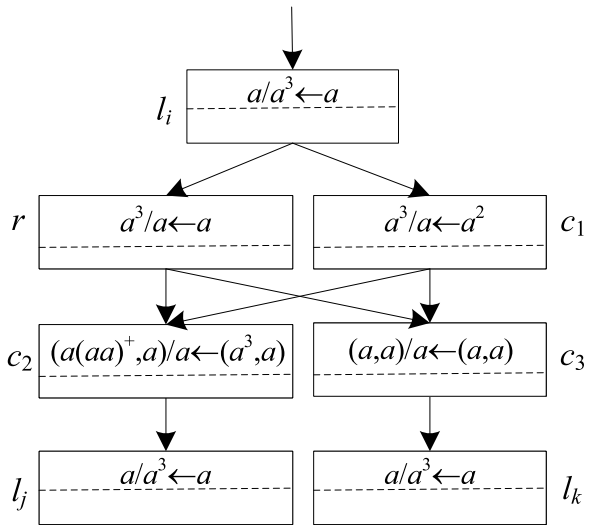

Fig. 6. Module SUB of Π_1, Π_2, and Π_3, simulating $l_i : (SUB(r), l_j, l_k)$.

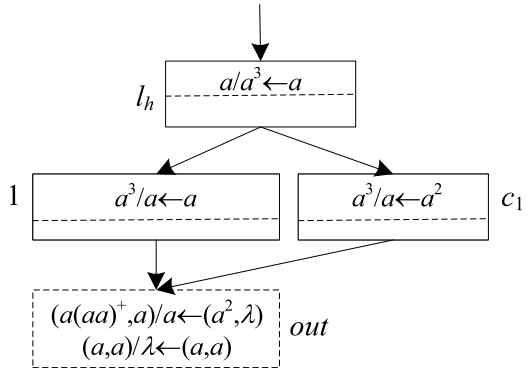

Fig. 7. Module FIN of Π_1, ending the computation.

applies rule $a/a^3 \leftarrow a$ to receive a spike, hence three spikes are generated. With three spikes in neuron σ_{l_i}, rule $a^3/a \leftarrow a$ in neurons σ_{c_1}, σ_{c_2} and σ_{c_3} can be applied at time $t+1$. Consequently, the three neurons each receive a spike. Since neurons σ_{c_2} and σ_{c_3} each have a spike, rule $(a, a)/a^2 \leftarrow (a, a)$ is applied at time $t+2$. Thus neuron σ_r receives two spikes, indicating that register r is incremented by 1. Due to a spike in neuron σ_{c_1}, rules $a/a \leftarrow a$ in σ_{l_j} and σ_{l_k} can be applied. Therefore, one of the rules in the two neurons is chosen nondeterministically. There are two cases:

(1) At time $t+2$, if rule $a/a \leftarrow a$ in neuron σ_{l_j} is applied, then it receives a spike, indicating that the system Π starts to simulate the instruction l_j.

(2) At time $t+2$, if rule $a/a \leftarrow a$ in neuron σ_{l_k} is applied, then it receives a spike. With a spike in neuron σ_{l_k}, the system Π starts to simulate the instruction l_k.

Consequently, the ADD module correctly simulates the ADD instruction: starting from neuron σ_{l_i} receiving a spike, the number of spikes stored in neuron σ_r is added by 2, and then a spike is received nondeterministically by neurons σ_{l_j} or σ_{l_k}.

(2) Module SUB (shown in Fig. 6) — simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$.

Assume that a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is simulated at time t, and a spike is received by neuron σ_{l_i} (thus three spikes are generated by rule $a/a^3 \leftarrow a$). Because there are three spikes in neuron σ_{l_i}, rule $a^3/a \leftarrow a$ in neuron σ_r and rule $a^3/a \leftarrow a^2$ in neuron σ_{c_1} can be simultaneously applied at time $t+1$. Thus

neurons σ_r and σ_{c_1} each receive a spike. Based on the number of spikes in neuron σ_r, the following two cases are considered:

(1) At time $t+2$, if neuron σ_r has $2n+1$ ($n \geq 1$) spikes (at this time, register r stores the number n), then rule $(a(aa)^+, a)/a \rightarrow (a^3, a)$ in neuron σ_{c_2} can be applied, but rule $(a, a)/a \rightarrow (a, a)$ in neuron σ_{c_3} cannot be applied. Thus three spikes are consumed in neuron σ_r and a spike is removed from neuron σ_{c_1}, and simultaneously neuron σ_{c_2} receives a spike. At time $t+3$, neuron σ_{l_j} receives a spike by its rule $a/a \rightarrow a$, indicating that the system starts to simulate the instruction l_j.

(2) At time $t+2$, if neuron c_r contains only one spike (at this time, no number exists in register r), then rule $(a, a)/a \rightarrow (a, a)$ in neuron σ_{c_3} can be applied, but rule $(a(aa)^+, a)/a \rightarrow (a^3, a)$ in neuron σ_{c_2} cannot be applied. The number of spikes contained in neuron σ_r becomes 0 by the rule, and neuron σ_{c_3} receives a spike. At time $t+3$, neuron σ_{l_k} receives a spike by its rule $a/a \rightarrow a$. Therefore, the system Π starts to simulate the instruction l_k.

Therefore, the SUB module correctly simulates the SUB instruction: the system starts from neuron σ_{l_i} receiving a spike, and it ends after neuron σ_{l_j} receives a spike (if a number greater than zero is contained in register r) or neuron σ_{l_k} receives a spike (if register r has no number).

(3) Module FIN (shown in Fig. 7) — outputting the result of computation.

Suppose now that at time t, neuron σ_{l_h} receives a spike, indicating that the register machine stops (i.e., the halting instruction l_h is reached), and neuron σ_1 has $2n$ spikes (i.e., the number stored in register 1 is n). Because neuron σ_{l_h} has three spikes, rule $a^3/a \rightarrow a$ in neuron σ_1 and rule $a^3/a \rightarrow a^2$ in neuron σ_{c_1} are applied at time $t+1$. Neurons σ_1 and σ_{c_1} each receive a spike. Thus the number of spikes contained in neuron σ_1 becomes odd. Because there is a spike in neuron σ_{c_1} at time $t+2$, rule $(a(aa)^+, a)/a \rightarrow (a^2, \lambda)$ in neuron σ_{out} can be applied to receive a spike. At this time, two spikes are removed from neuron σ_1 (meaning that the number in register 1 is reduced by 1), but no spike is consumed in neuron σ_{c_1}. The process is repeated until only one spike is contained in neuronσ_1, and neuron σ_{out} receives a spike each time. At time $t+n+2$, because only one spike is stored in neuron σ_1 and neuronσ_{c_1} also has a spike, rule $(a, a)/\lambda \rightarrow (a, a)$ in neuron σ_{out} is applied. Thus the spikes in neurons σ_1 and σ_{c_1} are consumed. Consequently, from time $t+2$ to time $t+n+1$, neuron σ_{out} receives a total of n spikes, indicating the number in register 1 when the computation halts.

According to our discussion of the modules, register machine M is correctly simulated by DeP system Π_1, where each neuron has at most two rules. Consequently, the theorem holds. □

3.2. Turing universality of DeP systems as number-accepting devices

Theorem 2. $N_{acc}DeP_*^2 = NRE$.

Proof. A DeP system Π_2 is designed to simulate deterministic register machine $M = (m, H, l_0, l_h, I)$ working in accepting mode. This proof will be accomplished by modifying the proof of Theorem 1. The system Π_2 includes the modules of three types: a deterministic ADD module, a SUB module, and an INPUT module.

Fig. 8 shows the INPUT module. For convenience, the input spike train $10^{n-1}1$ is recoded as $53^{n-1}5$, where the digits "5" and "3" denote five spikes and three spikes, respectively, to be imported every time. Note that the interval between the first and second "5" spikes in the spike train can be determined by $(n+1)-1=n$, indicating that n is the number to be accepted.

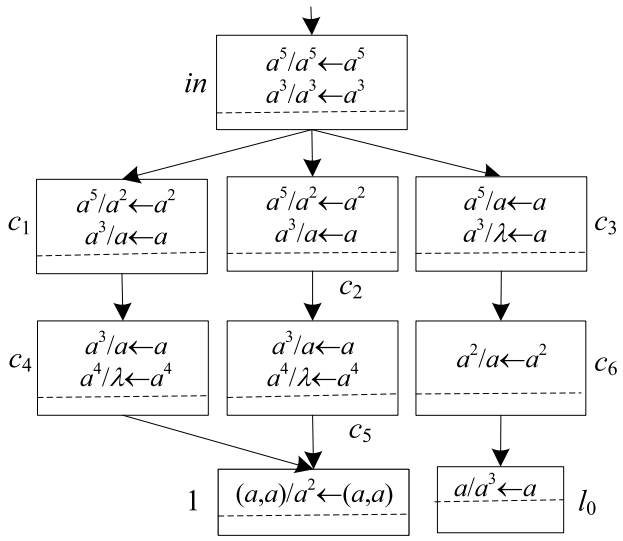

Fig. 8. The INPUT Module of Π_2.

Suppose that at time t, the first "5" spikes are received from the environment by neuron σ_{in}. At time $t+1$, rule $a^5/a^2 \rightarrow a^2$ in neuron σ_{c_1}, rule $a^5/a^2 \rightarrow a^2$ in neuron σ_{c_2}, and rule $a^5/a \rightarrow a$ in neuron σ_{c_3} are applied. Hence neurons σ_{c_2} and σ_{c_1} each receive two spikes and neuron σ_{c_3} receives a spike. At this time, neuron σ_{in} receives the first "3" spikes.

At time $t+2$, rule $a^3/a \rightarrow a$ in neuron σ_{c_1}, rule $a^3/a \rightarrow a$ in neuron σ_{c_2}, and rule $a^3/\lambda \rightarrow a$ in neuron σ_{c_3} are applied. Thus neurons σ_{c_2} and σ_{c_1} each have three spikes, while neuron σ_{c_3} still has only one spike. Moreover, neuron σ_{in} receives the second "3" spikes.

At time $t+3$, since neurons σ_{c_1} and σ_{c_2} each have three spikes, neurons σ_{c_4} and σ_{c_5} each receive a spike from neurons σ_{c_2} and σ_{c_1} by rule $a^3/a \rightarrow a$. Neurons σ_{c_2} and σ_{c_1} each receive a spike from neuron σ_{in} by rule $a^3/a \rightarrow a$. At this time, neuron σ_{in} receives the third "3" spikes.

At time $t+4$, neuron σ_1 receives two spikes from neurons σ_{c_4} and σ_{c_5} by rule $(a, a)/a^2 \rightarrow (a, a)$. Neurons σ_{c_4} and σ_{c_5} each receive a spike from neurons σ_{c_2} and σ_{c_1} by rule $a^3/a \rightarrow a$. Each of neurons σ_{c_2} and σ_{c_1} receives a spike from neuron σ_{in} by rule $a^3/a \rightarrow a$. At this time, neuron σ_{in} receives the fourth "3" spikes.

The process is repeated until neuron σ_{in} receives the second "5" spikes. At time $t+n+2$, neuron σ_{in} receives the second "5" spikes. At time $t+n+3$, rule $a^5/a^2 \rightarrow a^2$ in neuron σ_{c_1}, rule $a^5/a^2 \rightarrow a^2$ in neuron σ_{c_2}, and rule $a^5/a \rightarrow a$ in neuron σ_{c_3} are applied again, hence neurons σ_{c_2} and σ_{c_1} each contain four spikes and neuron σ_{c_3} has two spikes. At time $t+n+4$, four spikes in neurons σ_{c_1} and σ_{c_2} are consumed by rule $a^4/\lambda \rightarrow a^4$ in neurons σ_{c_4} and σ_{c_5}, and neuron σ_{c_6} receives a spike from neuron σ_{c_3} by rule $a^2/a \rightarrow a^2$. At time $t+n+5$, neuron σ_{l_0} receives a spike from neuron σ_{c_6}, but the rule in neuron σ_1 cannot be applied since neurons σ_{c_4} and σ_{c_5} have no spike.

Note that from time $t+4$ to time $t+n+4$, two spikes are received by neuron σ_1 each time. Therefore, $2n$ spikes are received in total by neuron σ_1 (meaning that the number of spikes in register 1 is n), and since a spike is received by neuron σ_{l_0}, the system starts to simulate the initial instruction l_0.

For accepting mode, deterministic ADD instructions, of the form $l_i : (ADD(r), l_j)$, are used in the register machine, shown in Fig. 9. Assume that at time t, neuron σ_{l_i} receives a spike by rule $a/a^3 \rightarrow a$. Thus neuron σ_{l_i} has three spikes. With three spikes in neuron σ_{l_i}, rule $a^3/a \rightarrow a$ can be applied at time $t+1$ in neurons σ_{c_1}, σ_{c_2}, and σ_{c_3}. Consequently, each of the three neurons

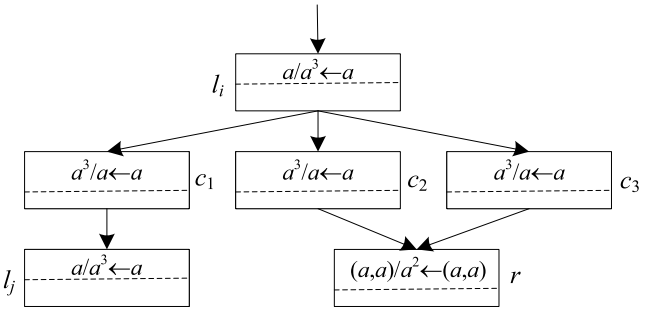

Fig. 9. Module ADD of Π_2 and Π_3, simulating $l_i : (ADD(r), l_j)$.

receives a spike. Since neurons σ_{c_2} and σ_{c_3} each have a spike, rule $(a, a)/a^2 \leftarrow (a, a)$ is applied at time $t + 2$. Hence two spikes are received by neuron σ_r. Because neuron σ_{c_1} has a spike, rule $a/a \leftarrow a$ in neuron σ_{l_j} is applied to receive a spike. With a spike in neuron σ_{l_j}, the system starts to simulate the instruction l_j.

Module SUB has not been changed (see Fig. 6). Module FIN is removed, but neuron σ_{l_h} is kept in the system. With the receipt of a spike by neuron σ_{l_h} register machine M attains instruction l_h and stops.

According to the discussion above, register machine is correctly simulated by DeP system Π_2 under accepting mode, where each neuron contains two rules at most. □

3.3. Small universal DeP systems for computing functions

We now design a small universal DeP system for computing functions. For computing the function $f : N^k \rightarrow N$, a register machine $M = (m, H, l_0, l_h, I)$ can work as follows: k initial arguments are introduced into k special registers in the register machine (generally, the first two registers are adopted), and all other registers are set to be empty; the register machine starts from instruction l_0, and then it works continually until it reaches halting instruction l_h; the number stored in a special register r_t is regarded as the computed value. Denote by $(\varphi_0, \varphi_1, \ldots)$ the fixed admissible enumeration of the unary partial recursive functions. If a recursive function g exists such that $\varphi_x(y) = M_u(g(x), y)$ holds for all natural numbers x, y, then the register machine is called universal.

Korec (1996) introduced a known small universal register machine for computing functions, denoted by $M_u = (8, H, l_0, l_h, I)$. The register machine M_u has 23 instructions and 8 registers numbered from 0 to 7. By introducing two numbers $g(x)$ and y into registers 1 and 2, the register machine M_u can compute any $\varphi_x(y)$; the number stored in register 0 is the computed function value when the register machine halts. A DeP system will be designed to simulate register machine M_u. For the sake of simplicity, the register machine M_u is changed: a new register 8 is introduced, and original halting instruction is replaced with three instructions: $l_{22} : (SUB(0), l_{23}, l_h)$; $l_{23} : (ADD(8), l_{22})$; $l_h : HALT$. Denote by M'_u the modified version of M_u, shown in Fig. 10. Therefore, register machine M'_u has nine registers, 25 labels, and 24 ADD and SUB instructions.

Theorem 3. *There exists a small universal DeP system having 115 neurons for computing functions.*

Proof. We design a DeP system Π_3 to simulate the universal register machine M'_u. The DeP system Π_3 consists of an INPUT module, an OUTPUT module, and ADD and SUB modules. The ADD and SUB modules are used to simulate the ADD and SUB instructions of M'_u, respectively. The INPUT module reads a spike

l_0: $(SUB(1), l_1, l_2)$ l_1: $(ADD(7), l_0)$
l_2: $(ADD(6), l_3)$ l_3: $(SUB(5), l_2, l_4)$
l_4: $(SUB(6), l_5, l_3)$ l_5: $(ADD(5), l_6)$
l_6: $(SUB(7), l_7, l_8)$ l_7: $(ADD(1), l_4)$
l_8: $(SUB(6), l_9, l_0)$ l_9: $(ADD(6), l_{10})$
l_{10}: $(SUB(4), l_0, l_{11})$ l_{11}: $(SUB(5), l_{12}, l_{13})$
l_{12}: $(SUB(5), l_{14}, l_{15})$ l_{13}: $(SUB(2), l_{18}\}, l_{19})$
l_{14}: $(SUB(5), l_{16}, l_{17})$ l_{15}: $(SUB(3), l_{18}, l_{20})$
l_{16}: $(ADD(4), l_{11})$ l_{17}: $(ADD(2), l_{21})$
l_{18}: $(SUB(4), l_0, l_{22})$ l_{19}: $(SUB(0), l_0, l_{18})$
l_{20}: $(ADD(0), l_0)$ l_{21}: $(ADD(3), l_{18})$
l_{22}: $(SUB(0), l_{23}, l_h)$ l_{23}: $(ADD(8), l_{22})$
l_h: $HALT$

Fig. 10. The universal register machine M'_u.

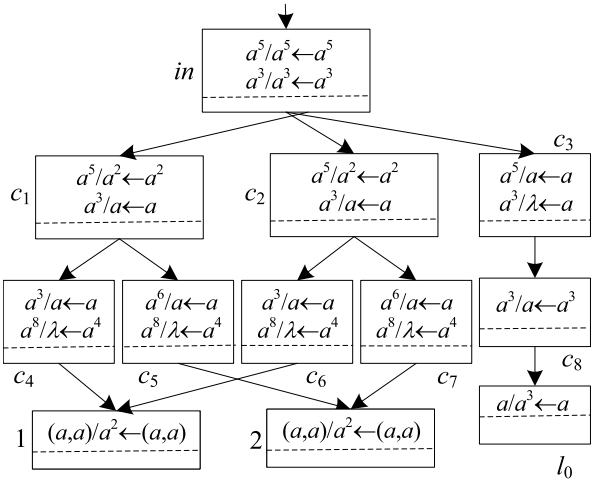

Fig. 11. INPUT Module of Π_3.

train from the environment, while the OUTPUT module exports the computation result.

When designing the DeP system Π_3, each register r in M'_u is associated with a neuron σ_r, and if the number $n \geq 0$ is stored in register r, then $2n$ spikes are contained in neuron σ_r. Moreover, each instruction l_i in M'_u is associated with a neuron σ_{l_i} in Π_3. If neuron σ_{l_i} receives a spike, then it starts the simulation of instruction l_i. If neuron σ_{l_h} contains a spike, then the system Π_3 completely simulates the computation of M'_u. As a result, the number of spikes received by the output neuron σ_{out} is the result computed by M'_u (in register 8). Suppose that all of the auxiliary neurons in the initial configuration are empty.

The INPUT module is shown in Fig. 11. For convenience, the input spike train $10^{g(x)}10^y1$ is recoded as $53^{g(x)}53^y5$, where the digits "5" and "3" respectively denote five spikes and three spikes to be imported each time. Note that neuron σ_1 has $2g(x)$ spikes and neuron σ_2 contains $2y$ spikes.

Suppose that at time t_1, neuron σ_{in} receives the first "5" spikes from the environment. Similar to the analysis of the INPUT module in the proof of Theorem 2, at time $t_1 + 1$, rule $a^5/a^2 \leftarrow a^2$ in neuron σ_{c_1}, rule $a^5/a^2 \leftarrow a^2$ in neuron σ_{c_2}, and rule $a^5/a \leftarrow a$ in neuron σ_{c_3} are applied, hence neurons σ_{c_2} and σ_{c_1} each receive

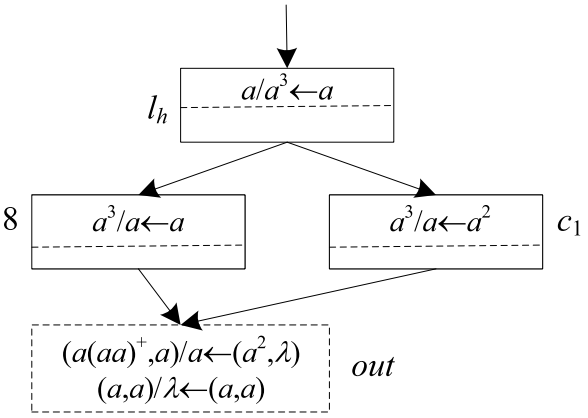

Fig. 12. OUTPUT Module of Π_3.

two spikes and neuron σ_{c_3} receives one spike. At this time, neuron σ_{in} receives the first "3" spikes. At time $t_1 + 2$, rule $a^3/a \leftarrow a$ in neuron σ_{c_1}, rule $a^3/a \leftarrow a$ in neuron σ_{c_2}, and rule $a^3/\lambda \leftarrow a$ in neuron σ_{c_3} are applied. Thus neurons σ_{c_2} and σ_{c_1} each contain three spikes, while neuron σ_{c_3} contains only one spike. At time $t_1 + 3$, because there are three spikes in neurons σ_{c_1} and σ_{c_2}, neurons σ_{c_4} and σ_{c_6} each receive a spike from neurons σ_{c_1} and σ_{c_2} by rule $a^3/a \leftarrow a$. At time $t_1 + 4$, neuron σ_1 receives two spikes from neurons σ_{c_4} and σ_{c_6} by rule $(a, a)/a^2 \leftarrow (a, a)$. The process is repeated until neuron σ_{in} receives the second "5" spikes. From time $t_1 + 4$ to time $t_1 + g(x) + 4$, $2g(x)$ spikes are received in total by neuron σ_1 (i.e., the number of spikes in register 1 is $g(x)$). Note that before time $t_1 + g(x) + 2$, neuron σ_{c_3} has only one spike.

Suppose that the second "5" spikes are received by neuron σ_{in} at time t_2 (in fact, $t_2 = t_1 + g(x) + 1$). Similarly, at time $t_2 + 1$, rule $a^5/a^2 \leftarrow a^2$ in neuron σ_{c_1}, rule $a^5/a^2 \leftarrow a^2$ in neuron σ_{c_2}, and rule $a^5/a \leftarrow a$ in neuron σ_{c_3} are applied, hence neurons σ_{c_2} and σ_{c_1} each have five spikes and neuron σ_{c_3} has two spikes. At time $t_2 + 2$, rule $a^3/a \leftarrow a$ in neuron σ_{c_1}, rule $a^3/a \leftarrow a$ in neuron σ_{c_2}, and rule $a^3/\lambda \leftarrow a$ in neuron σ_{c_3} are applied. Thus neurons σ_{c_2} and σ_{c_1} each contain six spikes, while neuron σ_{c_3} still has only two spikes. At time $t_2 + 3$, with six spikes in neurons σ_{c_1} and σ_{c_2}, neurons σ_{c_5} and σ_{c_7} each receive a spike from neurons σ_{c_2} and σ_{c_1} by rule $a^6/a \leftarrow a$. At time $t_2 + 4$, neuron σ_2 receives two spikes from neurons σ_{c_5} and σ_{c_7} by rule $(a, a)/a^2 \leftarrow (a, a)$. The process is repeated until neuron σ_{in} receives the third "5" spikes. From time $t_2 + 4$ to time $t_2 + y + 4$, $2y$ spikes are received in total by neuron σ_2 (i.e., the number of spikes in register 2 is y). Note that at time $t_2 + y + 1$, neuron σ_{c_1} receives a spike from neuron σ_{in}, so it has three spikes. At time $t_2 + y + 1$, rule $a^3/a \leftarrow a^3$ in neuron σ_{c_8} is used to receive a spike. Then neuron σ_{l_0} receives a spike, meaning that the system starts the simulation of initial instruction l_0.

From Fig. 10, we can observe that ADD instructions have the form $l_i : (ADD(r), l_j)$. Hence a deterministic ADD module can accomplish the simulation of the ADD instruction, as shown in Fig. 9. The working mechanism of the deterministic ADD module is illustrated in the proof of Theorem 2.

The SUB module in Fig. 6 can complete the simulation of SUB instruction $l_i : (SUB(r), l_j, l_k)$. The working principle of the SUB module is explained in the proof of Theorem 1.

Suppose now that the register machine M'_u halts, i.e., the instruction l_h arrives. Register 8 stores the computation result, which never decreases during the computation. The computation result will be exported by the OUTPUT module, as shown in Fig. 12.

Assume now that neuron σ_{l_h} receives a spike at time t, indicating that M'_u halts (that is, the halting instruction l_h is reached), and neuron σ_8 has $2n$ spikes (i.e., the number n is contained in register 8). Since neuron σ_{l_h} has three spikes, rule $a^3/a \leftarrow a$ in neuron σ_8 and rule $a^3/a \leftarrow a^2$ in neuron σ_{c_1} are applied at time $t + 1$. Neurons σ_8 and σ_{c_1} each receive a spike. Thus the number of spikes in neuron σ_8 becomes odd. At time $t + 2$, because there is a spike in neuron σ_{c_1}, rule $(a(aa)^+, a)/a \leftarrow (a^2, \lambda)$ in neuron σ_{out} can be applied to receive a spike. At this time, two spikes are removed from neuron σ_8 (meaning that the number in register 8 is decreased by 1), but no spike is consumed in neuron σ_{c_1}. The process is repeated until neuron σ_8 has only one spike, and neuron σ_{out} receives a spike each time. At time $t + n + 2$, because each of neurons σ_8 and σ_{c_1} has a spike, rule $(a, a)/\lambda \leftarrow (a, a)$ in neuron σ_{out} is applied. Thus the spikes in neurons σ_8 and σ_{c_1} are consumed. Consequently, from time $t + 2$ to time $t + n + 1$, n spikes are received in total by neuron σ_{out}, indicating exactly the number in register 8 when M'_u halts.

According to the discussion above, register machine M'_u is correctly simulated by DeP system Π_3. In DeP system Π_3, 115 neurons in total are used: (i) nine neurons for nine registers; (ii) 25 neurons for 25 instructions; (iii) 30 neurons for 10 ADD instructions; (iv) 42 neurons for 14 SUB instructions; (v) eight neurons for the INPUT module; (vi) one neuron for the OUTPUT module. □

4. Conclusions and future work

A new neural-like P system was investigated in this paper, called dendrite P (DeP) systems, abstracted by two characteristics of dendrites of nerve cells: (i) dendrites can perform mixed computations of analog and digital signals, and (ii) dendrites have the feedback characteristic. Based on the first characteristic, dendrites were regarded as information processors, while the second was used to develop a new kind of firing rules, dendrite rules. From the working mechanism, DeP systems differs from SNP systems in the following aspects:

(1) Neurons in DeP systems use a firing–storing process, i.e., the spikes in prepositive neurons of each neuron are processed by its dendrite rules, and then the generated spikes are stored in the neuron. However, neurons in SNP systems adopt the opposite process, i.e., storing–firing process.
(2) The firing condition of each dendrite rule depends on states of prepositive neurons of the neuron where the rule resides, and has nothing to do with the state of the neuron. However, in SNP systems, the firing condition of the usual firing rule depends only on the state of the neuron where the rule resides.
(3) Due to multiple regular expressions of dendrite rules, DeP systems provide a collaborative control mechanism of several neurons. However, SNP systems (and SNP systems with rules on synapses) lack this mechanism.

We proved that DeP systems can generate/accept any set of Turing-computable numbers, and constructed a small Turing-universal DeP system of 115 neurons for computing functions.

DeP systems seem to be more suitable for solving some real-life problems because of their collaborative control mechanism. Our future work will attempt to apply DeP systems to solve the problems, such as supervisory control problems in discrete event systems. In addition, DeP systems have the interesting feature: a conflict case exists in DeP systems when dendrite rules in different neurons share one or more prepositive (source) neurons. When discussing universality, the conflict case is processed by the nondeterministic choice strategy. However, the conflict case is potentially useful for some real-life applications, such as flexible manufacturing systems (FMSs), because the feature can be used to characterize the problems related to deadlock avoidance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Bernardini, F., & Gheorghe, M. (2004). Population P systems. *Journal of Universal Computer Science*, *10*(5), 509–539. http://dx.doi.org/10.3217/jucs-010-05-0509.

Cabarle, F. G. C., Adorna, H. N., Jiang, M., & Zeng, X. (2016). Spiking neural P systems with scheduled synapses. *IEEE Transactions on Nanobioscience*, *27*(5), 1337–1347. http://dx.doi.org/10.1109/TNB.2017.2762580.

Carandang, J. P., Villaflores, J. M. B., Cabarle, F. G. C., Adorna, H. N., & Martínez-Del-Amor, M. A. (2017). CuSNP: Spiking neural P systems simulators in CUDA. *Romanian Journal of Information Science and Technology*, *20*(1), 57–70.

Cavaliere, M., Ibarra, O. H., Păun, G., Egecioglu, O., Ionescu, M., & Woodworth, S. (2009). Asynchronous spiking neural P systems. *Theoretical Computer Science*, *410*(24), 2352–2364.

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

Chen, H. M., Ishdorj, T.-O., & Păun, G. (2007). Computing along the axon. *Progress in Natural Science*, *17*(4), 417–423.

Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., & Vaszil, G. (2009). Variants of P colonies with very simple cell structure. *International Journal of Computers, Communications & Control*, *IV*(3), 224–233. http://dx.doi.org/10.15837/ijccc.2009.3.2430.

Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., & Peng, H. (2019). Membrane computing and image processing: A short survey. *Journal of Membrane Computing*, *1*, 58–73. http://dx.doi.org/10.1007/s41965-018-00002-x.

Díaz-Pernil, D., Peña-Cantillana, F., & Gutiérrez-Naranjo, M. A. (2013). A parallel algorithm for skeletonizing images by using spiking neural P systems. *Neurocomputing*, *115*, 81–91. http://dx.doi.org/10.1016/j.neucom.2012.12.032.

Freund, R., Păun, G., & Pérez-Jiménez, M. J. (2005). Tissue-like P systems with channel-states. *Theoretical Computer Science*, *330*(1), 101–116. http://dx.doi.org/10.1016/j.tcs.2004.09.013.

Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M. J., Turcanu, A., Valencia-Cabrera, L., et al. (2013). 3-Col problem modelling using simple kernel P systems. *International Journal of Computational Methods*, *90*(4), 816–830. http://dx.doi.org/10.1080/00207160.2012.743712.

Ibarra, O. H., Păun, A., & Rodríguez-Paón, A. (2009). Sequential SNP systems based on min/max spike number. *Theoretical Computer Science*, *410*(30), 2982–2991. http://dx.doi.org/10.1016/j.tcs.2009.03.004.

Ionescu, M., Păun, G., Pérez-Jiménez, M. J., & Yokomori, T. (2011). Spiking neural dP systems. *Fundamenta Information*, *111*(4), 423–436.

Ionescu, M., Păun, G., & Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae*, *71*, 279–308.

Korec, I. (1996). Small universal register machines. *Theoretical Computer Science*, *168*(2), 267–301. http://dx.doi.org/10.1016/S0304-3975(96)00080-1.

London, M., & Hausser, M. (2005). Dentritic computation. *Annual Review of Neuroscience*, *28*, 503–532.

Macías-Ramos, L. F., Pérez-Jiménez, M. J., Song, T., & Pan, L. (2015). Extending simulation of asynchronous spiking neural P systems in P–lingua. *Fundamenta Informaticae*, *136*(3), 253–267.

Martínez-Del-Amor, M. A., Macías-Ramos, L. F., Valencia-Cabrera, L., & Pérez-Jiménez, M. J. (2016). Parallel simulation of population dynamics P systems: Updates and roadmap. *Natural Computing*, *15*(4), 565–573.

Moore, A. J., Ravassard, P. M., Ho, D., Acharya, L., Kees, A. L., Vuong, C., et al. (2017). Dynamics of cortical dendritic membrane potential and spikes in freely behaving rats. *Science*, *355*(6331), aaj1497.

Pan, L., Păun, G., Zhang, G., & Neri, F. (2017). Spiking neural P systems with communication on request. *International Journal of Neural Systems*, *28*(8), 1–13. http://dx.doi.org/10.1142/S0129065717500423.

Pan, L., Wang, J., & Hoogeboom, H. J. (2012). Spiking neural P systems with astrocytes. *Neural Computation*, *24*(3), 805–825. http://dx.doi.org/10.1162/NECO_a_00238.

Pavel, A. B., & Buiu, C. (2012). Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, *11*(3), 387–393.

Peng, H., Chen, R., Wang, J., Song, X., Wang, T., Yang, F., et al. (2018). Competitive spiking neural P systems with rules on synapses. *IEEE Transactions on NanoBioscience*, *16*(8), 888–895. http://dx.doi.org/10.1109/TNB.2017.2783890.

Peng, H., Li, B., Wang, J., Song, X., Wang, T., Valencia-Cabrera, L., et al. (2020). Spiking neural P systems with inhibitory rules. *Knowledge-Based Systems*, *188*, 1–10. http://dx.doi.org/10.1016/j.knosys.2019.105064, 105064.

Peng, H., Lv, Z., Li, B., Luo, X., Wang, J., Song, X., et al. (2020). Nonlinear spiking neural P systems. *International Journal of Neural Systems*, http://dx.doi.org/10.1142/S0129065720500082.

Peng, H., Shi, P., Wang, J., Riscos-Núñez, A., & Pérez-Jiménez, M. J. (2017). Multiobjective fuzzy clustering approach based on tissue-like membrane systems. *Knowledge-Based Systems*, *125*, 74–82. http://dx.doi.org/10.1016/j.knosys.2017.03.024.

Peng, H., & Wang, J. (2018). Coupled neural P systems. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(6), 1672–1682.

Peng, H., Wang, J., Ming, J., Shi, P., Pérez-Jiménez, M. J., Yu, W., et al. (2018). Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems. *IEEE Transaction on Smart Grid*, *9*(5), 4777–4784. http://dx.doi.org/10.1109/TSG.2017.2670602.

Peng, H., Wang, J., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2015). An unsupervised learning algorithm for membrane computing. *Information Sciences*, *304*(20), 80–91. http://dx.doi.org/10.1016/j.ins.2015.01.019.

Peng, H., Wang, J., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2019). Dynamic threshold neural P systems. *Knowledge-Based Systems*, *163*, 875–884.

Peng, H., Wang, J., Pérez-Jiménez, M. J., Wang, H., Shao, J., & Wang, T. (2013). Fuzzy reasoning spiking neural P system for fault diagnosis. *Information Sciences*, *235*, 106–116. http://dx.doi.org/10.1016/j.ins.2012.07.015.

Peng, H., Wang, J., Shi, P., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2016). An extended membrane system with active membrane to solve automatic fuzzy clustering problems. *International Journal of Neural Systems*, *26*(3), 1–17. http://dx.doi.org/10.1142/S0129065716500040.

Peng, H., Wang, J., Shi, P., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2017). Fault diagnosis of power systems using fuzzy tissue-like P systems. *Integrated Computer-Aided Engineering*, *24*(4), 401–411. http://dx.doi.org/10.3233/ICA-170552.

Peng, H., Yang, J., Wang, J., Wang, T., Sun, Z., Song, X., et al. (2017). Spiking neural P systems with multiple channels. *Neural Networks*, *95*, 66–71. http://dx.doi.org/10.1016/j.neunet.2017.08.003.

Păun, G. (2000). Computing with membranes. *Journal of Computer System Sciences*, *61*(1), 108–143. http://dx.doi.org/10.1006/jcss.1999.1693.

Păun, G. (2002). *Membrane computing: An introduction*. Springer.

Păun, G. (2007). Spiking neural P systems with astrocyte-like control. *Journal of Universal Computer Science*, *13*(11), 1707–1721. http://dx.doi.org/10.3217/jucs-013-11-1707.

Păun, A., & Păun, G. (2007). Small universal spiking neural P systems. *BioSystems*, *90*(1), 48–60. http://dx.doi.org/10.1016/j.biosystems.2006.06.006.

Păun, G., Rozenberg, G., & Salomaa, A. (2010). *The oxford handbook of membrane computing*. Oxford University Press, Inc.

Păun, A., & Sidoroff, M. (2012). Sequentially induced by spike number in SNP systems: Small universal machines. In *Membrane computing* (pp. 333–345). Springer.

Păun, G., & Pérez-Jiménez, M. J. (2010). Solving problems in a distributed way in membrane computing: dP systems. *International Journal of Computers, Communications & Control*, *V*(2), 238–250. http://dx.doi.org/10.1080/00207160.2012.743712.

Păun, G., & Păun, R. (2006). Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, *73*(1, 2), 213–227. http://dx.doi.org/10.3217/jucs-010-05-0509.

Song, T., Pan, L., & Păun, G. (2014). Spiking neural P systems with rules on synapses. *Theoretical Computer Science*, *529*, 82–95. http://dx.doi.org/10.1016/j.tcs.2014.01.001.

Song, T., & Pan, L. (2015). Spiking neural P systems with rules on synapses working in maximum spiking strategy. *IEEE Transaction on Nanobioscience*, *14*(4), 465–477.

Song, T., Pan, L., & Păun, G. (2012). Asynchronous spiking neural P systems with local synchronization. *Information Sciences*, *219*, 197–207. http://dx.doi.org/10.1016/j.ins.2012.07.023.

Song, X., Wang, J., Peng, H., Ning, G., Sun, Z., Wang, T., et al. (2018). Spiking neural P systems with multiple channels and anti-spikes. *Biosystems*, *169–170*, 13–19.

Song, B., Zhang, C., & Pan, L. (2016). Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences*, *378*(C), 177–193. http://dx.doi.org/10.1016/j.ins.2016.10.046.

Stuart, G., Spruston, N., Sakmann, B., & Hausser, M. (1997). Action potential initiation and backpropagation in central neurons. *Trends in Neurosciences*, *20*, 125–131.

Wang, J., Hoogeboom, H. J., Pan, L., Păun, G., & Pérez-Jiménez, M. J. (2010). Spiking neural P systems with weights. *Neural Computation*, *22*, 2615–2646. http://dx.doi.org/10.1162/NECO_a_00022.

Wang, J., Shi, P., & Peng, H. (2016). Membrane computing model for IIR filter design. *Information Sciences*, *329*, 164–176. http://dx.doi.org/10.1016/j.ins.2015.09.011.

Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M. J., & Wang, T. (2013). Weighted fuzzy spiking neural P systems. *IEEE Transactions on Fuzzy Systems*, *21*(2), 209–220. http://dx.doi.org/10.1109/TFUZZ.2012.2208974.

Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., & Pérez-Jiménez, M. J. (2015). Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transaction on Power Systems*, *30*(3), 1182–1194. http://dx.doi.org/10.1109/TPWRS.2014.2347699.

Wu, T., Păun, A., Zhang, Z., & Pan, L. (2018). Spiking neural P systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3349–3360. http://dx.doi.org/10.1109/TNNLS.2017.2726119.

Xue, J., Camino, A., Bailey, S. T., Liu, X., Li, D., & Jia, Y. (2018). Automatic quantification of choroidal neovascularization lesion area on OCT angiography based on density cell-like P systems with active membranes. *Biomedical Optics Express*, *9*(7), 3208–3219.

Zeng, X., Zhang, X., Song, T., & Pan, L. (2014). Spiking neural P systems with thresholds. *Neural Computation*, *26*(7), 1340–1361. http://dx.doi.org/10.1162/NECO_a_00605.

Zhang, G., Gheorghe, M., Pan, L., & Pérez-Jiménez, M. J. (2014). Evolutionary membrane computing: A comprehensive survey and new results. *Information Sciences*, *279*, 528–551.

Zhang, X., Pan, L., & Păun, A. (2015). On the universality of axon P systems. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(11), 2816–2829.

Zhang, G., Rong, H., Neri, F., & Pérez-Jiménez, M. J. (2014). An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, *24*(05), 1–16.

Zhang, X., Wang, J., & Pan, L. (2009). A note on the generative power of axon P systems. *International Journal of Computers, Communications & Control*, *IV*(1), 92–98.

Zhang, Z., Wu, T., Păun, A., & Pan, L. (2016). Numerical P systems with migrating variables. *Theoretical Computer Science*, *641*(C), 85–108. http://dx.doi.org/10.1016/j.tcs.2016.06.004.

Zhang, X., Zeng, X., Luo, B., & Pan, L. (2014). On some classes of sequential spiking neural P systems. *Neural Computation*, *26*(5), 974–997. http://dx.doi.org/10.1162/NECO_a_00580.

Zhang, X., Zeng, X., & Pan, L. (2008). On string language generated by spiking neural P systems with exhaustive use of rules. *Natural Computing*, *90*(1), 535–549. http://dx.doi.org/10.1007/s11047-008-9079-7.

Zhao, Y., Liu, X., & Qu, J. (2012). The k-medoids clustering algorithm by a class of P system. *Journal of Information & Computational Science*, *9*(18), 5777–5790.