Spiking Neural P Systems with Addition/Subtraction Computing on Synapses

Yun Jiang[1,2] ⋆ and Zhiqiang Chen[1,2]

[1] Chongqing Engineering Laboratory for Detection, Control and Integrated Systems
Chongqing Technology and Business University, Chongqing 400067, China
[2] School of Computer Science and Information Engineering,
Chongqing Technology and Business University, Chongqing 400067, China
jiangyun@email.ctbu.edu.cn

Summary. Spiking neural P systems (SN P systems, for short) are a class of distributed and parallel computing models inspired from biological spiking neurons. In this paper, we introduce a variant called SN P systems with addition/subtraction computing on synapses (CSSN P systems). CSSN P systems are inspired and motivated by the shunting inhibition of biological synapses, while incorporating ideas from dynamic graphs and networks. We consider addition and subtraction operations on synapses, and prove that CSSN P systems are computationally universal as number generators, under a normal form (i.e. a simplifying set of restrictions).

Key words: Membrane computing, Spiking neural P system, Computing on synapse, Computationally universal

1 Introduction

Brain is a rich source of inspiration for informatics. Specifically, it has provided plenty of ideas to construct high performance computing models, as well as to design efficient algorithm. Inspired from the biological phenomenon that neurons cooperate in the brain by exchanging spikes via synapses, various neural-like computing models have been proposed. In the framework of membrane computing, a kind of distributed and parallel neural-like computing model were proposed in 2006 [1], which is called spiking neural P systems (SN P systems for short).

SN P systems have neurons that process only one type of symbols, the spike, based on the indistinct signal used by biological neurons. Neurons are placed on nodes of a directed graph, and the edges between neurons are called synapses, again based on synapses of biological neurons. SN P systems processes spikes by applying rules, and two of the most common types are firing rules and forgetting rules: the former rules produce one or more spike, which is/are sent from the source neuron

⋆ Corresponding author.

to every neuron connected by a synpase, while the latter rules remove spikes from the neuron.

Since the human brain and biological neurons are rich sources of computing ideas, many variants of SN P systems have been introduced, taking inspiration from biological phenomena, e.g. synapse weight, neuron division, astrocytes, inhibitory synapses, as in [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Investigation on the theoretical and practical usefulness have also been applied to these variants: their computing power in relation to well-known models of computation, e.g. finite automata, register machines, grammars, computing numbers or strings as in [17, 18, 19, 20, 21, 22, 23, 24, 25, 26]; computing efficiency in solving hard problems, as in [27, 28].

Moreover, practical applications and software for simulations have been developed for SN P systems and their variants: to design logic gates, logic circuits [29] and databases [30], to perform basic arithmetic operations [31][32], to represent knowledge [33], to diagnose fault [34, 35, 36], to approximately solve combinatorial optimization problems [37].

In this work, we introduce a variant of SN P systems which we refer to as SN P systems with addition/subtraction computing on synapses (CSSN P systems, for short). CSSN P systems take inspirations and motivations from biological, mathematical and computing sources.

Biologically, it is known that not only neurons but also synapses can process spikes, as in [38]. Synapses monitor the spikes go through it and change the value of spikes according to their excitatory or inhibitory. Hence, it is natural to consider addition/subtraction computing of two consecutive spikes on synapses.

The mathematical and computing inspirations are taken from the study of dynamic graphs. Since SN P systems are in essence static graphs, it is natural to consider them for dynamic graphs as well.

In the survey of dynamic graphs, two main kinds of structural evolutions of the graphs are identified: node-centric evolutions, i.e. nodes or vertices are the focus, and edge-centric evolutions, i.e. edges are the focus. In the framework of SN P systems, several works have focused on dynamism for the neuron, as in [28][13]. More recently, SN P systems with structural plasticity were introduced in [39], with subsequent works in [40, 41, 42]. In these systems, synapses can be created or removed by plasticity rules of neurons, hence, structural evolution of the systems are more edge-centric.

For CSSN P systems however, we further focus on synapse dynamism, but this time we add an addition or subtraction computing to each synapse in the system. Specifically, for two consecutive spikes get through the synapse, if excitatory spike come first, the synapse does addition, otherwise, subtraction. Furthermore, we show that CSSN P systems are computationally universal, under a normal form (m. ore details below), for generating numbers

This work is organized as follows: Section 2 provides the definition of CSSN P systems and their semantics; Section 3 provides an example of CSSN P sys-

tems; Section 4 provides universality result on CSSN P systems; At last, Section 5 concludes this work and provides further directions for research.

2 Spiking Neural P Systems with Computing on Synapses

In this section we define our proposed variant, and provides the semantics. A spiking neural P system with computing on synapses, CSSN P system for short, of degree $m \geq 1$ is a construct of the form

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

where:

1. $O = \{a\}$ is a singleton alphabet, and a is called spike;
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$; $n_i \geq 0$ is the initial number of spikes contained in the neuron σ_i; R_i is a finite set of rules of the following two forms:
   (a) Firing rule: $E/a^c \to a^p; d$, where E is a a regular expression over $\{a\}$, $c \geq 1$, $d \geq 0$, with the restriction $c \geq p$. Specifically, when $d = 0$, it can be omitted;
   (b) Forgetting rule: $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \to a^p; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ is the set of synapses between neurons, with restriction $(i, i) \notin syn$ for $1 \leq i \leq m$, which means no σ_i has a synapse to itself;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the input and output neuron, respectively.

The *in* or *out* elements in the construction can be omitted, depending on whether they exist in the system or not. For a given neuron σ_i (also called neuron i or simply σ_i), we illustrate σ_i as an oval, and synapses between neurons as arcs. The input neuron has a synapse or arc from the environment, i.e. everything outside or not part of the system, while the output neuron has a synapse to the environment.

The firing rule of the form $E/a^c \to a^p; d$ with $c \geq p \geq 1$ is called an *extended rule*; if $p = 1$, the rule is called a *standard rule*. As a convention, if $L(E) = \{a^c\}$, the rule can be simply written as $a^c \to a^p; d$. Specifically, if $d = 0$, it can be omitted and the rule can be simply written as $a^c \to a^p$.

The semantics of applying firing rules are as follows: if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the firing rule $E/a^c \to a^p; d \in R_i$ can be applied, i.e., the number of spikes k in σ_i satisfies the requirement for applying the rule. Applying such a rule means consuming c spikes from σ_i and producing p spikes after d time units, thus $k-c$ spikes remains in σ_i. If $d = 0$, then the produced spikes are released immediately, and if $d = 1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is applied at step t, neuron σ_i becomes closed in the interval $[t, t + d)$, i.e., it cannot receive spikes and spikes

sent to it while it is closed are lost. At step $t+d$, neuron σ_i becomes open, i.e., it can receive spikes again, and at the same time it sends p spikes to come across all the synapses such that $(i,j) \in syn$.

The semantics of applying forgetting rules is as follows: if the neuron contains exactly s spikes, then the forgetting rule $a^s \to \lambda$ can be used, and this means that all s spikes are removed from the neuron.

The semantics of computing synapse is as follows: each synapse from syn monitors the flow of spikes come across it, and do the computation according to the dynamic status of these spikes. Specifically, synapses will do addition, subtraction or nothing according to the comparison result of two consecutive flows of spikes, and the spikes sent to the receiving neuron (neuron σ_j such that $(i,j) \in syn$) will be the results of these computation.

For each synapse $(i,j) \in syn$, there are several flows of spikes comes across the synapse during the computation of the system, i.e. there is a spike train on the synapse. In this case, synapse will compare the number of spikes come across it consecutively, and get excitatory, inhibitory, or normal accordingly. We say that two flows of spikes a^p and a^q come across the synapse consecutively, i.e. spikes a^p at step t and spikes a^q at step $t+1$, then there are three kinds of relationship between a^p and a^q.

case 1: $p < q$, the synapse gets excited and do addition, and the receiving neuron σ_j will get spikes a^{p+q};
case 2: $p = q$, the synapse gets normal and do nothing, and the receiving neuron σ_j will get spikes a^p;
case 3: $p > q$, the synapse gets inhibited and do subtraction, and the receiving neuron σ_j will get spikes a^{p-q}.

Specially, it is possible that during the computation of the system, there is only one flow of spikes comes across the synapse, for instance, spikes a^s come across the synapse in one step. In this case, there is no comparison and the synapse does not compute, and the receiving neuron σ_j will get spikes a^s.

A *configuration* of the system at a given step is the contribution of spikes among neurons, and the status of each neuron, whether closed or open. The *initial configuration* is given by n_i of each neuron σ_i. The system reaches a *halting configuration* when there is no rule can be applied and all neurons are open. A *computation* is defined as a sequence of configuration transitions, from an initial configuration, and following rule application semantics and synapse computing semantics. A *computation halts* if the system reaches a halting configuration.

The result of the computation can be defined in various ways in SN P systems. In this work we use the following definition: when a computation halts, the number of spikes present in the output neuron is said to be computed by an CSSN P system Π. We denote the set of all number computed in this way by Π as $N_{gen}(\Pi)$. In $N_{gen}(\Pi)$ we have Π able to *generate numbers* (we also say that Π works in the *generative mode*).

CSSN P systems can also accept numbers i.e. Π works in the *acceptive mode*. When working in the *acceptive mode*, the output neuron is ignored, and Π work as follows: a number is introduced into Π as the number of spikes present in the input neuron in the initial configuration, and the number is accepted by Π if the computation halts. The set of numbers accepted this way by Π is denoted as $N_{acc}(\Pi)$.

The families of all sets of $N_\alpha(\Pi)$, with $\alpha \in \{gen, acc\}$ are denoted as $N_\alpha SNPCOS_m(rule_k, cons_r, forg_q)$, with at most $m \geq 1$ neurons in the system, at most $k \geq 1$ rules in each neuron, consuming at most $r \geq 1$ spikes in any firing rule of any neuron, and forgetting at most $q \geq 1$ spikes in any forgetting rule of any neuron. We note that the parameter for the delay for the firing rules are specified in other literature, e.g. in [], but here we do not use it so the parameter is omitted.

3 An Example

In this section we provide an example Π_1 to further clarify the semantics of CSSN P systems, which is shown in Fig. 1.

The system Π_1 is composed of two neurons, labeled with 1 and 2, and they are the input and output neuron, respectively. Formally, system Π_1 is a structure of the form $\Pi = (O, \sigma_1, \sigma_2, syn, 1, 2)$, where:

- $O = \{a\}$;
- $\sigma_1 = (5, R_1)$, with $R_1 = \{a^5 \to a^4, a^5/a^2 \to a^2, a^5/a^3 \to a^3, a^3 \to a^3, a^3 \to a^2, a^2 \to a^2\}$;
- $\sigma_2 = (0, R_2)$, with $R_2 = \emptyset$;
- $syn = \{(1, 2)\}$.

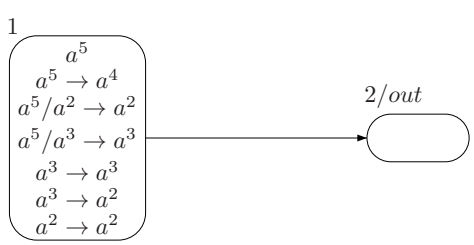

Fig. 1. A simple example of an SN P system with computing synapse

Neuron σ_1 fires at the first step of the computation. As shown in the table below, there are four sets of rules to apply. Also, the flow of spikes on synapse $(1, 2)$, computing on synapse, and spikes received by σ_2 are present in detail.

rules to apply	flow of spikes on synapse	computing on synapse	spikes received by σ_2
$a^5 \to a^4$	a^4	none	a^4
$a^5/a^2 \to a^2$, $a^3 \to a^3$	a^2a^3	addition	a^5
$a^5/a^2 \to a^2$, $a^3 \to a^2$	a^2a^2	none	a^2
$a^5/a^3 \to a^3$, $a^2 \to a^2$	a^3a^2	subtraction	a

From the table we can see that: when working in the generative mode, the set of all number computed by Π is $\{1, 2, 4, 5\}$, i.e. $\Pi_{gen} = \{1, 2, 4, 5\}$.

4 Universality Results

In this section, we present universality results for SN P systems with computing on synapses, for the generating modes.

Theorem 1. $N_{gen}SNPCOS_*(rule_3, cons_5, forg_5) = NRE$.

Proof. In order to prove Theorem, it is enough to simulate a register machine M with an CSSN P system Π with restrictions in the Theorem statement. Before constructing Π, we provide a general overview of the computation as follows: each register r in M corresponds to a σ_r in Π. If r stores the number n, then σ_r stores $2n$ spikes. If M applies an instruction l_i that performs some operation $OP \in \{ADD, SUB, HALT\}$, this means that a corresponding neuron σ_{l_i} becomes activated in order to simulate OP. Without loss of generality, register 1 of M is the output register and this register is never subjected to a SUB instruction. In this way, the spikes in σ_1 are never decremented.

In what follows, we provide the modules in most cases in a graphical manner for easier reference. The initial configuration of Π is such that all neurons are empty, except for σ_{l_0} which contains three spikes. The three spikes in σ_{l_0} begins the computation of Π, corresponding to simulating the initial instruction l_0 of M.

Module ADD: The module simulating $l_i : (ADD(r), l_j, l_k)$ is given in Fig. 2.

Once σ_{l_i} is activated, both neurons l_{i_1} and l_{i_2} receive three spikes.

With three spikes inside, neuron l_{i_2} applies the rule $a^3/a^2 \to a^2$ first, and then the rule $a \to \lambda$. As a result, the spike trains on synapses (l_{i_2}, r) and (l_{i_2}, l_{i_4}) are $a^2\lambda$, and the computing on these synapses is subtraction. In this way, σ_r receive two spikes, corresponding to incrementing the register r by one, and $\sigma_{l_{i_4}}$ also gets two spikes.

With three spikes inside, neuron l_{i_1} applies the rule $a^3/a \to a$ first, and then it must nondeterministically choose to apply either $a^2 \to a^2$ or $a^2 \to \lambda$. If the former rule is chosen, then the spike trains on synapses (l_{i_1}, l_{i_3}) and (l_{i_1}, l_{i_4}) are aa^2, and the synapses does addition. In this way, $\sigma_{l_{i_3}}$ receives three spikes, gets activated, and sends three spikes to σ_{l_j}, which makes neuron l_j activated; $\sigma_{l_{i_4}}$ receives five spikes (three from synapse (l_{i_1}, l_{i_4}) and two from synapse (l_{i_2}, l_{i_4})), which are forgotten according to the rule $a^5 \to \lambda$. If the latter rule is applied, then the spike trains on synapses (l_{i_1}, l_{i_3}) and (l_{i_1}, l_{i_4}) are $a\lambda$, and the synapses

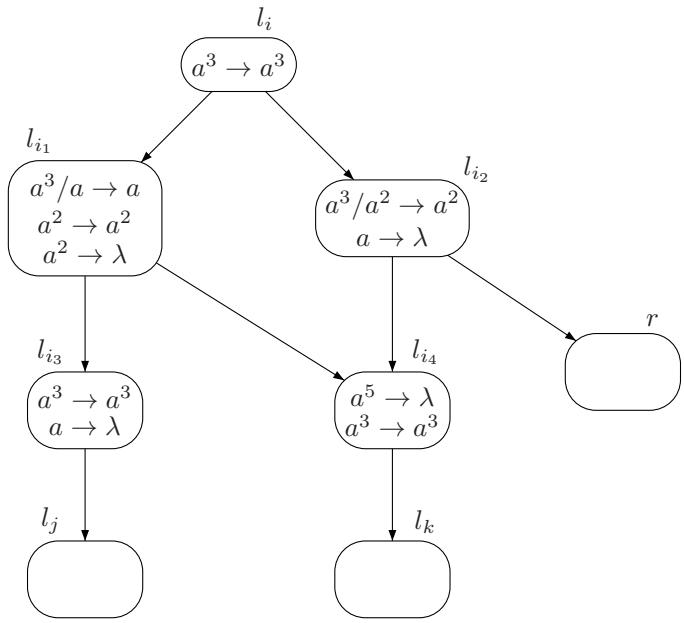

Fig. 2. Module ADD

does subtraction. In this way, $\sigma_{l_{i_3}}$ receives one spikes, which is forgotten according to the rule $a \to \lambda$; $\sigma_{l_{i_4}}$ receives three spikes (one from synapse (l_{i_1}, l_{i_4}) and two from synapse (l_{i_2}, l_{i_4})), gets activated, and sends three spikes to σ_{l_k}, which makes neuron l_k activated.

The functioning of the ADD module correctly simulates $l_i : (ADD(r), l_j, l_k)$ by incrementing σ_r with two spikes, followed by nondeterministically activating either σ_{l_j} or σ_{l_k}.

Module SUB: The module for simulating $l_i : (SUB(r), l_j, l_k)$ is given in Fig. 3. In the instruction l_i of M, we have two case, depending if r stores an empty or nonempty value.

Once σ_{l_i} is activated, both neurons r and l_{i_1} receive three spikes.

With three spikes inside, neuron l_{i_1} applies the rule $a^3/a \to a$ first, and then the rule $a^2 \to a^2$. As a result, the spike trains on synapses (l_{i_1}, l_{i_3}) and (l_{i_1}, l_{i_4}) are aa^2, and the computing on these synapses is addition. In this way, both $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$ receive three spikes.

Now, let's take a look at neuron σ_r.

On one hand, if register r stores a nonempty value $n \geq 1$, this means that initially there are at least two spikes in σ_r. After receiving three spikes from σ_{l_i}, σ_r contains at least five spikes (in general it contains $2n + 3$, $n \geq 1$ spikes). Only the rule $a^5(a^2)^*/a^5 \to a^2$ can be applied by σ_r. Five spikes are removed from σ_r (hence, only $2(n-1)$ spikes remain in σ_r) and two spikes is produced. The removal of five spikes corresponds to decrementing register r by one. Through the neuron

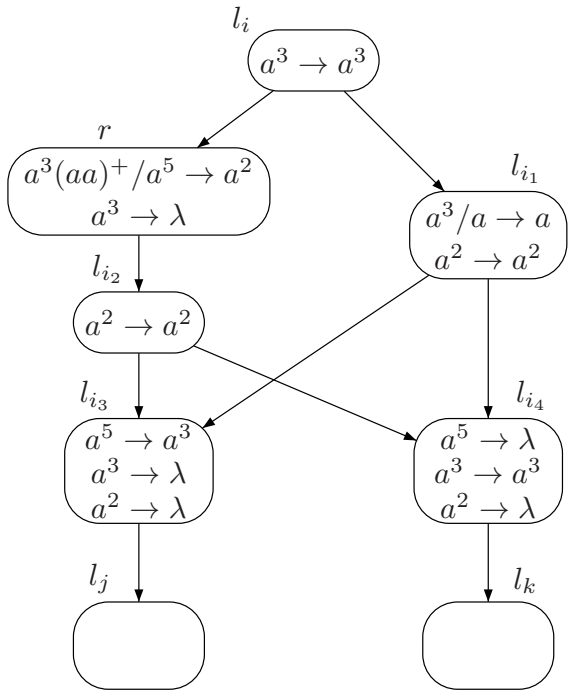

Fig. 3. Module SUB

l_{i_2}, the two spikes arrives at $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$. Together with the three spikes from $\sigma_{l_{i_1}}$, there are both five spikes in $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$. The rule $a^5 \to a^3$ is applied by $\sigma_{l_{i_3}}$, and the rule $a^5 \to \lambda$ is applied by $\sigma_{l_{i_4}}$. In this way, three spikes arrives at σ_{l_j}, but not σ_{l_k}. At this point, σ_{l_j} becomes activated.

On the other hand, if register r stores an empty value, this means that initially there is no spike in σ_r. When the three spikes from σ_{l_i} are available in σ_r, only the rule $a^3 \to \lambda$ can be applied. The three spikes are removed (hence, no spike remains in σ_r) and no spike is produced. As a result, there are both three spikes in $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$. The rule $a^3 \to \lambda$ is applied by $\sigma_{l_{i_3}}$, and the rule $a^3 \to a^3$ is applied by $\sigma_{l_{i_4}}$. In this way, three spikes arrives at σ_{l_k}, but not σ_{l_j}. At this point, σ_{l_k} becomes activated.

We also need to check if there is interference among several SUB modules operating on the same σ_r, i.e. when more than one SUB instruction operates on register r. However, due to the forgetting rule $a^2 \to \lambda$ in neurons l_{i_3} and l_{i_4}, there is no problem or interference. As shown in Fig. 3, each neuron r sends two spikes to all neurons with label l_{i_2}, then to all neurons with label l_{i_3} and l_{i_4} in the SUB module, but all these neurons will forget the two spikes immediately, except for the neurons $\sigma_{l_{i_3}}$ and $\sigma_{l_{i_4}}$ from the module of the SUB instruction whose simulation

proceeds correctly and which also receives three spikes from the corresponding neuron l_{i_1}.

The functioning of the *SUB* module correctly simulates the $l_i : (SUB(r), l_j, l_k)$ operation by either decrementing σ_r and activating σ_{l_j}, otherwise by activating σ_{l_k}. What remains now is to output the result of the computation.

Module OUTPUT: The module for halting the computation and producing the output is given in Fig. 4. Since the output register 1 is never decremented, this means that no *SUB* module operates on σ_1, and the number of spikes in σ_1 is always of the form $2n$. Once neuron l_h becomes activated, it produces a single spike so that σ_1 becomes activated at the next step.

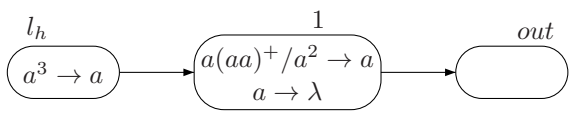

Fig. 4. Module OUTPUT

If register 1 is nonempty, rule $a(aa)^+/a^2 \to a$ in σ_1 is applied since σ_1 now has $2n+1$ spikes. This rule consumes two spikes, and one spike is sent to σ_{out}. The rule will continue to be applied and consume two spikes each step, stopping only when σ_1 has exactly one spike, which is removed by the rule $a \to \lambda$. In this way, the spike train on synapse $(1, out)$ is of the form $aa \ldots a\lambda$, which begins with n number of spikes and is ended with λ, i.e. $n-1$ pairs of consecutive aa, and then one pair of $a\lambda$. For the $n-1$ pairs of consecutive aa, synapse $(1, out)$ will do nothing, so σ_{out} receives one spike for each computing on synapse, i.e. $n-1$ spikes together. For the last pair of $a\lambda$, synapse will do subtraction, so σ_{out} receives one spike. Hence, the spikes stored in σ_{out} is the generated number, i.e. $(n-1)+1 = n$, which is exactly the number stored in output register 1 of M.

We note that all modules make use of at most three rules in each neuron, with any rule consuming at most five spikes, and forgetting at most five spike. Hence, the parameters of the Theorem are satisfied, and this completes the proof.

5 Final Remarks

In this work, we introduced spiking neural P systems with computing synapses (in short, CSSN P systems). Such systems incorporate not only biological inspiration, e.g. the shunting inhibition, but also computational and mathematical inspirations, e.g. dynamic graphs or time-varying networks.

Our result in this work show that CSSN P systems are computationally universal, even with a normal form. The normal form, as given by parameters in the theorem, includes: at most three rules in each neuron, with any rule consuming at most five spikes, and forgetting at most five spike, and all these rules have no delay.

We suspect that the result provided in this work could still be improved, i.e. improve the normal form. It is likely we can reduce $cons_5$ to $cons_4$ and $forg_5$ to $forg_4$. How to reduce these and other parameters in the system, using CSSN P semantics, remains open.

Another open problem is to consider more complicated synapse, e.g. synapse computes by multiplication or division. It is also worth considering computing synapses for SN P systems with anti-spikes.

Acknowledgments.

This work was supported by National Natural Science Foundation of China (61502063) and Chongqing Social Science Planning Project (2017YBGL142).

References

1. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fund. Inform. 71, 279–308 (2006)
2. Cavaliere, M., Ibarra, O.H., Păun, Gh., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous Spiking Neural P Systems. Theor. Comput. Sci. 410, 2352–2364 (2009)
3. Pan, L., Păun, Gh.: Spiking Neural P Systems with Anti-spikes. Int. J. Comput. Commun. IV, 273–282 (2009)
4. Ionescu, M., Păun, Gh., Pérez-Jiménez, M.J., Yokomori, T.: Spiking Neural dP Systems. Fund. Inform. 111, 423–436 (2011)
5. Pan, L., Wang, J., Hoogeboom, H.J.: Spiking Neural P Systems with Astrocytes. Neural. Comput. 24, 805–825 (2012)
6. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking Neural P Systems with Weighted Synapses. Neural. Process. Lett. 35, 13–27 (2012)
7. Song, T., Pan, L., Păun, Gh.: Spiking Neural P Systems with Rules on Synapses. Theor. Comput. Sci. 529, 82–95 (2014)
8. Wang, J., Hoogeboom, H.J., Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Weights. Neural. Comput. 22, 2615–2646 (2014)
9. Song, T., Liu, X., Zeng, X.: Asynchronous Spiking Neural P Systems with Anti-Spikes. Neural. Process. Lett. 42, 633–647 (2015)
10. Song, T., Pan, L.: Spiking Neural P Systems with Rules on Synapses Working in Maximum Spiking Strategy. IEEE. T. Nanobiosci. 14, 465–477 (2015)
11. Song, T., Pan, L.: Spiking Neural P Systems with Rules on Synapses Working in Maximum Spikes Consumption Strategy. IEEE. T. Nanobiosci. 14, 38–44 (2015)
12. Song, T., Gong, F., Liu, X., Zhao, Y., Zhang, X.: Spiking Neural P Systems With White Hole Neurons. IEEE. T. Nanobiosci. 15, 666-673 (2016)
13. Zhao, Y., Liu, X., Wang, W., Adamatzky, A.: Spiking Neural P Systems with Neuron Division and Dissolution. Plos. One. 11, e0162882 (2016)
14. Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like Spiking Neural P Systems. Theor. Comput. Sci. 623, 180–189 (2016)
15. Jiang, K., Chen, W., Zhang, Y., Pan, L.: Spiking Neural P Systems with Homogeneous Neurons and Synapses. Neurocomputing. 171, 1548–1555 (2016)

16. Song, T., Pan, L.: Spiking Neural P Systems with Request Rules. Neurocomputing. 193, 193–200 (2016)
17. Ibarra, O.H., Păun, A., Rodríguez-Patón, A.: Sequential SNP Systems Based on min/max spike number. Theor. Comput. Sci. 410, 2982–2991 (2009)
18. Neary, T.: A Boundary Between Universality and Non-Universality in Extended Spiking Neural P Systems. Lect. Notes. Comput. Sc. 6031, 475–487 (2009)
19. Song, T., Pan, L., Jiang, K., Song, B., Chen, W.: Normal Forms for Some Classes of Sequential Spiking Neural P Systems. IEEE. T. Nanobiosci. 12, 255–264 (2013)
20. Zhang, X., Zeng, X., Luo, B., Pan, L.: On Some Classes of Sequential Spiking Neural P Systems. Neural. Comput. 26, 974–997 (2014)
21. Wang, X., Song, T., Gong, F., Zheng, P.: On the Computational Power of Spiking Neural P Systems with Self-Organization. Sci. Rep. 6: 27624 (2016)
22. Chen, H., Freund, R., Ionescu, M.: On String Languages Generated by Spiking Neural P Systems. Fund. Inform. 75, 141–162 (2007)
23. Krithivasan, K., Metta, V.P., Garg, D.: On String Languages Generated by Spiking Neural P Systems with Anti-spikes. Int. J. Found. Comput. S. 22, 15–27 (2011)
24. Zeng, X., Xu, L., Liu, X.: On String Languages Generated by Spiking Neural P Systems with Weights. Inform. Sciences. 278, 423–433 (2014)
25. Song, T., Xu, J., Pan, L.: On the Universality and Non-Universality of Spiking Neural P Systems with Rules on Synapses. IEEE. T. Nanobiosci. 14, 960–966 (2015)
26. Wu, T., Zhang, Z., Pan, L.: On Languages Generated by Cell-like Spiking Neural P Systems. IEEE. T. Nanobiosci. 15, 455–467 (2016)
27. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X.: Deterministic Solutions to QSAT and Q3SAT by Spiking Neural P Systems with Pre-computed Resources. Theor. Comput. Sci. 411, 2345–2358 (2010)
28. Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Neuron Division and Budding. Sci. China. Inform. Sci. 54, 1596–1607 (2011)
29. Song, T., Zheng, P., Wong, M.L., Wang, X.: Design of Logic Gates Using Spiking Neural P Systems with Homogeneous Neurons and Astrocytes-like Control. Inform. Sciences. 372, 380–391 (2016)
30. Díaz-Pernil, D., Gutiérrez-Naranjo, M.J.: Semantics of Deductive Databases with Spiking Neural P Systems. Neurocomputing. DOI:10.1016/j.neucom.2017.07.007
31. Zeng, X., Song, T., Zhang, X., Pan, L.: Performing Four Basic Arithmetic Operations with Spiking Neural P Systems. IEEE. T. Nanobiosci. 11, 366–374 (2012)
32. Liu, X., Li, Z., Liu, J., Liu, L., Zeng, X.: Implementation of Arithmetic Operations with Time-free Spiking Neural P Systems. IEEE. T. Nanobiosci. 14, 617–624 (2015)
33. Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M.J., Wang, T. Weighted Fuzzy Spiking Neural P Systems. IEEE. T. Fuzzy. Syst. 21, 209–220, (2013)
34. Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy Reasoning Spiking Neural P Systems for Fault Diagnosis. Inform. Sciences. 235, 106–116 (2013)
35. Wang, J., Peng, H.: Adaptive Fuzzy Spiking Neural P Systems for Fuzzy Inference and Learning. Int. J. Comput. Math. 90, 857–868 (2013)
36. Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.J.: Fault Diagnosis of Electric Power Systems Based on Fuzzy Reasoning Spiking Neural P Systems. IEEE. T. Power. Syst. 30, 1182–1194 (2015)
37. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An Optimization Spiking Neural P System for Approximately Solving Combinatorial Optimization Problems. Int. J. Neur. Syst. 24, 1440006 (2014)

38. Mitchell, S.J., Silver, R.J.: Shunting Inhibition Modulates Neuronal Gain during Synaptic Excitation. Neuron. 38, 433–445 (2003)
39. Cabarle, F.G.C., Adorna, H.N., F., Pérez-Jiménez, M.J., Song, T.: Spiking Neural P Systems with Structural Plasticity. Neural. Comput. Appl. 26, 1905–1917 (2015)
40. Song, T., Pan, L.: A Normal Form of Spiking Neural P Systems with Structural Plasticity. Int. J. Swarm Intell. 1, 344–357 (2015)
41. Cabarle, F.G.C., Adorna, H.N., F., Pérez-Jiménez, M.J.: Sequential Spiking Neural P Systems with Structural Plasticity Based on Max/min Spike Number. Neural. Comput. Appl. 27, 1337–1347 (2016)
42. Cabarle, F.G.C., Adorna, H.N., F., Pérez-Jiménez, M.J.: Asynchronous Spiking Neural P Systems with Structural Plasticity. In: Proc. UCNC 2015, pp. 132–143 (2015)