Dynamic threshold neural P systems

Hong Peng [a], Jun Wang [b], Mario J. Pérez-Jiménez [c], Agustín Riscos-Núñez [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:
Membrane computing
P systems
Neural-like P systems
Dynamic threshold neural P systems
Universality

A B S T R A C T

Pulse coupled neural networks (PCNN, for short) are models abstracting the synchronization behavior observed experimentally for the cortical neurons in the visual cortex of a cat's brain, and the intersecting cortical model is a simplified version of the PCNN model. Membrane computing (MC) is a kind computation paradigm abstracted from the structure and functioning of biological cells that provide models working in cell-like mode, neural-like mode and tissue-like mode. Inspired from intersecting cortical model, this paper proposes a new kind of neural-like P systems, called dynamic threshold neural P systems (for short, DTNP systems). DTNP systems can be represented as a directed graph, where nodes are dynamic threshold neurons while arcs denote synaptic connections of these neurons. DTNP systems provide a kind of parallel computing models, they have two data units (feeding input unit and dynamic threshold unit) and the neuron firing mechanism is implemented by using a dynamic threshold mechanism. The Turing universality of DTNP systems as number accepting/generating devices is established. In addition, an universal DTNP system having 109 neurons for computing functions is constructed.

1. Introduction

Membrane Computing is a kind computation paradigm abstracted from the structure and functioning of biological cells. This paradigm provides models working in cell-like mode, based on the hierarchical structure of biological membranes in a cell, tissue-like mode, abstracted from the communication and cooperation of cells in biological tissues, and neural-like mode based on the way that neurons communicate with each other by means of short electrical impulses (spikes) but fired at precise moments of time.

Spiking neural P systems (in short, SNP systems) [1] are a kind of neural-like P systems in membrane computing [2], abstracted from the way neurons work together to deal with and communicate information in a network form of neural cells linked by synapses [3,4]. A SNP system has a directed graph structure, in which the neurons denote the nodes and the arcs denote the synaptic connections between these neurons. There are two kinds of components in each neuron: a data unit that is used to store the number of spikes, and several rules (firing and/or forgetting rules). SNP systems have a working alphabet that is a singleton whose unique element a is called *spike*. A global clock is considered to mark the time of these systems, and all the neurons operate in parallel. SNP systems provide a class of parallel computing models.

In the past years, many variants of SNP systems have been discussed, abstracted from a various of biological facts and/or combined the methods and ideas in computer sciences or mathematics. Considered a couple of anti-spikes $(a, \bar{a})$, SNP systems with anti-spikes were proposed in Pan et al. [5]. With the inspiration of astrocytes with excitatory and inhibitory influences, SNP systems with astrocytes were discussed in [6,7], respectively. In [8,9], SNP systems with rules on synapses have been discussed, in which rules were considered in synapses. At the same time, a competitive spike consumption strategy was discussed in Peng et al. [10]. By introducing weight and threshold mechanism, SNP systems with weights/thresholds have been investigated in [11,12], respectively. Inspired from the biological fact that the synapse has one or more chemical channels, Peng et al. [4] and Song et al. [13] investigated SNP systems with multiple channels. Considered a new communication strategy among neurons, Pan et al. [14] and Wu et al. [15] discussed SNP systems with communication on request. Motivated by the biological phenomena that each neuron has a charge either positive or negative, SNP systems with polarizations were discussed [16]. Under the restriction that at most one neuron can be applied at each time, Ibarra [17] and Zhang [18] discussed sequential SNP systems, respectively. In addition, Chen et al. [19] introduced axon P systems, in which nodes are organized in a linear structure. Due to the use of a global clock, a lot of SNP systems work synchronously, however, a number of asynchronous SNP systems were investigated [20,21]. In recent years, fuzzy logic has been integrated in SNP systems, called fuzzy SNP systems, such as, fuzzy reasoning SNP systems [22], weighted fuzzy SNP systems [23] and adaptive fuzzy SNP systems [24]. In addition, SNP systems have been applied to process a lot of real-world problems, for instance, fault diagnosis [25–28], image processing [29], data clustering [30–32] and combinatorial optimization problems [33]. The computational properties of SNP systems were investigated. Numerous variants of SNP systems, as number accepting/generating devices [1,6,20,21], language generating devices [34,35] and function computing device [36,37], have been proven to be Turing universal. SNP systems have been used to deal with some computationally hard problems in a linear or polynomial time [38,39].

Biological systems have always been an inspiration for developing algorithms. Cat's and guinea pig's visual cortexes have helped in developing some digital models. Eckhorn et al. [40,41] created mammal's neuron model by investigating the synchronous oscillating phenomenon of neurons in cat's visual cortex. Eckhorn's neuron model has been used to develop pulse coupled neural networks [42]. In Eckhorn's model, each neuron has three components (receptive field, nonlinear modulation and output module), and the receptive field consists of linking input and feeding input [43]. Kinser [44] proposed the intersecting cortical model (ICM) that is a modification of Eckhorn's neuron model [45]. In ICM model, feeding input and nonlinear modulation mechanism in Eckhorn's model are removed, and a dynamic threshold mechanism is introduced where the threshold can be changed according to the neurons's firing. From the perspective of application, regular-expression-based firing mechanism in SNP systems maybe suffer from a limitation when they are directly used to solve real-world problems: it is only suitable to the kind of some applications related to regular language. Therefore, it becomes an interesting and challenging problem how to develop some extensions to break the limitation. Compared with spiking neuron used in SNP systems, the ICM model can provide a stronger control ability for behavior of neurons. This paper focuses on how to develop a new neural-like model based on the ICM model under membrane computing framework, which is different from the existing SNP system in the mechanism.

This paper investigates a new variant of neural-like P systems, called dynamic threshold neural P systems (in short, DTNP systems), abstracted from the intersecting cortical model (for short, ICM model) that was derived from the synchronous oscillating phenomenon of neurons in cat's visual cortex [40]. A DTNP system contains a number of dynamic threshold neurons, and each dynamic threshold neuron has two data units and dynamic threshold mechanism used to process the neuron' firing. For convenience, we define DTNP systems by using the notations and terms as in SNP systems. The computational power of DTNP systems as number generating/accepting devices and function computing device is discussed. Specifically, DTNP systems as number generating/accepting devices which use an unbounded number of neurons and where each neuron has at most two rules, are Turing universal. Moreover, an universal DTNP system having 109 neurons for computing functions is established. From the point of view of working mechanism, the differences between DTNP systems and SNP systems can be summarized as follows:

(1) SNP systems are inspired from a simplified model of spiking neurons, while DTNP systems use dynamic threshold neurons, abstract from ICM model of the cortical neurons.
(2) SNP systems use a regular-expression-based firing mechanism, however, DTNP systems adopt a dynamic-threshold-based firing mechanism, which is often expressed by a conjunctive form.
(3) Each dynamic threshold neuron in DTNP systems has two data units (feeding input unit and dynamic threshold unit), and maximum spike consumption strategy and non-deterministic rule selection strategy are used to deal with the neuron's spiking.

The motivation of this paper stays in introducing a new dynamic threshold neuron, inspired from ICM model of the cortical neurons, and proposing a new variant of neural-like P systems based on the dynamic threshold neuron, i.e., DTNP systems. In addition, SNP systems can suffer from a certain limitation in directly solving real-world problems because they use a regular-language-based firing condition, for example, image and signal processing problems. However, the limitation is relaxed in DTNP systems in a conjunctive form, and a firing mechanism with dynamic threshold is used to handle the firing of neurons. It is well-known that pulse coupled neural networks as an analog of DTNP systems have been successfully used to solve image and signal processing problems. Therefore, the new variant seems to be suitable for solving those real-world problems. However, this work mainly focuses on computational power of DTNP systems as number generating/accepting and function computing devices.

The rest of this paper is organized as follows. The proposed DTNP systems are presented in Section 2. The universality of DTNP systems as number accepting/generating devices is proven in Section 3. The universality of DTNP systems as function computing device is investigated in Section 4. Section 5 draws conclusions and discussion.

2. DTNP systems

In this section, we describe in detail the proposed DTNP systems, which are inspired from the ICM model. For clarity and ease of understanding, we use notations and terms similar to those used in SNP systems. It is well known that in SNP systems, each neuron has a data unit (storing the number of spikes in the neuron) and some rules (characterizing the behavior of the system). However, ICM model has two kinds of data (feeding input and dynamic threshold) and a dynamic threshold mechanism where the threshold can be changed with the neurons's firing. Therefore, when designing the DTNP systems, two integrants that are different from SNP systems are considered:

(1) each dynamic threshold neuron has two data units: feeding input unit and dynamic threshold unit.
(2) the neuron's firing is based on a dynamic threshold mechanism instead of regular-expression-based or static-threshold-based firing rules.

As a result, the dynamic threshold mechanism can provides a stronger control ability for behavior of DTNP systems.

2.1. Definition

Definition 1. A DTNP system with degree $m \geq 1$ is formulated as a tuple:

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

where:

(1) $O = \{a\}$ is the singleton alphabet (a is called the spike);

(2) $\sigma_i = (u_i, \tau_i, R_i)$, $1 \leq i \leq m$, are neurons with dynamic threshold, where:

  (a) $u_i \geq 0$ is the number of spikes as feeding input via dendrites in σ_i;
  (b) $\tau_i \geq 0$ is the (dynamic) threshold in σ_i;
  (c) R_i denotes the finite set of firing rules of the form $E_i/(a^u, a^\tau) \to a^p$, in which $E_i = \{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau\}$ is a firing condition, $u \geq 1, \tau \geq 0, p \geq 0$ and $p \leq u$; In the case of $p \geq 1$, the rule is called a *firing rule*, and in the case of $p = 0$, the rule is called a *forgetting rule* (a^0 is denoted by λ);

(3) $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$ indicates a directed graph structure of synapses between neurons; note that $(i, i) \notin syn$, $1 \leq i \leq m$;
(4) $in, out \in \{\sigma_1, \ldots, \sigma_m\}$ denote the input neuron and output neuron of Π, respectively.

A DTNP system is regarded as a directed graph without self-loops (arcs from a node to itself), where the nodes of the graph are represented by neurons while the arcs denote synaptic connections between these neurons. A virtual node that represents the environment can be considered in this graph and there exist two distinguished neurons (the input neuron and the output neuron) that allow to the system communicate with its environment. Specifically, the output neuron provides an "outgoing arc" in such manner that each step that output neuron fires, a spike is transmitted into the environment of DTNP system. The input neuron provides an "ingoing arc" in such manner that a natural number n can be encoded in the system when two spikes are imported in the input neuron during n steps.

Apart from directed graph structure, there are two integrants in a DTNP system: data and rules. Data is used to describe the state of each neuron, and the rules characterize the dynamic behavior of the system. Different from SNP systems, each neuron σ_i in DTNP systems has two data units: a feeding input unit, denoted by u_i, and a dynamic threshold unit, denoted by τ_i, that is, $u_i(t)$ and $\tau_i(t)$ denote respectively the numbers of spikes in feeding input unit and dynamic threshold unit at time t. Assume $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out)$ is a DTNP system of degree m. A *configuration* C_t of Π at time t is denoted by a vector $(u_1(t), \tau_1(t), \ldots, u_m(t), \tau_m(t))$, which is an instantaneous description of the system, in which for each $1 \leq i \leq m$, $u_i(t)$ indicates the number of spikes contained in feeding input unit of neuron σ_i at time t and $\tau_i(t)$ is the dynamic threshold of neuron σ_i at time t. The *initial configuration* of Π is $(u_1, \tau_1, \ldots, u_m, \tau_m)$, that is, $u_i(0) = u_i$ and $\tau_i(0) = \tau_i$, for each $1 \leq i \leq m$.

DTNP systems have the rules of two types: firing rules and forgetting rules. Each neuron σ_i is associated a finite set of (firing and/or forgetting) rules of the type $E_i/(a^u, a^\tau) \to a^p$. Note that the expression $E_i = \{u_i \geq \tau_i \wedge u_i \geq u \wedge \tau_i \geq \tau\}$ can be omitted because it is injectively associated with σ_i. The semantics of the rules in DTNP systems can be illustrated as follows. The rule $r \equiv E_i/(a^u, a^\tau) \to a^p$ in neuron σ_i is applicable to a configuration C_t at time $t \geq 0$ of Π if $u_i(t) \geq \tau_i(t)$, $u_i(t) \geq u$ and $\tau_i(t) \geq \tau$. When rule $r \equiv E_i/(a^u, a^\tau) \to a^p$ is applicable to a configuration C_t we say that neuron σ_i is enabled at time t. When rule $E_i/(a^u, a^\tau) \to a^p$ is applied to a configuration C_t, neuron σ_i fires, consuming u spikes in feeding input unit and τ spikes in dynamic threshold unit. Besides, if $p \geq 1$ then it generates p spikes that are transmitted to each postsynaptic neurons σ_j where $(i, j) \in syn$. Thus, by applying the rule $E_i/(a^u, a^\tau) \to a^p$ to a configuration $C_t = (u_1(t), \tau_1(t), \ldots, u_m(t), \tau_m(t))$, we have

$$\begin{cases} u_i(t+1) = u_i(t) - u + n, \\ \tau_i(t+1) = \tau_i(t) - \tau + p. \end{cases}$$

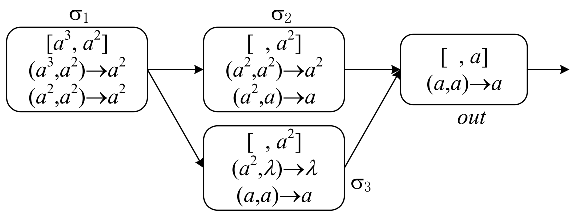


Fig. 1. An example of DTNP systems.

where n denotes the number of spikes received from other neurons, and p denotes the number of spikes generated by the neuron.

Two rules $r \equiv E_i/(a^u, a^\tau) \to a^p$ and $r' \equiv E_i/(a^{u'}, a^{\tau'}) \to a^{p'}$ in neuron σ_i can be applicable to a configuration C_t. In this case, only one of them will be applied to C_t according to the following criterion (*maximum spike consumption strategy is adopted*): if $u > u'$ (resp. $u < u'$) then rule r (resp. rule r') is chosen to apply (in particular, is possible to apply a forgetting rules instead of an applicable firing rule); if $u = u'$ then one of them (r or r') is non-deterministically selected.

For a configuration C_t, if no rule of the system can be applied, we say that it is a *halting configuration*. If C_t is a non-halting configuration then we use $C_t \Rightarrow_\Pi C_{t+1}$ to denote one *transition step* from configuration C_t to configuration C_{t+1}. In a DTNP system, the rules are sequentially applied for each neuron, however, neurons function in parallel.

DTNP systems can be considered working in *generating mode* or in *accepting mode*. In the case of DTNP systems as generating device, a *spike train* is associated with any computation $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_t \Rightarrow_\Pi C_{t+1} \ldots$ as follows: it is the binary sequence such that at position t is written 1 if the output neuron emits some spike at time t, otherwise a 0 is written. Let $ST(\Pi)$ denote the collection of all spike trains (over all computations) of Π. $ST(\Pi)$ can be associated with a set of numbers in several ways, in this paper, the computation result is considered as the number of steps between the first two consecutive spikes emitted to the environment. $N_2(\Pi)$ is used to express the set of numbers computed by Π, and $N_2DTNP_m^n$ denotes the families of all sets $N_2(\Pi)$ computed by DTNP systems having at most m neurons and at most n rules in each neuron. As usual, parameter m or n is replaced with $*$ if it is not bounded. In the case of generating mode, the role of the input neuron is irrelevant and it can be ignored.

For the case of DTNP systems as accepting device, a distinguished neuron is regarded as the *input neuron* and two spikes are imported to the neuron during k steps. Then, the number k is accepted if the computation halts. $N_{acc}(\Pi)$ is used to express the set of numbers accepted by Π and $N_{acc}DTNP_m^n$ denotes the families of all sets $N_{acc}(\Pi)$ accepted by DTNP systems having at most m neurons and at most n rules in each neuron. For accepting mode, the role of the output neuron is irrelevant and it can be ignored.

In the following, DTNP systems as number accepting/generating devices are proved to be computationally complete (equivalent with Turing machines).

2.2. An illustrative example

To understand the mechanism of DTNP systems, we here give an example working in generating mode that can generate the finite spike train.

Let us suppose that Π is a DTNP system, consisting of four neurons $\sigma_1, \sigma_2, \sigma_3$ and *out*, shown in Fig. 1. Initially, neurons σ_2, σ_3 and *out* have no spike in their feeding input units, while neuron σ_1 has three spikes in its feeding input unit, that is, $u_1(0) = 3$, $u_2(0) = 0$, $u_3(0) = 0$, $u_{out}(0) = 0$. The initial thresholds of the four neurons are set: $\tau_1(0) = 2, \tau_2(0) = 2, \tau_3(0) = 2, \tau_{out}(0) = 1$. Thus, $C_0 = (3, 2, 0, 2, 0, 2, 0, 1)$.

At time 1, since $u_1(0) = 3 \geq \tau_1(0) = 2$, rules $(a^3, a^2) \to a^2$ and $(a^2, a^2) \to a^2$ can be applied. Nevertheless, rule $(a^3, a^2) \to a^2$ (with $u = 3, \tau = p = 2$) will be applied in neuron σ_1 according to maximum spike consumption strategy, and then it sends two spikes to neurons σ_2 and σ_3. Thus, $u_1(1) = 0, \tau_1(1) = 2, u_2(1) = 2, \tau_2(1) = 2, u_3(1) = 2, \tau_3(1) = 2, u_{out}(1) = 0, \tau_{out}(1) = 1$. Therefore, $C_1 = (0, 2, 2, 2, 2, 2, 0, 1)$.

At time 2, since $u_3(1) = 2 \geq \tau_3(1) = 2$, rule $(a, a) \to a$ and forgetting rule $(a^2, \lambda) \to \lambda$ can be enabled, but, according to maximum spike consumption strategy, rule $(a^2, \lambda) \to \lambda$ will be applied in neuron σ_3, consuming only two spikes in feeding input unit of that neuron. With respect to neuron σ_2, rules $(a^2, a^2) \to a^2$ (with $u = \tau = p = 2$) and $(a^2, a) \to a$ (with $u' = 2, \tau' = p' = 1$) can be applied because $u_2(1) \geq \tau_2(1), u_2(1) \geq u, \tau_2(1) \geq \tau, u_2(1) \geq u', \tau_2(1) \geq \tau'$. Thus, one of them is selected non-deterministically, and two cases are considered as follows:

(1) *Case 1: rule $(a^2, a^2) \to a^2$ is applied in neuron σ_2 at time 2.* In this case, neuron σ_2 consumes two spikes in each of its two data units and sends two spikes to neuron *out*. Then, we have $u_{out}(2) = 2, \tau_{out}(2) = 1$. Thus, $C_2 = (0, 2, 0, 2, 0, 2, 2, 1)$. At time 3, since $u_{out}(2) \geq \tau_{out}(2)$, rule $(a, a) \to a$ is enable and sends one spike into the environment. Then, we have $u_{out}(3) = 1, \tau_{out}(3) = 1$, and $C_3 = (0, 2, 0, 2, 0, 2, 1, 1)$. At time 4, since $u_{out}(3) \geq \tau_{out}(3)$, rule $(a, a) \to a$ is applied again and sends the second spike into the environment. Then, we have $u_{out}(4) = 0, \tau_{out}(4) = 1$ and no rule is applicable to any neuron. Thus, the generated spike train is "0011", and $C_4 = (0, 2, 0, 2, 0, 2, 0, 1)$.

(2) *Case 2: rule $(a^2, a) \to a$ is applied in neuron σ_2 at time 2.* In this case, neuron σ_2 consumes two spikes in its feeding input unit and transmits one spike into neuron *out*. Then, we have $u_{out}(2) = 1, \tau_{out}(2) = 1$. Thus, $C_2 = (0, 2, 0, 2, 0, 2, 1, 1)$. At time 3, with one spike in neuron *out*, rule $(a, a) \to a$ is applied to transmit one spike into the environment. Then, we have $u_{out}(3) = 0, \tau_{out}(3) = 1$ and no rule is applicable to any neuron. Thus, $C_3 = (0, 2, 0, 2, 0, 2, 0, 1)$. The system halts, and the generated spike train is "001".

3. Turing universality of DTNP systems as number-generating/accepting device

In this section, computational completeness of DTNP systems as number accepting/generating devices is discussed. The universality of DTNP systems is proven by simulating register machines. Specifically, we showed that DTNP systems can accept/generate all recursively enumerable sets of numbers, which is usually denoted by *NRE*.

Let us recall that a *register machine* can be denoted by $M = (m, H, l_0, l_h, I)$, in which m denotes the number of registers; H denotes the set of instruction labels; I denotes the set of instructions bijectively labeled by elements of H; $l_0 \in H$ indicates the starting label, while $l_h \in H$ is the halting label and corresponds to instruction *HALT*. There are instructions of the following three forms in I:

(1) $l_i : (ADD(r), l_j, l_k), l_i \in H \setminus \{l_h\}, l_j, l_k \in H, 1 \leq r \leq m$ (add 1 to register r and then go non-deterministically to one of the instructions with labels l_j or l_k).
(2) $l_i : (SUB(r), l_j, l_k), l_i \in H \setminus \{l_h\}, l_j, l_k \in H, 1 \leq r \leq m$ (if register r is non-zero, then decrease the value of register r by one and go to the instruction with label l_j; otherwise go to the instruction with label l_k).
(3) $l_h : HALT$ (halting instruction).

It is worth pointing out that the following convention will be considered throughout this paper: when comparing or evaluating the computation power of both number accepting/generating devices, the number 0 is ignored; this is associated with a frequently made convention in automata and grammars theory, in which the empty string λ is omitted when evaluating two language accepting/generating devices.

3.1. Turing universality of systems working in the generating mode

For the generating mode, a register machine can compute a number n: Initially, all registers are set to be empty; starting from the instruction l_0, the register machine continuously applies instructions in I; if it arrives the halting instruction, then the value contained in the first register is called as the number computed by register machines. It has been proven that the family *NRE* can be computed by register machines. We first give the universal result of DTNP systems as number generating device as follows.

Theorem 1. $N_2DTNP_*^2 = NRE$

Proof. It is obvious that $N_2DTNP_*^2 \subseteq NRE$ due to the Turing-Church thesis and the universality of Turing machines. So it suffices to prove $NRE \subseteq N_2DTNP_*^2$ and we will use the characterization of the family *NRE* by means of register machines working in the generative mode. For this purpose, let $M = (m, H, l_0, l_h, I)$ be a register machine. In general, suppose that all registers except for register 1 are empty, and that during the computation register 1 is never decreased.

To simulate the register machine M, a DTNP system Π is constructed, including the modules of three types: an FIN module that provides the computation result, shown in Fig. 4, and SUB and ADD modules that are applied to simulate the SUB and ADD instructions, shown in Fig. 2 and Fig. 3 respectively.

Assume that each register r of M corresponds to a neuron σ_r in Π that has no rule. The number contained in register r is encoded: if register r contains the number $n \geq 0$, then neuron σ_r stores $2n$ spikes in its feeding input unit. Each instruction l corresponds to a neuron σ_l, and in modules we design some auxiliary neurons. Initially, neuron σ_{l_0} contains two spikes in its feeding input unit, and all auxiliary neurons have no spike. Assume that each neuron contains a certain initial threshold: (i) each neuron that corresponds to an instruction has initial threshold a^2; (ii) each register contains initial threshold a^3; (iii) initial thresholds of other neurons are set to a, a^2 or a^3. Since neuron σ_{l_i} has two spikes, the system Π begins to simulate the instruction $l_i : (OP(r), l_j, l_k)$ (*OP* denotes operation ADD or operation SUB): starting from the activated neuron σ_{l_i}, the simulation deals with neuron σ_r as indicated by OP, and then two spikes are introduced into one of neurons σ_{l_j} and σ_{l_k}. Once neuron σ_{l_h} is activated and the FIN module is executed, this indicates that the simulation of M is completed. During the computation, output neuron σ_{out} applies its rules to send the spikes into the environment twice at times t_1 and t_2 respectively, and the interval $t_2 - t_1$ that is associated with the number contained in register 1, is considered as the computation result. Note that if any rule $(a^u, a^\tau) \to a^p$ in any module satisfies that $\tau = p$, then the number of spikes in dynamic threshold units remains unchanged (as already explained in Section 2).

To prove that the register machine M can be correctly simulated by the system Π, we will illustrate how do SUB and ADD modules simulate the SUB and ADD instructions respectively, and how the computation result is exported by the FIN module.

(1) Module ADD (shown in Fig. 2) - simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$.

The DTNP system Π begins from simulation of instruction l_0, which is an ADD instruction. Suppose that at time t, an ADD

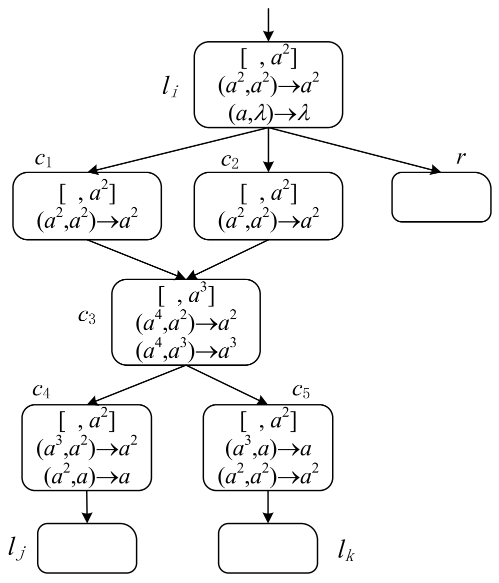


Fig. 2. Module ADD (simulating $l_i : (ADD(r), l_j, l_k)$).

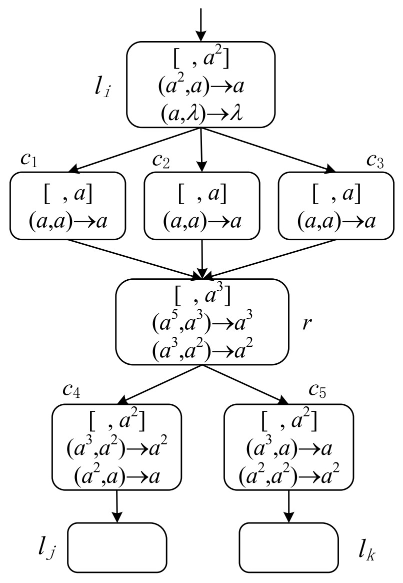


Fig. 3. Module SUB (simulating $l_i : (SUB(r), l_j, l_k)$).

instruction $l_i : (ADD(r), l_j, l_k)$ is executed in M. Let us see how that instruction is simulated by the DTNP system constructed. Assume that $C_t = (n_1, \tau_1, n_2, \tau_2, \ldots, n_8, \tau_8)$ is the configuration of module ADD at time t, which is associated to eight neurons $\sigma_{l_i}, \sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}, \sigma_{c_4}, \sigma_{c_5}, \sigma_{l_j}$ and σ_{l_k}, respectively. Thus, $C_t = (2, 2, 0, 2, 0, 2, 0, 3, 0, 2, 0, 2, 0, 2, 0, 2)$. Since neuron σ_{l_i} has received two spikes, its feeding input unit has two spikes, which is equal to the number of spikes in dynamic threshold unit. Therefore, rule $(a^2, a^2) \to a^2$ is enabled and then two spikes are transmitted to neurons $\sigma_{c_{i,1}}$, $\sigma_{c_{i,2}}$ and σ_r. The fact that neuron σ_r receives two spikes means that register r is increased by 1. Thus, $C_{t+1} = (0, 2, 2, 2, 2, 2, 0, 3, 0, 2, 0, 2, 0, 2, 0, 2)$.

At time $t + 1$, with two spikes in neurons $\sigma_{c_{i,1}}$ and $\sigma_{c_{i,2}}$, the rule $(a^2, a^2) \to a^2$ is applied and each of them sends two spikes to neuron $\sigma_{c_{i,3}}$. Thus, neuron $\sigma_{c_{i,3}}$ receives four spikes. Therefore, $C_{t+2} = (0, 2, 0, 2, 0, 2, 4, 3, 0, 2, 0, 2, 0, 2, 0, 2)$. At time $t + 2$, with four spikes in neurons $\sigma_{c_{i,3}}$, two rules $(a^4, a^2) \to a^2$ and $(a^4, a^3) \to a^3$ can be applied to neuron $\sigma_{c_{i,3}}$. Since the two rules will consume the same number of spikes in feeding input unit, one of them is selected non-deterministically. Two cases are considered as follows:

(i) At time $t + 2$, if rule $(a^4, a^3) \to a^3$ is applied, neuron $\sigma_{c_{i,3}}$ sends three spikes to neurons $\sigma_{c_{i,4}}$ and $\sigma_{c_{i,5}}$, respectively. Thus, $C_{t+3} = (0, 2, 0, 2, 0, 2, 0, 3, 3, 2, 3, 2, 0, 2, 0, 2)$. At time $t + 3$, with three spikes in $\sigma_{c_{i,4}}$, rule $(a^3, a^2) \to a^2$ is applied, then two spikes are sent to neuron σ_{l_j}. Thus, neuron σ_{l_j} receives two spikes, meaning that system Π begins to simulate the instruction l_j of M. At the same time, since rule $(a^3, a) \to a$ in neuron $\sigma_{c_{i,5}}$ can be applied, and it transmits one spike to neuron σ_{l_k}. But the spike in neuron σ_{l_k} will be removed by rule $(a, \lambda) \to \lambda$ in neuron σ_{l_k}. Therefore, $C_{t+4} = (0, 2, 0, 2, 0, 2, 0, 3, 0, 2, 0, 2, 2, 2, 0, 2)$.

(ii) At time $t + 2$, if rule $(a^4, a^2) \to a^2$ is applied, neuron $\sigma_{c_{i,3}}$ transmits two spikes to neurons $\sigma_{c_{i,4}}$ and $\sigma_{c_{i,5}}$, respectively. Thus, $C_{t+3} = (0, 2, 0, 2, 0, 2, 0, 3, 2, 2, 2, 2, 0, 2, 0, 2)$. At time $t + 3$, with two spikes in $\sigma_{c_{i,4}}$, rule $(a^2, a) \to a$ is enabled and one spike is transmitted to neuron σ_{l_j}, but the spike will be consumed by rule $(a, \lambda) \to \lambda$ in neuron σ_{l_j}. At the same time $t + 3$, with two spikes in $\sigma_{c_{i,5}}$, rule $(a^2, a^2) \to a^2$ is applied, then two spikes are transmitted to neuron σ_{l_k}, and with two spikes in neuron σ_{l_k}, the system Π begins to simulate the instruction l_k of M. Therefore, $C_{t+4} = (0, 2, 0, 2, 0, 2, 0, 3, 0, 2, 0, 2, 0, 2, 2, 2)$.

Consequently, ADD instruction can correctly simulated by ADD module: from neuron σ_{l_i} receiving two spikes, the number of spikes in neuron σ_r is increased by two, and one of neurons σ_{l_j} and σ_{l_k} is selected non-deterministically.

(2) Module SUB (shown in Fig. 3) - simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$.

Assume that in the register machine M a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is executed at time t. Let us see how that instruction is simulated by the DTPN system constructed. Assume that $C'_t = (n_1, \tau_1, n_2, \tau_2, \ldots, n_9, \tau_9)$ is the configuration of module SUB at time t, which is associated to eight neurons $\sigma_{l_i}, \sigma_{c_1}, \sigma_{c_2}, \sigma_{c_3}, \sigma_r, \sigma_{c_4}, \sigma_{c_5}, \sigma_{l_j}$ and σ_{l_k}, respectively. Thus, $C'_t = (2, 2, 0, 1, 0, 1, 0, 1, 2n, 3, 0, 2, 0, 2, 0, 2, 0, 2)$. Since neuron σ_{l_i} receives two spikes at time t, rule $(a^2, a) \to a$ is enabled, and it transmits one spike to neurons $\sigma_{c_{i,1}}$, $\sigma_{c_{i,2}}$ and $\sigma_{c_{i,3}}$. Therefore, $C'_{t+1} = (0, 2, 1, 1, 1, 1, 1, 1, 2n, 3, 0, 2, 0, 2, 0, 2, 0, 2)$. At time $t + 1$, each of neurons $\sigma_{c_{i,1}}, \sigma_{c_{i,2}}$ and $\sigma_{c_{i,3}}$ transmits one spike to neuron σ_r by rule $(a, a) \to a$. Neuron σ_r receives three spikes. Thus, $C'_{t+2} = (0, 2, 0, 1, 0, 1, 0, 1, 2n + 3, 3, 0, 2, 0, 2, 0, 2, 0, 2)$. According to the number of spikes in σ_r, two cases are considered as follows:

(i) If at time t the contents of register r in M is $n > 0$ then at time $t + 2$, neuron σ_r contains has $2n + 3$ spikes (in this case, $2n + 3 \geq 5$). Then, rule $(a^5, a^3) \to a^3$ will be applied in neuron σ_r, so three spikes are sent to neurons $\sigma_{c_{i,4}}$ and $\sigma_{c_{i,5}}$ respectively. Thus, $C'_{t+3} = (0, 2, 0, 1, 0, 1, 0, 1, 2n - 2, 3, 3, 2, 3, 2, 0, 2, 0, 2)$. At time $t + 3$, with three spikes in $\sigma_{c_{i,4}}$, rule $(a^3, a^2) \to a^2$ is applied, and then two spikes are transmitted to neuron σ_{l_j}. Therefore, neuron σ_{l_j} receives two spikes, meaning that the system Π begins to simulate the instruction l_j of M. At the same time, since rule $(a^3, a) \to a$ in neuron $\sigma_{c_{i,5}}$ can be applied, it transmits one spike to neuron σ_{l_k}. However, this spike will be consumed by rule $(a, \lambda) \to \lambda$ at time $t + 4$. Therefore, $C'_{t+4} = (0, 2, 0, 1, 0, 1, 0, 1, 2n - 2, 3, 0, 2, 0, 2, 2, 2, 0, 2)$.

(ii) If at time t the contents of register r in M is 0 then at time $t + 2$, neuron σ_r contains has 3 spikes. Then only rule $(a^3, a^2) \to a^2$ is applicable in neuron σ_r. When applying such a rule, two spikes are transmitted to neurons $\sigma_{c_{i,4}}$ and

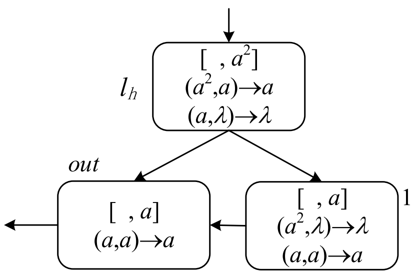


Fig. 4. Module FIN (ending the computation).

$\sigma_{c_{i,5}}$, respectively. Thus, $C'_{t+3} = (0, 2, 0, 1, 0, 1, 0, 1, 0, 3, 2, 2, 2, 2, 0, 2, 0, 2)$. At time $t + 3$, with two spikes in $\sigma_{c_{i,4}}$, rule $(a^2, a) \rightarrow a$ is enabled and one spike is transmitted to neuron σ_{l_j}, but the spike will be removed by rule $(a, \lambda) \rightarrow \lambda$ in neuron σ_{l_j}, at the next step. Moreover, with two spikes in $\sigma_{c_{i,5}}$, rule $(a^2, a^2) \rightarrow a^2$ is applied, and then two spikes are sent to neuron σ_{l_k}. Thus, $C'_{t+4} = (0, 2, 0, 1, 0, 1, 0, 1, 0, 3, 0, 2, 0, 2, 0, 2, 2, 2)$. Moreover, neuron σ_{l_k} receives two spikes. Since neuron σ_{l_k} has two spikes, the system Π begins to simulate the instruction l_k.

Consequently, SUB instruction can be correctly simulated by the SUB module: the system begins with two spikes in neuron σ_{l_i}, and ends with σ_{l_j} receiving two spikes (if the number contained in register r is greater than 0) or with σ_{l_k} receiving two spikes (if the number contained in register r is 0).

(3) Module FIN (shown in Fig. 4) - outputting the result of computation.

Let $C''_t = (n_1, \tau_1, n_2, \tau_2, n_3, \tau_3)$ be the configuration of module FIN at time t, which is associated to three neurons σ_{l_i}, σ_1 and σ_{out}, respectively. Suppose that at time t the computation in M halts and the result is $n > 0$, i.e., the halting instruction l_h arrives and register 1 has natural number n. Then, at that time t, neuron σ_{l_h} receives two spikes and neuron σ_1 contains $2n$ spikes. Thus, $C''_t = (2, 2, 2n, 1, 0, 1)$. Since neuron σ_{l_h} has two spikes, rule $(a^2, a) \rightarrow a$ is enabled and one spike is transmitted to neurons σ_1 and σ_{out}. Simultaneously, in neuron σ_1 rule $(a^2, \lambda) \rightarrow \lambda$ is applied. Thus, at configuration C''_{t+1}, neuron σ_1 contains $2n - 1$ spikes and neuron σ_{out} contains one spike, i.e., $C''_{t+1} = (0, 2, 2n - 1, 1, 1, 1)$.

Now, two cases are considered. If $n = 1$ then in neuron σ_1 rule $(a, a) \rightarrow a$ is applied and in neuron σ_{out} rule $(a, a) \rightarrow a$ is applied. Thus, $C''_{t+2} = (0, 2, 0, 1, 1, 1)$. Consequently, at configuration C''_{t+2} neuron σ_1 contains no spikes, neuron σ_{out} contains one spike and one spike has been transmitted to the environment. Then in neuron σ_1 no rule is applied and in neuron σ_{out} rule $(a, a) \rightarrow a$ is applied. $C''_{t+3} = (0, 2, 0, 1, 0, 1)$. Therefore, at configuration C''_{t+3} neurons σ_1 and σ_{out} contain no spikes, and one spike has been transmitted to the environment. Hence, in the case $n = 1$ the DTNP system halts and the result is $(t + 3) - (t + 2) = 1$.

If $n > 1$ then in neuron σ_1 rule $(a^2, \lambda) \rightarrow \lambda$ is applied and in neuron σ_{out} rule $(a, a) \rightarrow a$ is applied. $C''_{t+2} = (0, 2, 2n - 2, 1, 1, 1)$. Consequently, at configuration C''_{t+2} neuron σ_1 contains $2n - 3$ spikes, neuron σ_{out} contains no spike and one spike has been transmitted to the environment. Then, in neuron σ_1 rule $(a^2, \lambda) \rightarrow \lambda$ is applied and in neuron σ_{out} no rule is applied. $C''_{t+3} = (0, 2, 2n - 3, 1, 1, 1)$. Therefore, at configuration C''_{t+3} neuron σ_1 contains $2n-5$ spikes and neuron σ_{out} contains no spike. Repeating the reasoning, at configuration $C''_{t+n} = (0, 2, 1, 1, 0, 1)$ neuron σ_1 contains one spike and neuron σ_{out} contains no spike, at configuration $C''_{t+n+1} = (0, 2, 0, 1, 1, 1)$ neuron σ_1 contains no spike and neuron σ_{out} contains one spike. At configuration $C''_{t+n+2} = (0, 2, 0, 1, 0, 1)$ neurons σ_1 and σ_{out} contain no spikes, one spike has been transmitted to the environment, and the system halts. Hence, in the case $n > 1$, the result of the system is $n = (t + n + 2) - (t + 2)$, which is exactly the number contained in register 1 while the computation of M halts.

Based on the above analysis, it can be found that the register machine as number generating device is correctly simulated by DTNP system, in which each neuron has at most two rules. Therefore, the theorem holds. □

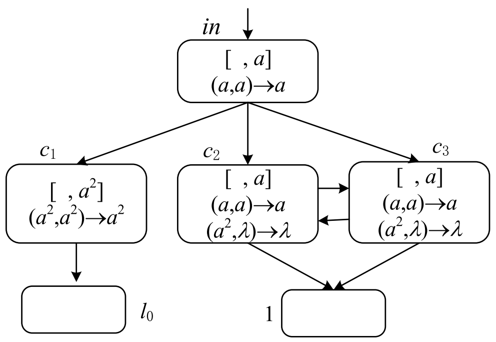


Fig. 5. The INPUT Module of Π'.

3.2. *Turing universality of systems working in the accepting mode*

In the following, we prove the universal result of DTNP systems as number accepting device.

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

Proof. A DTNP system Π' is constructed to simulate a deterministic register machine that works in accepting mode, $M = (m, H, l_0, l_h, I)$. The proof of Theorem 1 is modified to illustrate the following proof. The DTNP system Π' contains an SUB module, a deterministic ADD module and an INPUT module. Initially, no spike is contained in any of the auxiliary neurons. For initial thresholds in neurons, the same setting in the proof of Theorem 1 is used.

The INPUT module is shown in Fig. 5. Neuron σ_{in} is used to read spike train $10^{n-1}1$ from the environment. Note that the interval between the two spikes in the spike train, $n = (n+1) - 1$, indicates the number accepted by the system.

Suppose that at time t, the first spike is imported into neuron σ_{in} from the environment. With one spike in neuron σ_{in}, one spike is transmitted into neurons σ_{c_1}, σ_{c_2} and σ_{c_3} by rule $(a, a) \rightarrow a$. At time $t + 1$, neurons σ_{c_2} and σ_{c_3} each transmits one spike to neuron σ_1. Thus, neuron σ_1 receives two spikes. Note that with only one spike in neuron σ_{c_1}, rule $(a^2, a^2) \rightarrow a^2$ cannot be applied.

From time $t + 1$, until the second spike entering neurons σ_{in}, one spike is transmitted into neuron σ_1 by neurons σ_{c_2} and σ_{c_3} respectively. For neuron σ_1, since $2n$ spikes are received in total from time $t + 2$ to time $t + n + 1$, this indicates that the number contained in register 1 is n.

At time $t + n + 1$, σ_{in} receives the second spike, and then it fires to transmit one spike to neurons σ_{c_1}, neurons σ_{c_2} and σ_{c_3}. At time $t+n+2$, since neuron σ_{c_1} receives one spike, it has two spikes, thus, rule $(a^2, a^2) \rightarrow a^2$ is applied to transmits two spikes to neuron σ_{l_0}. Since neuron σ_{l_0} has two spikes, the system will be activated to simulate the instruction l_0.

Since the register machine is used as number accepting device, the deterministic ADD instructions, $l_i : (ADD(r), l_j)$, are considered, shown in Fig. 6. With two spikes in neuron σ_{l_i}, rule $(a^2, a^2) \rightarrow a^2$ is enabled to transmit two spikes to neurons σ_{l_j} and σ_r. Since neuron σ_{l_j} receives two spikes, the system begins to simulate the instruction l_j.

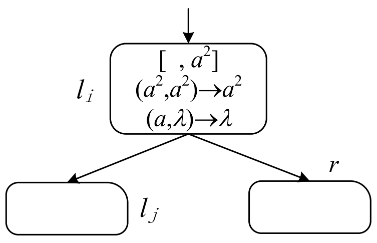
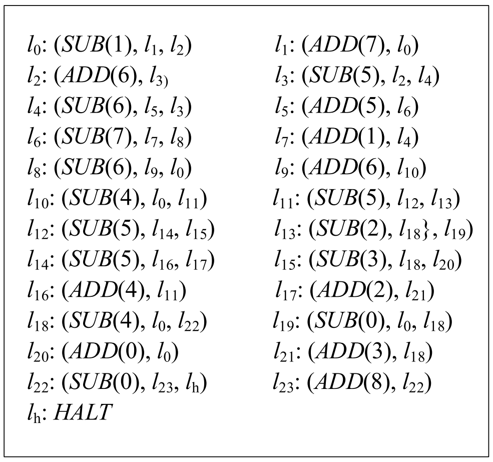


Fig. 6. Module ADD of Π' (simulating $l_i : (ADD(r), l_j)$).



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}: $(SUB(0), l_{23}, l_{\mathrm{h}})$	l_{23}: $(ADD(8), l_{22})$
l_{h}: $HALT$	

Fig. 7. A universal register machine for computing functions [8].

We use module SUB in Fig. 3 to simulate the SUB instructions l_i : $(SUB(r), l_j, l_k)$), which has been described in the proof of Theorem 1. Module FIN is omitted, but neuron σ_{l_h} is remained in the DTSN system. When neuron σ_{l_h} receives two spikes, this indicates that register machine M arrives the halting instruction l_h and halts.

According to the above analysis, it can be found that the register machine as number accepting device is correctly simulated by DTNP system, in which each neuron has at most two rules. Therefore, the theorem holds. □

4. Turing universality of DTNP systems for computing functions

In this section, universality of DTNP systems for computing functions will be discussed. A register machine used to compute a function $f : N^k \to N, M = (m, H, l_0, l_h, I)$, is described as follows. Initially, let all registers be empty, and k arguments in the first k registers are introduced into the register machine. The register machine starts from instruction l_0, and then it runs continuously until the halting instruction l_h arrives. As a result, the number contained in another special register r_t is considered to be computation result, i.e., value of function f. A fixed admissible enumeration of the unary partial recursive functions can be denoted by $(\varphi_0, \varphi_1, \ldots)$. Generally, a register machine is called universal if only if a recursive function g exists so that $\varphi_x(y) = M_u(g(x), y)$ for $\forall\, x, y \in N$.

Fig. 7 gives a register machine $M_u = (9, H, l_0, l_h, I)$ consisting of 9 registers (labeled from 0 to 8), a HALT instruction and 25 instructions (14 SUB and 10 ADD instructions) [8]. Note that register machine M_u is a modified version of the register machine provided in [46], where it adds a new register 8, while original halting instruction is replaced by three instructions: l_{22} : $(SUB(0), l_{23}, l_h)$; l_{23} : $(ADD(8), l_{22})$; l_h : $HALT$. In [46], the original

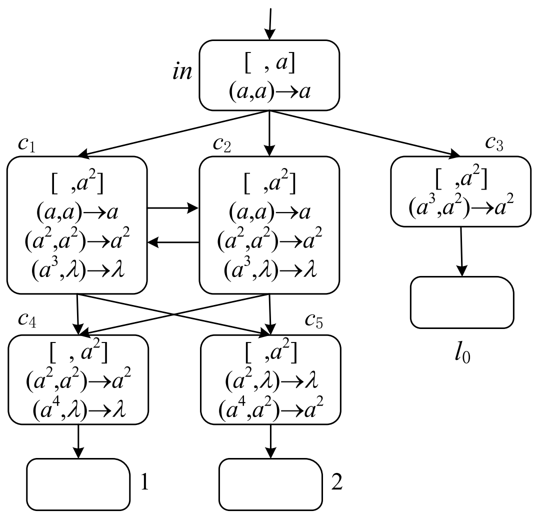


Fig. 8. Module INPUT.

register machine has been proven to be universal for computing functions. Moreover, the modified register machine has been widely used to investigate the universality of SNP systems and variants for computing functions. In this work, register machine M_u will be simulated by constructing a DTNP system.

Theorem 3. *There is an universal DTNP system containing 109 neurons as function computing devices.*

Proof. To simulate universal register machine M_u, a DTNP system Π'' is constructed, which consists of an OUTPUT module, an INPUT module, and ADD and SUB modules. The OUTPUT module is used to export final result, and a spike train is read from the environment by the INPUT module. The ADD instructions of M_u is simulated by the ADD modules, while the SUB instructions is simulated by the SUB modules. For each register r in M_u, there is a neuron σ_r, and we assume that neuron σ_r has $2n$ spikes in its feeding input unit if register r contains the number $n \geq 0$. In addition, each instruction l_i corresponds to a neuron σ_{l_i}. When neuron σ_{l_i} receives two spikes, this means that the instruction l_i will be simulated. If neuron σ_{l_h} has two spikes, this indicates that Π'' completes the simulation of M_u. Initially, all neurons are assumed to be empty.

The INPUT module is provided in Fig. 8, which is used to read the spike train $10^{g(x)}10^y1$ from the environment, where neuron σ_1 receives $2g(x)$ spikes and $2y$ spikes are stored in neuron σ_2.

The INPUT module can be described as follows. At the beginning, the first spike is imported into neuron σ_{in} from the environment. Neuron σ_{in} uses rule $(a, a) \to a$ to send one spike into neurons σ_{c_1}, σ_{c_2} and σ_{c_3}. Since neurons σ_{c_1} and σ_{c_2} have one spike respectively, rule $(a, a) \to a$ is enabled in the two neurons. At each step from this time until the second spike is read into neurons σ_{c_1} and σ_{c_2}, the two neurons send one spike to each other, and they together send two spikes each to neurons σ_{c_4} and σ_{c_5}. At each step, neuron σ_{c_4} uses the rule $(a^2, a^2) \to a^2$ to send two spikes to neuron σ_1; however, two spikes contained in σ_{c_5} will be removed by rule $(a^2, \lambda) \to \lambda$. Note that neuron σ_{in} receives the second spike after $g(x)$ steps. Therefore, $g(x)$ spikes are placed in σ_1.

When the second spike enters both neurons σ_{c_1} and σ_{c_2}, they have two spikes in their feeding input units. Thus, rule $(a^2, a^2) \to a^2$ in neurons σ_{c_1} and σ_{c_2} can be applied respectively. As the result, neurons σ_{c_1} and σ_{c_2} send two spikes to each other, and they together send four spikes to neurons σ_{c_4} and σ_{c_5}. In neuron σ_{c_4}, the received four spikes are removed by rule $(a^4, \lambda) \to \lambda$. At each step,

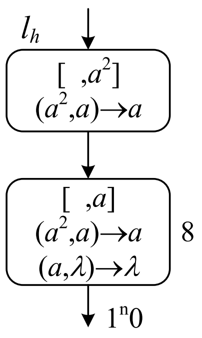


Fig. 9. Module OUPUT.

Table 1
The comparison of different computing models in the term of small number of computing units.

Computing models	The number of neurons
DTPN systems	109
SNQ P systems with one type of spike [15]	181
SNQ P systems with two types of spikes [14]	49
SNP systems [36]	84
Recurrent neural networks [47]	886

neuron σ_{c_5} uses rule $(a^4, a^2) \to a^2$ to send two spikes to neuron σ_2. Therefore, neuron σ_2 had $2y$ spikes once neurons σ_{c_1} and σ_{c_2} receive the third spike.

Note that during neuron σ_{c_3} receives the first two spikes, no rule in the neuron can be enabled. However, once it receives the third spike, neuron σ_{c_3} fires by rule $(a^3, a^2) \to a^2$ and sends two spikes to neuron σ_{l_0}. Since neuron σ_{l_0} contains two spikes, the system begins to simulate instruction l_0 of M'_u.

From Fig. 7, it can be found that all ADD instructions are with the form $l_i : (ADD(r), l_j)$. Therefore, a deterministic ADD module, shown in Fig. 6, is used to simulate the ADD instruction. For the ADD module, its work mechanism has been illustrated in the proof of Theorem 2.

We use the SUB module in Fig. 3 to simulate the SUB instruction $l_i : (SUB(r), l_j, l_k)$. The work mechanism of the SUB module has been illustrated in the proof of Theorem 1.

Suppose that the computation in M_u halts now, meaning that instruction l_h reaches. The result of the computation is stored in register 8, and during the computation the result is never decremented. The OUTPUT module is used to export the computation result, as shown in Fig. 9.

Suppose that at step t neuron σ_{l_h} receives two spikes, and neuron σ_8 has $2n$ spikes, where $n > 0$. Rule $(a^2, a) \to a$ in neuron σ_{l_h} is enabled, and then it transmits one spike to neuron σ_8. Since one spike is transmitted to neuron σ_8 at step $t + 1$, the number of spikes stored in its feeding input unit becomes odd, hence one spike is transmitted to the environment via rule $(a^2, a) \to a$. From step $t + 1$ to step $t + n + 1$, rule $(a^2, a) \to a$ is constantly applied to send one spike to the environment. Since only one spike is remained in neuron σ_8 at step $t + n + 2$, the spike is removed by applying rule $(a, \lambda) \to \lambda$. As a result, n spikes are emitted to the environment, which is exactly the number contained in register 8 of M_u when the system halts.

For DTNP system Π'', we use a total of 109 neurons, including (i) 5 neurons for the INPUT module, (ii) 70 neurons for 14 SUB instructions, (iii) 25 neurons for 25 instructions, and (iv) 9 neurons for 9 registers. According to above analysis to the modules in system Π'', it is obviously indicated that system Π'' can correctly simulate register machine M_u. □

Theorem 3 gives a small number of computing units (i.e., neurons) for DTNP systems as function computing devices to achieve universality. To further evaluate computational power of DTNP systems, Table 1 provides the comparison result of the proposed variant with other computing models in the term of small number of computing units. It can be observed from Table 1 that recurrent neural networks [47] and SNQ P systems with one type of spike [15] need 886 and 161 respectively to achieve Turing universality for computing function. However, DTPN systems need fewer neurons to achieve Turing universality for computing function. Although SNQ P systems with two types of spikes [14] need 49 neurons that are less than that of DTNP systems, they have two types of spikes. However, DTNP systems have only one type of spike. SNP systems [36] need only 84 neurons to achieve Turing universality for computing function, however, DTNP systems need more neurons. The comparison indicates that dynamic threshold mechanism has an influence on the computing capability of DTNP systems.

5. Conclusions

In this paper, inspired from intersecting cortical model (ICM), a new kind of neural-like P systems, dynamic threshold neural P systems (in short, DTNP systems) was presented. DTNP systems have a directed graph structure and are a kind of parallel computing systems. Computational completeness of DTNP systems working as the number generating/accepting devices was established by simulating register machines. Moreover, we constructed an universal DTNP system for computing Turing computable functions.

It is surprising that SNP systems have been proved to be computationally universal despite its inspiration from a simple model of spiking neuron. Unfortunately, successful examples of direct application of SNP systems to solve real-world problems are rarely addressed. From the semantics of SNP systems, regular-expression-type firing mechanism maybe restrict their application in real-world problems. Usually, some nonnative mechanisms need to be introduced in SNP systems in order to overcome the limitation, for example, fuzzy logic [22–28].

Main motivation behind this work is to propose a new variant of neural-like P systems based on dynamic threshold neurons that are inspired from the intersecting cortical model. However, this paper mainly focuses on the construction of the variant and establishing its Turing universality results as the number generating/accepting and function computing devices. Our a further work is considered as follows. It is well-known that the intersecting cortical model was used to develop pulse coupled neural networks (PCNNs), which have been successfully applied to process many image processing problems. Note that DTNP systems also are inspired from intersecting cortical model, so they are analogous to PCNNs, however, they are different from PCNNs in some mechanism aspects. Maybe, DTNP systems become an alternative model for solving these real-world problems.

As a new line of research, we will propose to study the computational efficiency of DTNP systems by introducing concepts of computational complexity theory. Additionally, other interesting questions concerning to DTNP systems can be further investigated, for instance, the universality as language generating devices, asynchronous and sequential working modes, and so on.

Acknowledgments

The authors thank the anonymous reviewers for providing very insightful and constructive suggestions, which have greatly help improve the presentation of this paper.

This work was partially supported by the National Natural Science Foundation of China (No. 61472328), Research Fund of

Table A.2
Acronyms and notations used in this paper.

PCNN	Pulse coupled neural networks.
ICM	Intersecting cortical model.
SNP systems	Spiking neural P systems.
DTNP systems	Dynamic threshold neural P systems.
NRE	All recursively enumerable sets of numbers.
Π	A DTNP system.
$N_2(\Pi)$	The set of numbers computed by Π.
$N_2DTNP_m^n$	The families of all sets computed by DTNP systems having at most m neurons and at most n rules in each neuron.
$N_{acc}(\Pi)$	The set of numbers accepted by Π.
$N_{acc}DTNP_m^n$	The families of all sets accepted by DTNP systems having at most m neurons and at most n rules in each neuron.
O	The singleton alphabet.
a	The spike.
σ_i	The ith neuron in a DTNP system or a module.
u_i	The number of spikes as feeding input via dendrites in σ_i.
τ_i	The (dynamic) threshold in neuron σ_i.
R_i	The finite set of firing rules in neuron σ_i.
E_i	The firing condition.
λ	The empty object.
syn	The directed graph structure of synapses between neurons.
in	The input neuron.
out	The output neuron.
C_0	Initial configuration.
C_t	The configuration at step t.
$ST(\Pi)$	The collection of all spike trains of Π.
M, M_u	A register machine.
m	The number of registers.
r	A register.
H	The set of instruction labels.
I	The set of instructions bijectively labeled by elements of H.
l_0	The starting label.
l_h	The halting label.
l_i	The ith instruction.
HALT	The halting instruction.
ADD	Add instruction or module.
SUB	Subtraction instruction or module.
FIN	The output module.
INPUT	The input module.

Sichuan Science and Technology Project (No. 2018JY0083), Chunhui Project Foundation of the Education Department of China (Nos. Z2016143 and Z2016148), and Research Foundation of the Education Department of Sichuan province (No. 17TD0034), China.

Appendix. Nomenclature

The acronyms and notations used in this paper are provided in Table A.2.

References

[1] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fund. Inform. 71 (2006) 279–308.
[2] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[3] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrance Computing, Oxford University Press, New York, 2010.
[4] H. Peng, J. Yang, J. Wang, T. Wang, Z. Sun, X. Song, X. Luo, X. Huang, Spiking neural P systems with multiple channels, Neural Netw. 95 (2017) 66–71.
[5] L. Pan, Gh. Păun, Spiking neural P systems with anti-spikes, Int. J. Comput. Commun. Control IV (3) (2009) 273–282.
[6] Gh. Păun, Spiking neural P systems with astrocyte-like control, J. UCS 13 (11) (2007) 1707–1721.
[7] L. Pan, J. Wang, H.J. Hoogeboom, Spiking neural P systems with astrocytes, Neural Comput. 24 (3) (2012) 805–825.
[8] T. Song, L. Pan, Gh. Păun, Spiking neural P systems with rules on synapses, Theoret. Comput. Sci. 529 (2014) 82–95.
[9] T. Song, L. Pan, Spiking neural P systems with rules on synapses working in maximum spiking strategy, IEEE Transaction on Nanobioscience 14 (4) (2015) 465–477.
[10] H. Peng, R. Chen, J. Wang, X. Song, T. Wang, F. Yang, Z. Sun, Competitive Spiking Neural P Systems with Rules on Synapses, IEEE Trans. NanoBiosci. 16 (8) (2018) 888–895.
[11] J. Wang, H.J. Hoogeboom, L. Pan, Gh. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with weights, Neural Comput. 22 (2010) 2615–2646.
[12] X. Zeng, X. Zhang, T. Song, L. Pan, Spiking neural P systems with thresholds, Neural Comput. 26 (7) (2014) 1340–1361.
[13] X. Song, J. Wang, H. Peng, G. Ning, Z. Sun, T. Wang, F. Yang, Spiking neural P systems with multiple channels and anti-spikes, Biosystems 169–170 (2018) 13–19.
[14] L. Pan, Gh. Păun, G. Zhang, F. Neri, Spiking neural p systems with communication on request, Int. J. Neural Syst. 28 (8) (2017) 1–13, paper id: 1750042.
[15] T. Wu, F. Bîlbîe, A. Păun, L. Pan, F. Neri, Simplified and yet Turing universal spiking neural P systems with communication on request, Int. J. Neural Syst. (2018). paper id: 1850013. Available at https://doi.org/101142/S0129065718500132.
[16] T. Wu, A. Păun, Z. Zhang, L. Pan, Spiking neural p systems with polarizations, IEEE Trans. Neural Netw. Learn. Syst. 28 (8) (2018) 3349–3360.
[17] O.H. Ibarra, A. Păun, A. Rodríguez-Paťon, Sequential SNP systems based on min/max spike number, Theoret. Comput. Sci. 410 (30) (2009) 2982–2991.
[18] X. Zhang, X. Zeng, B. Luo, L. Pan, On some classes of sequential spiking neural P systems, Neural Comput. 26 (5) (2014) 974–997.
[19] H.M. Chen, T.-O. Ishdorj, Gh. Păun, Computing along the axon, Progress Natual Sci. 17 (4) (2007) 417–423.
[20] M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Egecioglu, M. Ionescu, S. Woodworth, Asynchronous spiking neural P systems, Theoret. Comput. Sci. 410 (24) (2009) 2352–2364.
[21] T. Song, L. Pan, Gh. Păun, Asynchronous spiking neural P systems with local synchronization, Inform. Sci. 219 (2012) 197–207.
[22] H. Peng, J. Wang, M.J. Pérez-Jiménez, H. Wang, J. Shao, T. Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, Inform. Sci. 23 (20) (2013) 106–116.
[23] J. Wang, P. Shi, H. Peng, M.J. Pérez-Jiménez, T. Wang, Weighted fuzzy spiking neural P system, IEEE Trans. Fuzzy Syst. 21 (2) (2013) 209–220.
[24] J. Wang, H. Peng, Adaptive fuzzy spiking neural P systems for fuzzy inference and learning, Int. J. Comput. Math. 90 (4) (2013) 857–868.
[25] T. Wang, G.X. Zhang, J.B. Zhao, Z.Y. He, J. Wang, M.J. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, IEEE Trans. Power Syst. 30 (3) (2015) 1182–1194.
[26] J. Wang, H. Peng, M. Tu, M.J. Pérez-Jiménez, A fault diagnosis method of power systems based on an improved adaptive fuzzy spiking neural P systems and PSO algorithms, Chin. J. Electron. 25 (2) (2016) 320–327.
[27] H. Peng, J. Wang, P. Shi, M.J. Pérez-Jiménez, A. Riscos-Núñez, Fault diagnosis of power systems using fuzzy tissue-like P systems, Integr. Comput.-Aided Eng. 24 (2017) 401–411.
[28] H. Peng, J. Wang, J. Ming, P. Shi, M.J. Pérez-Jiménez, W. Yu, C. Tao, Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems, IEEE Trans. Smart Grid 9 (5) (2018) 4777–4784.
[29] D. Díaz-Pernil, F. Peña-Cantillana, M.A. Gutiérrez-Naranjo, A parallel algorithm for skeletonizing images by using spiking neural P systems, Neurocomputing 115 (2013) 81–91.
[30] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, An unsupervised learning algorithm for membrane computing, Inform. Sci. 304 (2015) 80–91.
[31] H. Peng, J. Wang, P. Shi, M.J. Pérez-Jiménez, A. Riscos-Núñez, An extended membrane system with active membrane to solve automatic fuzzy clustering problems, Int. J. Neural Syst. 26 (3) (2016) 1–17, Article No. 1650004.
[32] H. Peng, P. Shi, J. Wang, A. Riscos-Núñez, M.J. Pérez-Jiménez, Multiobjective fuzzy clustering approach based on tissue-like membrane systems, Knowl.-Based Syst. 125 (2017) 74–82.
[33] G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, Int. J. Neural Syst. 24 (5) (2014) 1–16, Article No. 1440006.
[34] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez, On string languages generated by spiking neural P systems, Fund. Inform. 75 (1) (2007) 141–162.
[35] X. Zhang, X. Zeng, L. Pan, On string language generated by spiking neural P systems with exhaustive use of rules, Nat. Comput. 90 (1) (2008) 535–549.
[36] A. Păun, Gh. Păun, Small universal spiking neural P systems, BioSystems 90 (1) (2007) 47–60.
[37] A. Păun, M. Sidoroff, Sequentially induced by spike number in SNP systems: small universal machines, in: Membrane Computing, Springer, 2012, pp. 333–345.
[38] L. Pan, Gh. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with neuron division and budding, Sci. China Inf. Sci. 54 (8) (2011) 1596–1607.
[39] A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez, Uniform solutions to SAT and Subset Sum by spiking neural P systems, Nat. Comput. 8 (4) (2009) 681–702.
[40] R. Eckhorn, H.J. Reitboeck, M. Arndt, et al., Feature linking via synchrozation among distributed asemblies: simulation of results from cat cortex, Nat. Comput. 2 (3) (1990) 293–307.

[41] R. Eckhorn, A. Frien, R. Bauer, et al., High frequency oscillations in primary visual cortex of awarke monkey, NeuroKey 4 (3) (1993) 243–246.
[42] J.L. Johnson, D. Ritter, Observation of periodic waves in a pulse-coupled neural network, Opt. Lett. 18 (1993) 1253–1255.
[43] J.L. Johnson, M.L. Padget, PCNN models and applications, IEEE Trans. Neural Netw. 10 (3) (1999) 480–498.
[44] J.M. Kinser, A simplified pulse-coupled neural network, Proc. SPIE 2760 (1996) 563–569.
[45] U. Ekblad, J.M. Kinser, J. Atmera, N. Zetterlunda, The intersecting cortical model in image processing, Nucl. Instrum. Methods Phys. Res. A 525 (2004) 392–396.
[46] I. Korec, Small universal register machines, Theoret. Comput. Sci. 168 (2) (1996) 267–301.
[47] H.T. Siegelmann, E.D. Sontag, On the computational power of neural nets, J. Comput. System Sci. 50 (1) (1995) 132–150.