On spiking neural P systems

Oscar H. Ibarra · Mario J. Pérez-Jiménez · Takashi Yokomori

Abstract This work deals with several aspects concerning the formal verification of SN P systems and the computing power of some variants. A methodology based on the information given by the transition diagram associated with an SN P system is presented. The analysis of the diagram cycles codifies invariants formulae which enable us to establish the soundness and completeness of the system with respect to the problem it tries to resolve. We also study the universality of asynchronous and sequential SN P systems and the capability these models have to generate certain classes of languages. Further, by making a slight modification to the standard SN P systems, we introduce a new variant of SN P systems with a special I/O mode, called *SN P modules*, and study their computing power. It is demonstrated that, as string language acceptors and transducers, SN P modules can simulate several types of computing devices such as finite automata, a-finite transducers, and systolic trellis automata.

Keywords Spiking neural P system · Formal verification · Universality · Asynchronous · Sequential · Finite automaton · Finite state transducer · Systolic trellis automaton

1 Introduction

Spiking neural P systems (SN P systems, for short) were introduced by Ionescu et al. (2006) with the aim of incorporating in membrane computing ideas of spiking neurons,

O. H. Ibarra (✉)
Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
e-mail: ibarra@cs.ucsb.edu

M. J. Pérez-Jiménez
Department of Computer Science and AI, University of Seville, Sevilla, Spain
e-mail: marper@us.es

T. Yokomori
Department of Mathematics, Faculty of Education and Integrated Arts and Sciences, Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku, Tokyo 169-8050, Japan
e-mail: yokomori@waseda.jp

which is a promising research line in neural computing (see, e.g., Gerstner and Kistler 2002; Maass 2002).

SN P systems have a biological motivation based on recent discoveries on neural coding. Various studies show evidence of precise temporal correlations between pulses of different neurons and stimulus-dependent synchronisation of the activity in populations of neurons (see, e.g., Eckhorn et al. 1988 or Gray and Kistler 2002). The flow of information is carried on the action potentials, which are encoded by objects (*spikes*) of the same type. These spikes are placed inside the neurons and can be sent from presynaptic to postsynaptic neurons according to specific rules and making use of the time as a support of information.

In the field of membrane computing, SN P systems try to capture the fact that most of the neural impulses are almost identical from an electrical point of view and the intervals of time between signals are crucial.

In this paper we study some interesting topics related to SN P systems, and they are organized as follows. The next section introduces basic concepts about SN P systems. In Section 3, a (restricted) methodology for the formal verification of these systems is presented. Section 4 is devoted to study of some characterizations of generalized models of SN P systems with an extended form of spiking rules. Finally, the computational power of new variants of SN P systems with a special input/output mode is presented in Section 5.

2 Spiking neural P systems

Informally, an SN P system consists of a set of neurons placed in the nodes of a directed graph which send signals (called *spikes*) along the arcs of the graph (representing the *synapses* between neurons). The system evolves according to a set of *spiking rules* and *forgetting rules* each of which is associated with a neuron that uses the rules for sending or internally consuming spikes. The rules for spiking should take into account all spikes present in a neuron not only part of them, although not all spikes are consumed in this way. The produced spikes are sent (maybe with a delay of some steps) to all neurons where there is a synapse outgoing from the neuron where the rule was applied.

A global clock is assumed and in each time unit each neuron which can use a rule should use one such rule (thus, only one rule is used in each neuron). One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment.

Definition 2.1 A spiking neural P system (abbreviated as SN P system) of degree $m \geq 1$, is a tuple of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out)$, where:

- $O = \{a\}$ is the singleton alphabet (the object a is called *spike*);
- $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where $n_i \geq 0$ and R_i is a finite set of rules of the types:

(1) Spiking rules: $E/a^c \rightarrow a; d$,where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
(2) Forgetting rules: $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots, m\}$ (synapses);
- $out \in \{1, 2, \ldots, m\}$.

That is, an SN P system consists of a set of neurons $\{\sigma_1, \ldots, \sigma_m\}$ of the form $\sigma_i = (n_i, R_i)$ where n_i represents the number of spikes initially contained by neuron σ_i. Besides, *syn* specifies the set of arcs of a directed graph representing the *synapses* between neurons, and *out* indicates the index of the *output neuron*.

If a spiking rule is of the form $E/a^c \rightarrow a; d \in R_i$, with $L(E) = \{a^c\}$, then such a rule only is applicable when neuron σ_i contains exactly k spikes. In this case, we denote that rule by $a^c \rightarrow a; d$.

A spiking rule $E/a^c \rightarrow a; d \in R_i$, is applicable in a step t if the neuron σ_i contains exactly k spikes, $a^k \in L(E)$ and $k \geq c$. By applying such a rule, c spikes are consumed, the neuron σ_i is fired and it produces one spike after d time units. If $d = 0$, then the spikes are emitted immediately, otherwise the spikes are emitted after d steps. In the case $d \geq 1$, in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, and it cannot receive new spikes. In the step $t+d$, the neuron spikes and becomes again *open*, hence it can receive spikes. The spike emitted by a neuron σ_i is replicated and it goes to all neurons σ_j such that $(i, j) \in syn$.

A forgetting rule $a^s \rightarrow \lambda \in R_i$ is applicable in a step t if the neuron σ_i contains exactly s spikes in the instant t. By applying such a rule, s spikes are removed from the neuron σ_i.

In each time unit, if some rules from R_i is applicable, then one rule (and only one) must be applied, chosen non-deterministically among all possible rules applicable to σ_i. That is, each neuron processes sequentially its spikes by using, at most, one rule in each time unit. Let us recall that a spiking rule and a forgetting rule cannot be simultaneously applied to a neuron.

The rules are used in a maximally parallel way at the level of the system: at each step, all neurons which can use some rule, must evolve using one rule.

A *configuration* of an SN P system of degree $m \geq 1$ is a tuple $((p_1, q_1), \ldots, (p_m, q_m))$ where p_i $(1 \leq i \leq m)$ describes the number of spikes present in the neuron σ_i, and q_i $(1 \leq i \leq m)$ represents the number of steps to count down until it becomes open. The *initial configuration* of Π is $((n_1, 0), \ldots, (n_m, 0))$, that is, all neurons are open initially. Using the rules of the system in the way described before, a configuration C_2 can be reached from another configuration C_1; such a step is called a *transition*, and we denote it by $C_1 \Longrightarrow C_2$. Any (finite or infinite) sequence of transitions $C_0 \Longrightarrow C_1 \Longrightarrow C_2 \Longrightarrow \cdots C_r$ is a *computation* if C_0 is the initial configuration of the system, and either $r = \infty$ or $r \in \mathbf{N}$ and all neurons are open and no rule can be used (in this case, C_r is called an *halting* computation).

Let Π be an SN P system and let $\mathcal{C} = C_0 \Longrightarrow C_1 \Longrightarrow C_2 \Longrightarrow \cdots$ be a computation in Π. The *spike train* of computation $\mathcal{C}$, denoted by $st(\mathcal{C})$, is the sequence of steps i such that C_i emits a spike out, and we write it in the form $st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle$, with $1 \leq t_1 < t_2 < \ldots$. That is, we consider the moments when the output neuron spikes not when it fires.

The set of all spike trains (over the set $COM(\Pi)$ of all computations) of Π is denoted by $ST(\Pi)$, and can be considered as the result of the *evolution* of the system Π. In Sects. 2 and 3 we consider SN P systems as computing devices, which compute sets of natural numbers. One can associate a set of numbers with $ST(\Pi)$ in several ways. We denote $N(\mathcal{C}) = \{n \mid n = t_i - t_{i-1},\ \text{for } 2 \leq i \leq k, st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle\}$, and as in Ionescu et al. (2006), we can consider the intervals between consecutive spikes as numbers computed by a computation, with several alternatives:

- Taking into account only the first $k \geq 2$ spikes:

$$N_k(\Pi) = \{n \mid \exists \mathcal{C} \in COM(\Pi)(st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle,\ \text{and } st(\mathcal{C}) \text{ has at least } k \text{ spikes, and } n = t_i - t_{i-1},\ \text{for } 2 \leq i \leq k)\}$$

- Taking into account all spikes of computations with infinite spike trains:

$$N_\omega(\Pi) = \{n \mid \exists \mathcal{C} \in COM(\Pi)(st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle \text{ infinite, and } n = t_i - t_{i-1},\ \text{for } i \geq 2)\}$$

- Taking into account all intervals of all computations:

$$N_{all}(\Pi) = \bigcup_{k \geq 2} N_k(\Pi) \cup N_\omega(\Pi)$$

Definition 2.2 An SN P system Π is said to be weakly ω-coherent if there is a computation $\mathcal{C}$ in Π such that $N(\mathcal{C}) = N_{all}(\Pi)$.

That is, an SN P system is weakly ω-coherent if there exists a computation which provides all numbers which all other computations can provide.

3 Formal verification in SN P systems

This section aims to provide a methodology for the verification of certain SN P systems. The idea is to describe the evolution of the system by a transition diagram in such a way that the analysis of its cycles provide invariant formulae of the (evolutive) process.

Definition 3.1 Given a P system, Π, the computation tree, $T(\Pi)$, associated with Π is the rooted labelled tree defined as follows: (a) the nodes of the tree are labelled by configurations of Π; (b) the label of the root is the initial configuration of Π; and (c) the children of a node are labelled by the configurations obtained from the configuration labelling the node through a step of transition.

The maximal branches of the computation tree associated with a P system are called *computations* of the system.

It is possible to consider an orientation in the computation tree, $T(\Pi)$, in a natural way through the parent–child relation, that is, (u,v) is an oriented arc in the tree if and only if u is the parent of v (or v is a child of u).

Next, we define the *transition diagram* of an SN P system as a directed graph where the nodes are configurations and an arc is drawn between two nodes/configurations if a transition is possible between them.

Definition 3.2 Given a SN P system, Π, the oriented graph associated with Π is obtained from the (oriented) computation tree of Π, by identifying the nodes having the same label.

We intend to show that transition diagrams are an interesting tool to establish the formal verification of SN P systems. Next, we justify this assertion with an example.

3.1 An example

Let us consider the SN P system Π_1 whose graphical representation is depicted in Fig. 1. We aim to prove that the system Π_1 computes the numerical set $N_{all}(\Pi_1) = \{r + 2i \mid i \geq 1\}$, in the sense described in the previous section.

Theorem 1

(a) *For each computation, $\mathcal{C}$ of Π_1 we have $N(\mathcal{C}) \subseteq \{r + 2i \mid i \geq 1\}$ (soundness).*

(b) *For each $i \geq 1$ there exists a computation, $\mathcal{C}$ of Π_1 such that $r + 2i \in N(\mathcal{C})$ (completeness). Moreover, The SN P system Π_1 is weakly ω-coherent.*

(c) *For each $q \geq 2$, let $N_{q^*}(\Pi_1)$ be the set*

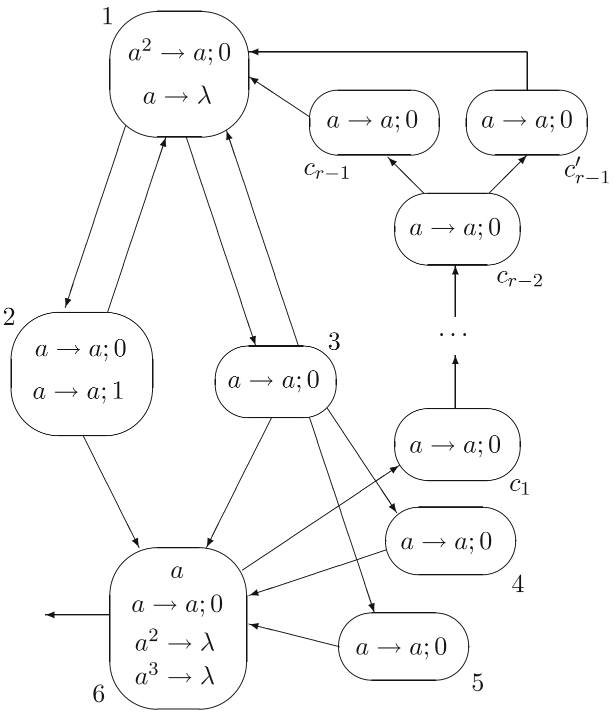


Fig. 1 The SN P system Π_1

$$\{t_q - t_{q-1} | st(\mathcal{C}) = \langle t_1, \ldots, t_{q-1}, t_q, \ldots \rangle, \textit{ for some computation } \mathcal{C} \textit{ of } \Pi_1\}$$

Then, we have $N_{q^*}(\Pi_1) = N_{all}(\Pi_1) = \{r + 2i | i \geq 1\}$.

Proof A configuration C of Π_1 can be described by a tuple $C = (C(1), \ldots, C(6 + r))$ where $C(j)$ is the multiset over $\{a\}$ contained in neuron j, for $1 \leq j \leq 6$, in neuron c_{j-6}, for $7 \leq j \leq 5 + r$, or in neuron c'_{r-1} for $j = 6 + r$. We denote by C^{δ} (with $\delta = 0, 1$) the configuration obtained from C applying in neuron 2 the rule $a \to a; \delta$.

It is easy to prove that the only configurations sending a spike to the neighbouring neurons are C_1 and C^1_{r+3}, and that $C^{00}_{r+4} = C^0_{r+2}$ and $C^{01}_{r+4} = C^1_{r+2}$. The transition diagram associated with Π_1 is depicted in Fig. 2, and we deduce that there is no halting computation.

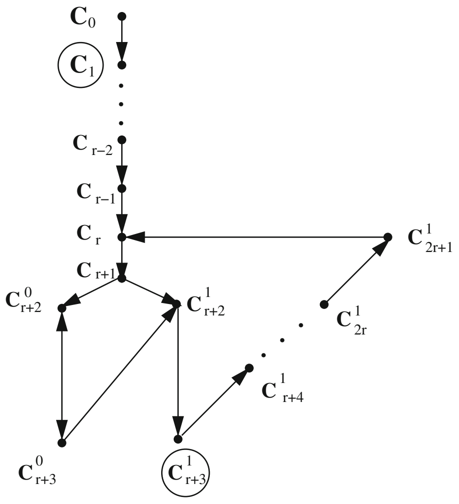
Let us denote by σ the path $C_0 \to C_1 \to \cdots \to C_r \to C_{r+1}$, and let τ be the path $C^1_{r+3} \to C^1_{r+4} \to \cdots \to C^1_{2r+1} \to C_r \to C_{r+1}$. Then, Length($\sigma$) $= r + 1$ and Length(τ) $= r$.

For each $j \geq 0$, let $\gamma(j)$ the path $C_{r+1} \to C^0_{r+2} \overset{(j)}{\longleftrightarrow} C^0_{r+3} \to C^1_{r+2} \to C^1_{r+3}$, meaning that the computation goes through C^0_{r+2} exactly j times. Then, Length ($\gamma(j)$) $= 2(j + 1)$.

Taking into account that the computations of Π_1 are the maximal branches of the computation tree, for every computation, $\mathcal{C}$, of Π_1 there exists an infinite sequence of natural numbers $\{i_k \mid k \geq 1\}$ such that $\mathcal{C}$ can be described through the following path in the transition diagram associated with Π_1 (we denote it by $\mathcal{C}(\{i_k \mid k \geq 1\})$) :

$$\sigma\gamma(i_1)\tau\gamma(i_2)\tau\gamma(i_3)\tau\gamma(i_4)\tau\gamma(i_5)\ldots$$

Fig. 2 Transition diagram of Π_1



Next, we describe the spike train of the computation $\mathcal{C}(\{i_k \mid k \geq 1\}) \equiv C_0 \Rightarrow C_1 \Rightarrow C_2 \Rightarrow C_3 \ldots$, through the sequence of steps i such that the configuration C_i sends a spike out. We have:

$$\begin{cases} t_1 = 1 \\ t_{k+1} = t_k + r + 2(i_k + 1) \end{cases}$$

Hence, $N_{all}(\mathcal{C}(\{i_k \mid k \geq 1\}) = \{t_{k+1} - t_k \mid k \geq 1\} = \{r + 2(i_k + 1) \mid k \geq 1\}$.

(a) Let $\mathcal{C}$ be a computation of Π_1. Then there exists an infinite sequence of natural numbers $\{i_k \mid k \geq 1\}$ such that $\mathcal{C} = \mathcal{C}(\{i_k \mid k \geq 1\})$. Then, $N(\mathcal{C}(\{i_k \mid k \geq 1\}) = \{r + 2(i_k + 1) \mid k \geq 1\} \subseteq \{r + 2i \mid i \geq 1\}$. Hence, $N_{all}(\Pi_1) \subseteq \{r + 2i \mid i \geq 1\}$.

(b) Let us prove that there exists a computation $\mathcal{C}$ of Π_1 such that $N(\mathcal{C}) = N_{all}(\Pi_1) = \{r + 2i \mid i \geq 1\}$.
Indeed, let s be the infinite sequence $\{i_k \mid k \geq 1\}$ such that $i_k = k - 1$, for each $k \geq 1$. Then, $N(\mathcal{C}(s)) = \{r + 2(i_k + 1) \mid k \geq 1\} = \{r + 2i \mid i \geq 1\}$. Hence, $\{r + 2i \mid i \geq 1\} = \{r + 2(i_k + 1) \mid k \geq 1\} = N(\mathcal{C}(s)) \subseteq N_{all}(\Pi_1) \subseteq \{r + 2i \mid i \geq 1\}$.

(c) If $\{i_k \mid k \geq 1\}$ is an infinite sequence of natural numbers, then

$$N(\mathcal{C}(\{i_k \mid k \geq 1\})) = \{r + 2(i_k + 1) \mid k \geq 1\}$$

For each $j \geq 0$ let $\mathcal{C}_j = \mathcal{C}(\{i_k \mid k \geq 1\}$, where $i_k = j$, for every $k \geq 1$.
Let $q \geq 2$. For each $j \geq 0$, we have $N_{q^*}(\mathcal{C}_j) = \{t_q - t_{q-1}\} = \{r + 2(j + 1)\}$. Then, $\{r + 2i \mid i \geq 1\} = \{r + 2(j + 1) \mid j \geq 0\} = \{N_{q^*}(\mathcal{C}_j) \mid j \geq 0\}$. Hence, $N_{q^*}(\Pi_1) \subseteq N_{all}(\Pi_1) = \{r + 2i \mid i \geq 1\}$. $\square$

4 Characterizations

In this section, we will discuss some characterizations of SN P systems. We will mainly look at a generalized model of SN P systems with an extended form of spiking rules. This model was introduced and studied in Chen et al. (2008) and Păun and Păun (2007).

An *extended rule* has the form $E/a^j \to a^p; d$. This rule operates in the same manner as before except that firing sends p spikes along each outgoing synapse (and these p spikes are received simultaneously by each neighboring neuron). Clearly, when $p = 1$ the extended rules reduce to the standard (or non-extended) rules in the original definition. Note also that forgetting rules are just a special case of firing rules, i.e., when $p = 0$.

We will consider systems with three types of neurons:

1. A neuron is *bounded* if every rule in the neuron is of the form $a^i/a^j \to a^p; d$, where $1 \leq j \leq i$, $p \geq 0$, and $d \geq 0$. There can be several such rules in the neuron. These rules are called *bounded rules*.
2. A neuron is *unbounded* if every rule in the neuron is of the form $a^i(a^k)^*/a^j \to a^p; d$, where $i \geq 0, k \geq 1, j \geq 1, p \geq 0, d \geq 0$. Again, there can be several such rules in the neuron. These rules are called *unbounded rules*.
3. A neuron is *general* if it can have *general rules*, i.e., bounded as well as unbounded rules.

One can allow rules like $\alpha_1 + \cdots + \alpha_n \to a^p; d$ in the neuron, where all α_i's have bounded (resp., unbounded) regular expressions as defined above. But such a rule is equivalent to putting n rules $\alpha_i \to a^p; d$ $(1 \leq i \leq n)$ in the neuron. It is known that any regular set over a 1-letter symbol a can be expressed as a finite union of regular sets of the form $\{a^i(a^j)^k \mid k \geq 0\}$ for some $i, j \geq 0$. Note such a set is finite if $j = 0$. We can define three types of SN P systems:

1. *Bounded SN P system*—a system in which every neuron is bounded.
2. *Unbounded SN P system*—a system in which every neuron is either bounded or unbounded.
3. *General SN P system*—a system with general neurons (i.e., each neuron can contain both bounded and unbounded rules).

Let $k \geq 1$. A *k-output SN P system* has k output neurons, $O_1,\ldots,O_k$. We say that the system generates a k-tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$ if, starting from the initial configuration, there is a sequence of steps such that each output neuron O_i generates (sends out to the environment) exactly n_i spikes and then the system eventually halts. We will consider systems with delays and systems without delays (i.e., $d = 0$ in all rules).

4.1 Asynchronous general SN P systems

The standard model of SN P systems is synchronized, meaning that all neurons fire at each step of the computation whenever they are fireable. This synchronization is quite powerful: It is known that a set $Q \subseteq \mathbb{N}^1$ is recursively enumerable if and only if it can be generated by a 1-output general SN P system (with or without delays) (Ionescu et al.2006; Ibarra et al. 2007). This result holds for systems with standard rules or extended rules, and it generalizes to systems with multiple outputs. Thus, such systems are universal.

In Cavaliere et al. (2007) the computational power of SN P systems that operate in an asynchronous mode was introduced and studied. In an *asynchronous SN P system*, we do not require the neurons to fire at each step. During each step, any number of fireable neurons are fired (including the possibility of firing no neurons). When a neuron is fireable it may (or may not) choose to fire during the current step. If the neuron chooses not to fire, it may fire in any later step as long as the rule is still applicable. (The neuron may still receive spikes while it is waiting, which may cause the neuron to no longer be fireable.) Hence there is no restriction on the time interval for firing a neuron. Once a neuron chooses to fire, the appropriate number of spikes are sent out after a delay of exactly d time steps and are received by the neighboring neurons during the step when they are sent.

We recall the definition of a counter machine. A nondeterministic multicounter machine (CM) M is a nondeterministic finite automaton with a finite number of counters (it has no input tape). Each counter can only hold a nonnegative integer. The machine starts in a fixed initial state with all counters set to zero. During the computation, each counter can be incremented by 1, decremented by 1, or tested for zero. A distinguished set of k counters (for some $k \geq 1$) is designated as the output counters. The output counters are non-decreasing (i.e., cannot be decremented). A k-tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$ is generated if M eventually halts in an accepting state, where all non-output counters are set to zero and the contents of the output counters are n_1, ..., n_k, respectively. We will refer to a CM with k output counters (the other counters are auxiliary counters) as a k-output CM.

It is well-known that a set $Q \subseteq \mathbb{N}^k$ is generated by a k-output CM if and only if Q is recursively enumerable. Hence, k-output CMs are universal.

The following result was shown in Cavaliere et al. (2007). It says that SN P systems which operate in an asynchronous mode of computation are still universal provided that the neurons are allowed to use extended rules.

Theorem 2 *A set $Q \subseteq \mathbb{N}^k$ is recursively enumerable if and only if it can be generated by an asynchronous k-output general SN P system with extended rules. The result holds for systems with or without delays.*

It remains an open question whether the above result holds for the case when the system uses only standard (i.e., non-extended) rules.

4.2 Asynchnronous unbounded SN P systems

Next, we will study *unbounded* SN P systems, again assuming the use of extended rules. Recall that these systems can only use bounded and unbounded neurons (i.e., no general neurons are allowed). In contrast to Theorem 2, these systems can be characterized by partially blind multicounter machines (PBCMs).

A *partially blind k-output multicounter machine* (k-output PBCM) (Greibach 1978) is a k-output CM, where the counters cannot be tested for zero. The output counters are non-decreasing. The other counters can be incremented by 1 or decremented by 1, but if there is an attempt to decrement a zero counter then the computation aborts (i.e., the computation becomes invalid). Again, by definition, a successful generation of a k-tuple requires that the machine enters an accepting state with all non-output counters set to zero.

It is well known that k-output PBCMs can be simulated by k-dimensional vector addition systems, and vice-versa (Greibach 1978). Hence, such counter machines are not universal. In particular, a k-output PBCM can generate the reachability set of a vector addition system.

4.2.1 Systems without delays

In Cavaliere et al. (2007), asynchronous unbounded SN P systems without delays were investigated. The systems considered in Cavaliere et al. (2007) are restricted to halt in a pre-defined configuration. Specifically, a computation is valid if, at the time of halting, the numbers of spikes that remain in the neurons are equal to pre-defined values; if the system halts but the neurons do not have the pre-defined values, the computation is considered invalid and the output is ignored. These systems were shown to be equivalent to PBCMs in Cavaliere et al. (2007). However, it was left as an open question whether the 'pre-defined halting' requirement was necessary to prove this result. It was recently shown in Ibarra and Woodworh (2007b) (see also Woodworh 2007) that this condition is, in fact, not necessary. Note that for these systems, firing zero or more neurons at each step is equivalent to firing one or more neurons at each step (otherwise, since there are no delays, the configuration stays the same when no neuron is fired).

Theorem 3 *A set $Q \subseteq \mathbb{N}^k$ is generated by a k-output PBCM if and only if it can be generated by an asynchronous k-output unbounded SN P system without delays. Hence, such SN P systems are not universal.*

Note that by Theorem 2, if we allow both bounded rules and unbounded rules to be present in the neurons, SN P systems become universal. Again, it is an open problem whether the above theorem holds for the case when the system uses only standard rules.

It is known that PBCMs with only one output counter can only generate semilinear sets of numbers. Hence:

Corollary 4.1 *Asynchronous*1*-output unbounded SN P systems without delays can only generate semilinear sets of numbers.*

The results in the following corollary can be obtained using Theorem 3 and the fact that they hold for *k*-output PBCMs.

Corollary 4.2

1. *The family ofk-tuples generated by asynchronous k-output unbounded SN P systems without delays is closed under union and intersection, but not under complementation.*
2. *The membership, emptiness, infiniteness, disjointness, and reachability problems are decidable for asynchronous k-output unbounded SN P systems without delays; but containment and equivalence are undecidable.*

4.2.2 Systems with delays

Theorem 3 assumed an asynchronous SN P system without delays. However, it is possible that allowing delays would give additional power. For asynchronous unbounded SN P systems with delays, we can no longer assume that firing zero or more neurons at each step is equivalent to firing one or more neurons at each step.

Note that not every step in a computation has at least one neuron with a fireable rule. In a given configuration, if no neuron is fireable but at least one neuron is closed, we say that the system is in a *dormant* step. If there is at least one fireable neuron in a given configuration, we say the system is in a *non-dormant* step. (Of course, if a given configuration has no fireable neuron, and all neurons are open, we are in a halting configuration.) Thus, an SN P system with delays might be dormant at some point in the computation until a rule becomes fireable. However, the clock will keep on ticking. Interestingly, the addition of delays does not increase the power of the system (Ibarra and Woodworth 2007b; see also Woodworth 2007):

Theorem 4 *A set $Q \subseteq \mathbb{N}^k$ is generated by ak-output PBCM if and only if it can be generated by an asynchronous k-output unbounded SN P system with delays.*

This result contrasts the result in Ibarra et al. (2007) which shows that synchronous unbounded SN P systems with delays and standard rules (but also standard output) are universal.

4.3 Asynchronous bounded SN P systems

In this section we consider *asynchronous* SN P systems, where the neurons can only use *bounded* rules. We show that these bounded SN P systems with extended rules generate precisely the semilinear sets.

A *k-output monotonic* CM is a nondeterministic machine with k counters, all of which are output counters. The counters are initially set to zero and can only be incremented by 1 or 0 (they cannot be decremented). When the machine halts in an accepting state, the k-tuple of values in the k-counter is said to be generated by the machine. Clearly, a k-output monotonic CM is a special case of a PBCM, where all the counters are output counters and all the instructions are addition instructions.

It is known that a set $Q \subseteq \mathbb{N}^k$ is semilinear if and only if it can be generated by a k-output monotonic CM (Harju et al. 2002). We can show the following (see, e.g., Ibarra and Woodworth 2006; Woodworth 2007):

Theorem 5 *$Q \subseteq \mathbb{N}^k$ can be generated by a k-output monotonic CM if and only if it can be generated by a k-output asynchronous bounded SN P system with extended rules. The result holds for systems with or without delays.*

At present, we do not know whether Theorem 5 holds when the system is restricted to use only standard (non-extended) rules. However, we can show the result holds for synchronous bounded SN P systems using only standard rules (Woodworth 2007).

4.4 Sequential SN P systems

Sequential SN P systems are another closely related model introduced in Ibarra et al. (2006). These are systems that operate in a sequential mode. This means that at every step of the computation, if there is at least one neuron with at least one rule that is fireable, we only allow one such neuron and one such rule (both nondeterministically chosen) to be fired. If there is no fireable rule, then the system is dormant until a rule becomes fireable. However, the clock will keep on ticking. The system is called *strongly sequential* if at every step, there is at least one neuron with a fireable rule.

Unlike for asynchronous systems (considered in the previous section), where the results relied on the fact that the systems use extended rules, the following results in Ibarra et al. (2006) and Woodworth (2007) hold for systems that use standard rules as well as for systems that use extended rules.

Theorem 6 *The following results hold for systems with delays.*

1. *Sequential k-output unbounded SN P systems with standard rules are universal.*
2. *Strongly sequential k-output general SN P systems with standard rules are universal.*
3. *Strongly sequential k-output unbounded SN P systems with standard rules andk-output PBCMs are equivalent.*

The above results also hold for systems with extended rules.

Item 3 in the above theorem improves the result in Ibarra et al. (2006) which required a special halting configuration similar to the halting configuration in Cavaliere et al. (2007). In fact, this halting requirement is not necessary (Ibarra and Woodworth 2007b).

4.5 SN P systems as language generators

SN P systems can be used as language generators, as described in two recent papers (Chen et al. 2006a, b). Consider an SN P system Π with output neuron, *out*, which is bounded. Interpret the output as follows. At times when *out* spikes, a is interpreted to be 1, and at times when it does not spike, interpret the output to be 0. We say that a binary string $x = a_1 \ldots a_n$, where $n \geq 1$, is generated by Π if starting in its initial configuration, it outputs x and halts. We assume that the SN P systems use standard rules.

4.5.1 Regular languages

It was recently shown in Chen et al. (2006a) that for any finite binary language F, the language $F1$ (i.e., with a supplementary suffix of 1) can be generated by a bounded SN P system. This is not true in general if F is an infinite regular language as was shown in Ibarra and Woodworth (2007a):

Observation 4.1 *Let $F = 0^*$. Then $F1$ cannot be generated by a bounded SN P system.*

However, it was also shown in Ibarra and Woodworth (2007a) that by just modifying the previous lanuage F to begin with at least one zero, one can generate $F1$:

Observation 4.2 *Let $F = 0^+$. Then $F1$ can be generated by a bounded SN P system. Thus, it is possible to generate some languages where F is an infinite language.*

The following result, also shown in Ibarra and Woodworth (2007a), contrasts Observation 4.1:

Observation 4.3 *Let $F = 0^*$. Then $1F$ can be generated by a bounded SN P system.*

Observation 4.3 actually generalizes to the following rather surprising result shown in Ibarra and Woodworth (2007a):

Theorem 7 *Let $L \subseteq (0+1)*$. Then the language $1L$ (i.e., with a supplementary prefix 1) can be generated by a bounded SN P system if and only if L is regular. (The result holds also for $0L$, i.e., the supplementary prefix is 0 instead of 1.)*

4.5.2 Another way of generating languages

We can define another way of "generating" a string. We say that a binary string $x = a_1 \ldots a_n$, where $n \geq 0$, is generated by Π if it outputs $1x10^d$, for some d which may depend on x, and halts. Thus, in the generation, Π outputs a 1 before generating x, followed by 10^d for some d. (Note that the prefix 1 and the suffix 10^d are not considered as part of the string.) The set $L(\Pi)$ of binary strings generated in the manner described is the *language* generated by Π. Note that d provides the "space" needed for the computation, just like worktape space in Turing machine computations. One can characterize various classes of languages by putting restrictions on the value of d as a function of the length of x. See Ibarra and Woodworth (2007a).

5 A new variant of SN P systems with I/O mode

In the study of SN P systems there are several manners of defining string languages generated or accepted by those systems. One can consider as language the set of binary strings (studied in the previous section) or traces associated with halting computations (Chen et al. 2006b). Another way is to associate a string over an arbitrary alphabet in such a way that a symbol a_i is associated with a step when a certain designated neuron (output neuron) emits i spikes, leading to a string language over the alphabet generated by an extended SN P system (Chen et al. 2008).

Based on the idea suggested from the latter manner, in this section we introduce a new variant of SN P systems with a special I/O mode, called *SN P modules*, and study their computing power, to show that several types of computing devices based on finite state control can be simulated by SN P modules.

5.1 Spiking neural P module

A *spiking neural P module* (in short, an SN P module), of degree $m \geq 1$, is a construct of the form:

$$\Pi = (\{a\}, \sigma_1, \ldots, \sigma_m, syn, N_i, N_o),$$

where

1. $\{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where
   (a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   (b) R_i is a finite set of *rules* of the following form: $E/a^c \rightarrow a^p$ with a regular expression E over $\{a\}$ and $c \geq p \geq 1$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
4. N_i and $N_o (\subseteq \{1, 2, \ldots, m\})$ indicate the sets of *input neurons* and *output neurons*, respectively.

An SN P module Π behaves in the usual way: An application of a rule of the form $E/a^c \rightarrow a^p$ implies that c spikes are consumed and p spikes are produced, provided the number of spikes in the neuron is covered by E, and so forth. (Note that each rule in Π has no delay and both firing and spiking occur without any time delay.)

In contrast to the usual SN P systems, however, an SN P module Π has the following distinguished feature:

In each step of computation, each input neuron σ_{c_i} (c_i in N_i) takes as input *a number of a's at a time* (from the environment, denoted by Env), while each output neuron $\sigma_{c_o}(c_o \text{ in } N_o)$ produces as output a^p to Env, if a rule of the form $E/a^c \rightarrow a^p$ is successfully applied in σ_{c_o}. (Note that N_i and N_o may share some neurons.)

Thus, an SN P module is a special form of an *extended* SN P system having neurons with *input–output mode*.

In the sequel, we only deal with the case when a regular language $L(E)$ is a singleton $\{a^m\}$; therefore, each rule is given in the form $a^m/a^c \rightarrow a^p$. In particular, for a rule of the form $a^c/a^c \rightarrow a^p$, we simply denote it by $a^c \rightarrow a^p$.

5.2 Computing with SN P modules

As is shown below, SN P modules can simulate in a direct manner several types of computing devices based on finite state transitions.

5.2.1 Simulating finite automata

Let $M = (Q, \Sigma, \delta, q_n, F)$ be a deterministic finite state automaton (DFA), where $\Sigma = \{b_1, \ldots, b_m\}$, $Q = \{q, \ldots, q_n\}$ and q_n is the initial state. We demonstrate that DFA M can be simulated by an SN P module.

Consider an SN P module:

$$\Pi_a = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn, \{3\}, \{3\}),$$

where

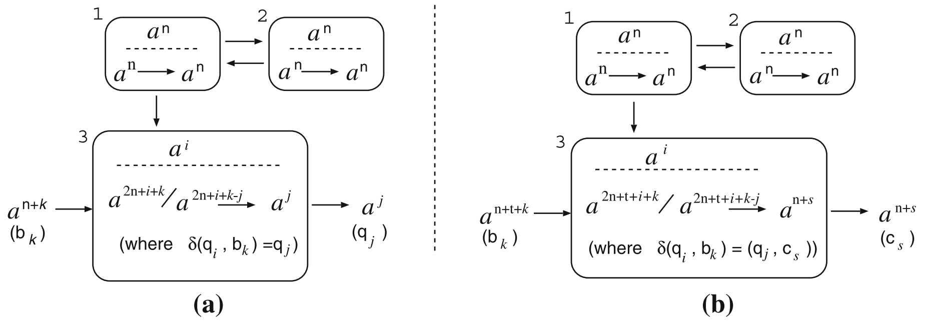
$$\begin{aligned} \sigma_1 &= (n, \{a^n \to a^n\}),\\ \sigma_2 &= (n, \{a^n \to a^n\}),\\ \sigma_3 &= (n, \{a^{2n+i+k}/a^{2n+i+k-j} \to a^j | \delta(q_i, b_k) = q_j\}),\\ syn &= \{(1,2),(2,1),(1,3)\}. \end{aligned}$$

The module is given in a pictorial way in (a) of Fig. 3. Note that n and m are given *fixed* numbers, and that for each $1 \le i \le n$, q_i in Q is represented by a^i, while for each $1 \le k \le m$, b_k in Σ is represented by a^{n+k}. The number of spikes a^i in neuron 3 is refered to (or identified) as a state of Π_a.

The manner of constructing Π_a is a modification of the one presented for the generating SN P system in Chen et al. (2008).

This system works as follows. Neurons 1 and 2 can fire and spike in the first step, and exchange a^n each other. At the same time, neuron 1 emits a^n to neuron 3. This action is repeatedly performed in arbitrary times.

Suppose that, in the first step, neuron 3 contains a^i and is ready to receive input a^{n+k} (representing b_k in Σ) from Env. In the next step, it can fire by receiving a^n (from neuron 1)



1
2
a^n
$a^n \to a^n$
3
a^i
$a^{2n+i+k}/a^{2n+i+k-j} \to a^j$
(where $\delta(q_i, b_k) = q_j$)
a^{n+k}
(b_k)
a^j
(q_j)
(a)
$a^{2n+t+i+k}/a^{2n+t+i+k-j} \to a^{n+s}$
(where $\delta(q_i, b_k) = (q_j, c_s)$)
a^{n+t+k}
a^{n+s}
(c_s)
(b)

Fig. 3 (**a**) SN P module Π_a simulating a finite automaton M: Π_a is currently in the state q_i (represented by a^i). If input b_k (represented by a^{n+k}) is read, then Π_a changes its state to q_j (represented by a^j), and at the same time produces it as output symbol. Thus, one transition $\delta(q_i, b_k) = q_j$ is simulated and the latest state (q_j) is available (on demand). (**b**) SN P module Π_s simulating a finite transducer S: Π_s is currently in the state q_i (represented by a^i). If input b_k (represented by a^{n+t+k}) is read, then Π_s changes its state to q_j (represented by a^j) and produces as output c_s (represented by a^{n+s}). In this moment, there remains a^j in neuron 3 keeping the state q_j. Thus, one transition process $\delta(q_i, b_k) = (q_j, c_s)$ is simulated together with input a^{n+k}, and emits a_j (representing q_j) to Env by consuming $(2n + i + k - j)$ spikes, leaving a^j in neuron 3. Hence, one state transition $\delta(q_i,b_k) = q_j$ is simulated.

Thus, (with 1 step delay) for a given input $w = b_{i_1}, \ldots, b_{i_r}$ in Σ^*, Π_a produces a sequence of states: $z = q_{i_1}, \ldots, q_{i_r}$ (represented by $a^{i_1}, \ldots, a^{i_r}$) such that $\delta(q_{i_\ell}, b_{i_\ell}) = q_{i_{\ell+1}}$ for each $\ell = 0, \ldots, r$ where $q_{i_0} = q_n$. We denote this configuration by $z = \Pi_a(w)$. Then, it holds that w is accepted by M (i.e., $\delta(q_n, w) \in F$) iff $z = \Pi_a(w)$ ends up with a state in F (i.e., $q_{i_r}$is in F).

We now define the language accepted by Π_a as:

$$L(\Pi_a) = \{w \in \Sigma^* | \Pi_a(w) \text{ is in } Q^*F\}.$$

Then, the following is clearly proved.

Theorem 8 *Any regular language L can be expressed as* $L = L(\Pi_a)$ *for some SN P module* Π_a.

5.2.2 *Simulating finite state transducers*

By slightly modified construction presented above, one can construct an SN P module which simulates a finite state transducer (or sequential machine) as well.

Let $S = (Q, \Sigma, \Delta, \delta, q_n, F)$ be a deterministic finite state transducer (with accepting states), where $\Sigma = \{b_1, \ldots, b_m\}$, $\Delta = \{c_1, \ldots, c_t\}$, $Q = \{q_1, \ldots, q_n\}$ and q_n is the initial state.

We now construct an SN P module:

$$\Pi_s = (\{a\}, \sigma_1, \sigma_2, \sigma_3, syn, \{3\}, \{3\}),$$

where

$$\begin{aligned}
\sigma_1 &= (n, \{a^n \rightarrow a^n\}),\\
\sigma_2 &= (n, \{a^n \rightarrow a^n\}),\\
\sigma_3 &= (n, \{a^{2n+t+i+k}/a^{2n+t+i+k-j} \rightarrow a^{n+s} | \delta(q_i, b_k) = (q_j, c_s)\}),\\
syn &= \{(1,2), (2,1), (1,3)\}.
\end{aligned}$$

It should be noted that n, m and t are given *fixed* numbers. Further, for each $1 \leq i \leq n (1 \leq s \leq t)$, q_i in $Q(c_s$ in $\Delta)$ is represented by a^i (a^{n+s}, respectively), while for each $1 \leq k \leq m$, b_k in Σ is represented by a^{n+t+k}.

The module is given in Fig. 3b. The manner of constructing Π_s is a modification of the one for Π_a presented above, and the computation process of an input w (in Σ^*) by Π_s is in parallel to the one by Π_a, with the difference that the former produces an output symbol in Δ, while the latter provides the latest state in Q.

From the manner of constructing Π_s and the previous argument to claim that $L = L(\Pi_a)$, we have the following.

Theorem 9 *Any finite state transducer S can be simulated by some SN P module* Π_s.

5.2.3 *Simulating systolic trellis automata*

Systolic automata are parallel computing models in the form of regular network structures of simple processors with 1-way flow of data. There are two types of underlying structures for regular networks: systolic trees and systolic trellis, while both employ in common a simple finite state control devices as functional elements (of processors).

Here we shall show that systolic trellis automata can be simulated by regular networks of SN P modules. (Note that the developed construction here also apply to simulate systolic tree automata in a straightforward manner.)

Formally, a homogeneous *systolic trellis automaton* (trellis automaton, in short) is a construct

$$K = (\Sigma, \Gamma, \Gamma_0, f)$$

where Σ, Γ and Γ_0 (with $\Sigma \subseteq \Gamma, \Gamma_0 \subseteq \Gamma$) are finite alphabets of terminal, operating and accepting symbols, respectively, and $f : \Gamma \times \Gamma \to \Gamma$ is the transition function. The domain of f is extended to Γ^* as follows: If $|w| = 1$, then $f(w) = w = f^0(w)$, and if $w = x_1 \ldots x_n \in \Gamma^n$ ($n \geq 2$), then

$$f(w) = f(x_1, x_2) f(x_2, x_3) \ldots f(x_{n-1}, x_n).$$

The language accepted by K is defined by: $L(K) = \{w \in \Sigma^* | f^{|w|-1}(w) \in \Gamma_0\}$ (Culik et al. 1996).

Example Consider a trellis automaton $K = (\{a, b\}, \{a, b, A, B, X, Y\}, \{X\}, f)$ where

$$f(a,a) = A, \quad f(a,b) = X, \quad f(b,a) = Y, \quad f(A,A) = A,$$
$$f(A,B) = X, \quad f(A,X) = A, \quad f(B,B) = B, \quad f(X,B) = B$$

and for any other pair (x, y), $f(x, y) = Y$. Note that X is the only accepting symbol, while Y is a *trap* symbol to behave in such a way that once Y is introduced, it leads to a rejecting computation of an input. Figure 4 illustrates examples of both accepting and rejecting computations in K. It is easy to see that $L(K) = \{a^n b^n \mid n \geq 1\}$.

We demonstrate that K can be simulated by a network of SN P modules with *two input neurons* and one output neuron.

Consider an SN P module:

$$\Pi_f = (\{a\}, \sigma_{in1}, \sigma_{in2}, \sigma_{out}, \sigma_1, \ldots, \sigma_{p+1}, syn, \{in1, in2\}, \{out\}),$$

where p is the cardinality of Γ, and the details of the module is given in Fig. 5a.

Note that p is a given *fixed* number, and that for each $1 \leq i \leq p$, q_i in Γ is represented by a^i.

The module Π_f works as follows. Suppose that, in t-th step, neuron in1 (in2) takes as input a^i (a^j) from input 1 (input 2, respectively). Then, a^i of in1 is passed through neuron ($p + 1$) in ($t + 1$)-th step and will reach neuron "out" in ($t + 2$)th step, while a^j of in2 is

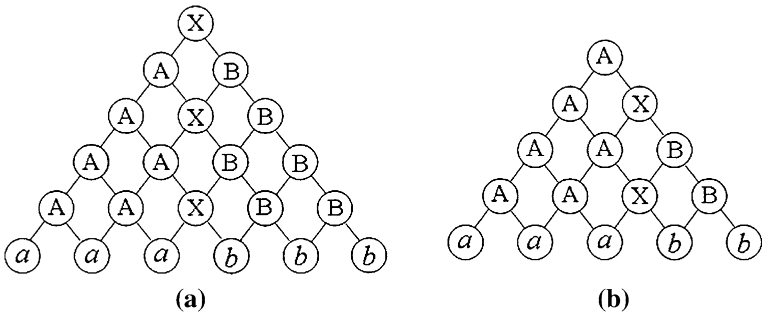


Fig. 4 (**a**) An accepting computation of *aaabbb*; (**b**) A rejecting computation of *aaabb*

distributed to all neurons 1 through p in $(t + 1)$th step, and the sum of p copies of it (i.e., a^{jp}) will reach neuron "out" in $(t + 2)$th step.

On the other hand, in tth step neuron "out" contains no spike and is ready to receive both a^i from neuron $(p + 1)$ and the total amount of a^{jp} from neurons 1 through p, where neuron $(p + 1)$ will emit a^i and each neuron i $(1 \leq i \leq p)$ will emit a^j in $(t + 1)$th step. Therefore, in $(t + 2)$th step, neuron "out" can fire and emit a^k to the environment, by fully exhausting $(i + jp)$ spikes and leaving no spike there. Hence, one state transition $f(q_i, q_k) = q_k$ is simulated by Π_f in three steps.

In order to simulate the global process of computation in K, we have only to allocate Π_f to all nodes of the underlying network structure of K (Fig. 5b) so that the function f is replaced by Π_f. Let $NW(\Pi_f)$ be such a trellis network consisting of Π_fs.

Let $w = q_{i_1}, \ldots, = q_{i_r}$ in Σ^* be an input which is simultaneously fed, in the form $a^{i_1} \ldots a^{i_r}$, from the frontier of the network (shown in Fig. 5b). Then, an input w is accepted by K iff $f^{|w|-1}(w)$ is in Γ_0 iff after $3(|w| - 1)$ steps the module Π_f placed at the root of $NW(\Pi_f)$ produces as output a^h representing some q_h in Γ_0.

Theorem 10 *Any trellis automaton K with the transition function f can be simulated by a trellis network $NW(\Pi_f)$ for some SN P module Π_f.*

6 Conclusions

In this work, we have studied some topics related to SN P systems. Firstly, we presented the transition diagram, associated with a SN P system, as a tool to formally verify that such systems solve a given problem. The methodology is based on the information, encoded by the diagram, about some invariant formulae of the evolution.

Secondly, different characterizations of the power of SN P systems with an extended form of spiking rules were presented. Specifically, we focused on asynchronous (with or without delays), sequential SN P systems and we also used the models as language generators.

Finally, we have started a new research direction to explore the potential computing power of SN P systems with IO mode. It is known that the class of homogeneous trellis

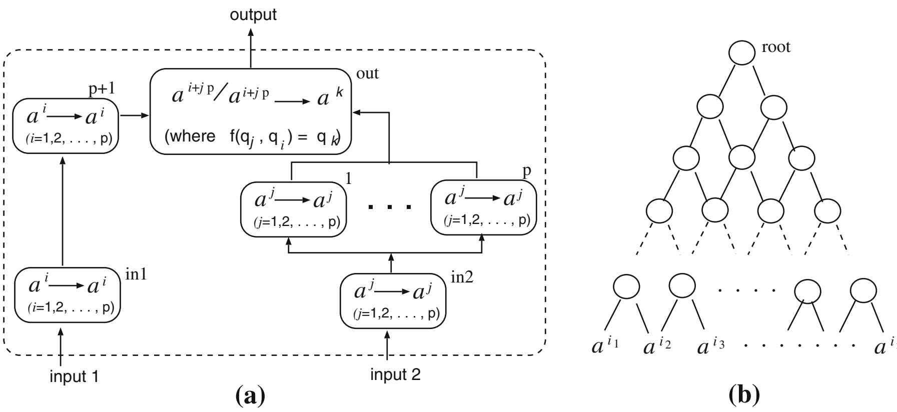


Fig. 5 (**a**) SN P module Π_f simulating a transition function f of trellis automaton K: Π_f takes q_i (represented by a^i) for input 1 and q_j (represented by a^{jp}) for input 2, and produces as output q_k (represented by a^k). Thus, one transition process $f(q_i,q_j) = q_k$ is simulated. (**b**) The underlying network structure of K

languages contains the class of linear languages and is closed under Boolean operations, and is also recongized in time $O(n^2)$ by Turing machines (Culik et al. 1986; Ibarra and Kim 1984). Furthermore, as mentioned earlier, the class of systolic tree languages can be simulated in terms of networks of SN P modules in a natural manner.

In these respects, there remains much to be investigated on SN P modules Π and their networks $NW(\Pi)$ with systolic structures: how far their computing power goes beyond regularity, and their closure and decidability properties, and so on.

Acknowledgments The work of the first author was supported in part by NSF Grant CCF-0524136. The second author was supported by the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the project of excellence TIC-581 of the Junta de Andalucía.

References

Cavaliere M, Egecioglu, Ibarra OH, Ionescu M, Păun Gh, Woodworth S (2007) Asynchronous spiking neural P systems; decidability and undecidability. In: Proceedings of DNA 13, Memphis, TN, USA, pp 246−255

Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2006a) On string languages generated by spiking neural P systems. In: Proceedings of the 4th Brainstorming Week on membrane computing, Seville, Spain, pp 169–194

Chen H, Ionescu M, Păun A, Păun Gh, Popa B (2006b) On trace languages generated by (small) spiking neural P systems. In: Pre-proceedings of the 8th workshop on descriptional complexity of formal systems, Las Cruces, NM, USA, pp 94–105

Chen H, Ionescu M, Ishdorj T-O, Păun A, Păun Gh, Pérez-Jiménez MJ (2008) Spiking neural P systems with extended rules: universality and languages. Nat Comput 7:147–166

Culik K II, Gruska J, Salomaa A (1986) Systolic trellis automata: stability, decidability and complexity. Inf Control 71:2181–230

Eckhorn R, Bauer R, Jordan W, Brosch M, Kruse W, Munk M, Reitboeck HJ (1988) Coherent oscillations: a mechanism of feature linking in the visual cortex? Biol Cybern 60:121–130

Gerstner W, Kistler W (2002) Spiking neuron models. Single neurons, populations, plasticity. Cambridge University Press, Cambridge, MA

Gray CM, Singer W (1989) Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. Proc Natl Acad Sci 86:1698–1702

Greibach S (1978) Remarks on blind and partially blind one-way multicounter machines. Theor Comput Sci 7(3):311–324

Harju T, Ibarra OH, Karhumaki J,Salomaa A (2002) Some decision problems concerning semilinearity and commutation. J Comput Syst Sci 65:278–294

Ibarra OH, Kim SM (1984) Characterization and computational complexity of systolic trellis automata. Theor Comput Sci 29:123–153

Ibarra OH, Woodworth S (2006) Characterizations of some restricted spiking neural P systems. In: Proceedings of the 7th workshop on membrane computing. LNCS 4361:424–442

Ibarra OH, Woodworth S (2007a) Characterizing regular languages by spiking neural P systems. Int J Found Comput Sci 18:1247–1256

Ibarra OH, Woodworth S (2007b) Spiking neural P systems: some characterizations. In: Proceedings of the 16th international symposium on fundamentals of computation theory. LNCS 4639:23–37

Ibarra OH, Woodworth S, Yu F, Păun A (2006) On spiking neural P systems and partially blind counter machines. In: Proceedings of the 5th international conference on unconventional computation. LNCS 4135:113–129

Ibarra OH, Păun A, Păun Gh, Rodríguez-Patón A, Sosik P, Woodworth S (2007) Normal forms for spiking neural P systems. Theor Comput Sci 372:196-217

Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fundamenta Informaticae 71(2–3): 279–308

Maass W (2002) Computing with spikes. Special Issue Found Inf Process TELEMATIK 8(1):32–36

Păun A, Păun Gh (2007) Small universal spiking neural P systems. BioSystems 90(1):48–60

Woodworth S (2007) Computability limits in membrane computing. PhD Dissertation, Department of Computer Science, University of California, Santa Barbara, CA