

Notes on spiking neural P systems and finite automata

Francis George C. Cabarle¹ · Henry N. Adorna¹ ·
Mario J. Pérez-Jiménez²

Published online: 6 July 2016
© Springer Science+Business Media Dordrecht 2016

Abstract Spiking neural P systems (in short, SN P systems) are membrane computing models inspired by the pulse coding of information in biological neurons. SN P systems with standard rules have neurons that emit at most one spike (the pulse) each step, and have either an input or output neuron connected to the environment. A variant known as SN P modules generalize SN P systems by using extended rules (more than one spike can be emitted each step) and a set of input and output neurons. In this work we continue relating SN P modules and finite automata. In particular, we amend and improve previous constructions for the simulations of deterministic finite automata and state transducers. Our improvements reduce the number of neurons from three down to one, so our results are optimal. We also simulate finite automata with output, and we use these simulations to generate automatic sequences.

Keywords Membrane computing · Spiking neural P systems · Finite automata · Automatic sequence

✉ Francis George C. Cabarle
fccabarle@up.edu.ph

Henry N. Adorna
hнадorna@dcs.upd.edu.ph

Mario J. Pérez-Jiménez
marper@us.es

¹ Algorithms and Complexity Lab, Department of Computer Science, University of the Philippines Diliman, Diliman, Quezon City 1101, Philippines

² Department of Computer Science and AI, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

1 Introduction

Spiking neural P systems (in short, SN P systems) introduced in Ionescu et al. (2006), incorporated into membrane computing the idea of pulse coding of information in computations using spiking neurons [see for example Maass 2002; Maass and Bishop 1999 and references therein for more information]. In pulse coding from neuroscience, pulses known as *spikes* are not distinct, so information is instead encoded in their multiplicity or the time they are emitted.

On the computing side, SN P systems have *neurons* processing only one object (the spike symbol a), and neurons are placed on nodes of a directed graph. Arcs between neurons are called *synapses*. SN P systems are known to be universal in both generative (an output is given, but not an input) and accepting (an input is given, but not an output) modes. SN P systems can also solve hard problems in feasible (polynomial to constant) time. Another active line of investigation on the computability and complexity of SN P systems is taking mathematical and biological inspirations in order to create new variants, e.g. asynchronous operation, weighted synapses, rules on synapses, structural plasticity. We do not go into details, and we refer to Cabarle et al. (2015), Ionescu et al. (2006), Leporati et al. (2007), Pan et al. (2011), Păun and Pérez-Jiménez (2009), Song and Pan (2015), Song et al. (2015), Zeng et al. (2014), Zeng et al. (2013) and Zhang et al. (2015) and references therein.

SN P systems with standard rules (as introduced in their seminal paper) have neurons that can emit at most one pulse (the spike) each step, and either an input or output neuron connected to the environment, but not both. In Păun et al. (2007), SN P systems were equipped with both an input and output neuron, and were known as *SN P transducers*. Furthermore, extended rules were introduced in Chen et al. (2008) and Păun and Păun (2007), so that a

neuron can produce more than one spike each step. The introduced *SN P modules* in Ibarra et al. (2010) can then be seen as generalizations of SN P transducers: more than one spike can enter or leave the system, and more than one neuron can function as input or output neuron.

In this work we continue investigations on SN P modules. In particular we amend the problem introduced in the construction of Ibarra et al. (2010), where SN P modules were used to simulate deterministic finite automata and state transducers. Our constructions also reduce the neurons for such SN P modules: from three neurons down to one. Our reduction relies on more involved superscripts, similar to some of the constructions in Neary (2010).

We also provide constructions for SN P modules simulating DFA with output. Establishing simulations between DFA with output and SN P modules, we are then able to generate automatic sequences. Such class of sequences contain, for example, a well known and useful automatic sequence known as the Thue-Morse sequence. The Thue-Morse sequence, among others, play important roles in many areas of mathematics (e.g. number theory) and computer science (e.g. automata theory). Aside from DFA with output, another way to generate automatic sequences is by iterating morphisms. We invite the interested reader to Allouche and Shallit (2003) for further theories and applications related to automatic sequences.

This paper is organized as follows: Sect. 2 provides our preliminaries. In Sect. 3 the main results are presented. Lastly, some final remarks are drawn and then provided in Sect. 4.

2 Preliminaries

It is assumed that the readers are familiar with the basics of membrane computing (a good introduction is Păun (2002) with recent results and information in the P systems webpage in <http://ppage.psyste.ms.eu/> and a recent handbook in Păun et al. (2010)) and formal language theory (available in many monographs). We only briefly mention notions and notations which will be useful throughout the paper.

2.1 Language theory and string notations

We denote the set of natural numbers as $\mathbb{N} = \{0, 1, 2, \dots\}$. Let V be an alphabet, V^* is the set of all finite strings over V with respect to concatenation and the identity element λ (the empty string). The set of all non-empty strings over V is denoted as V^+ so $V^+ = V^* - \{\lambda\}$. We call V a singleton if $V = \{a\}$ and simply write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$. If a is a symbol in V , then $a^0 = \lambda$. A regular expression over an alphabet V is constructed starting from λ and the symbols of V using the operations union, concatenation, and $+$. Specifically, (i) λ and each $a \in V$ are regular expressions,

(ii) if E_1 and E_2 are regular expressions over V then $(E_1 \cup E_2)$, E_1E_2 , and E_1^+ are regular expressions over V , and (iii) nothing else is a regular expression over V . The length of a string $w \in V^*$ is denoted by $|w|$. Unnecessary parentheses are omitted when writing regular expressions, and $E^+ \cup \{\lambda\}$ is written as E^* . We write the language associated with a regular expression E as $L(E)$. If V has k symbols, then $[w]_k = n$ is the base- k representation of $n \in \mathbb{N}$.

2.2 Deterministic finite automata

Definition 1 A *deterministic finite automaton* (in short, a DFA) D , is a 5-tuple $D = (Q, \Sigma, q_1, \delta, F)$, where:

- $Q = \{q_1, \dots, q_n\}$ is a finite set of states,
- $\Sigma = \{b_1, \dots, b_m\}$ is the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $q_1 \in Q$ is the initial state,
- $F \subseteq Q$ is a set of final states.

Definition 2 A *deterministic finite state transducer* (in short, a DFST) with accepting states T , is a 6-tuple $T = (Q, \Sigma, \Delta, q_1, \delta', F)$, where Q , Σ , q_1 , and F are as above, and $\Delta = \{c_1, \dots, c_r\}$ is the output alphabet, while $\delta' : Q \times \Sigma \rightarrow Q \times \Delta$ is the transition function.

Definition 3 A *deterministic finite automaton with output* (in short, a DFAO) M , is a 6-tuple $M = (Q, \Sigma, \delta, q_1, \Delta, \tau)$, where Q , δ , Σ , q_1 , and Δ are as above, and $\tau : Q \rightarrow \Delta$ is the output function.

A given DFAO M defines a function from Σ^* to Δ , denoted as $f_M(w) = \tau(\delta(q_1, w))$ for $w \in \Sigma^*$. If $\Sigma = \{1, \dots, k\}$, denoted as Σ_k , then M is a *k-DFAO*.

Definition 4 A sequence $\mathbf{a} = (a_n)_{n \geq 0}$, is *k-automatic* if there exists a *k-DFAO*, M , such that given $w \in \Sigma_k^*$, $a_n = f_M(w) = \tau(\delta(q_1, w))$, where $[w]_k = n$.

Example 1 (Thue-Morse sequence) The Thue-Morse sequence $\mathbf{t} = (t_n)_{n \geq 0}$ counts the number of 1's (mod 2) in the base-2 representation of n . The 2-DFAO for \mathbf{t} is given in Fig. 1. In order to generate \mathbf{t} , the 2-DFAO is in state q_1 with output 0, if the input bits seen so far sum to 0 (mod 2). In state q_2 with output 1, the 2-DFAO has so far seen input bits that sum to 1 (mod 2). For example, we have $t_0 = 0$, $t_1 = t_2 = 1$, and $t_3 = 0$.

2.3 Spiking neural P systems

Definition 5 A *spiking neural P system* (in short, an SN P system) of degree $m \geq 1$, is a tuple of the form $\Pi = (\{a\}, \sigma_1, \dots, \sigma_m, \text{syn}, \text{in}, \text{out})$

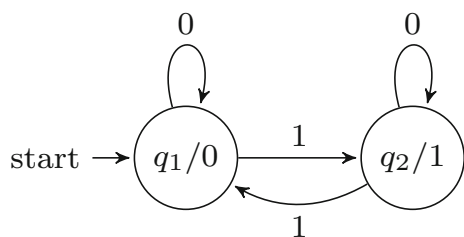


Fig. 1 2-DFAO generating the Thue-Morse sequence

where:

- $\{a\}$ is the singleton alphabet (a is called *spike*);
- $\sigma_1, \dots, \sigma_m$ are *neurons* of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
 - $n_i \geq 0$ is the *initial number of spikes* inside σ_i ;
 - R_i is a finite *set of rules* of the general form: $E/a^c \rightarrow a^p$, where E is a regular expression over $\{a\}$, $c \geq 1$, with $p \geq 0$, and $c \geq p$;
- $syn \subseteq \{1, \dots, m\} \times \{1, \dots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
- $in, out \in \{1, \dots, m\}$ indicate the *input* and *output* neurons, respectively.

A rule $E/a^c \rightarrow a^p; d$ in neuron σ_i (we also say neuron i or simply σ_i if there is no confusion) is called a *spiking rule* if $p \geq 1$. If $p = 0$, the rule is written simply as $a^c \rightarrow \lambda$, known as a *forgetting rule*. If a spiking rule has $L(E) = \{a^c\}$, we simply write it as $a^c \rightarrow a^p$.

The rules are applied as follows: If σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \rightarrow a^p \in R_i$ with $p \geq 1$, is enabled and can be applied. Rule application means consuming c spikes, so only $k - c$ spikes remain in σ_i . The neuron produces p spikes (also referred to as *spiking*) to every σ_j where $(i, j) \in syn$. Applying a forgetting rule means producing no spikes.

SN P systems operate under a global clock, i.e. they are *synchronous*. At every step, every neuron that can apply a rule must do so. It is possible that at least two rules $E_1/a^{c_1} \rightarrow a^{p_1}$ and $E_2/a^{c_2} \rightarrow a^{p_2}$, with $L(E_1) \cap L(E_2) \neq \emptyset$, can be applied at the same step. The system *nondeterministically* chooses exactly one rule to apply. The system is *globally parallel* (each neuron can apply a rule) but is *locally sequential* (a neuron can apply at most one rule).

A *configuration* or state of the system at time t can be described by $C_t = \langle r_1, \dots, r_m \rangle$ for $1 \leq i \leq m$, where neuron i contains $r_i \geq 0$ spikes. The initial configuration of the system is therefore $C_0 = \langle n_1, \dots, n_m \rangle$. Rule application provides us a *transition* from one configuration to another. A computation is any (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration C_0 ; (b) for each $n \geq 2$, the n th configuration of the sequence is obtained from the previous configuration in

one transition step; and (c) if the sequence is finite (called *halting computation*) then the last term is a *halting configuration*, i.e. a configuration where all neurons are open and no rule can be applied.

If σ_{out} produces i spikes in a step, we associate the symbol b_i to that step. In particular, the system (using rules in its output neuron) generates strings over $\Sigma = \{p_1, \dots, p_m\}$, for every rule $r_\ell = E_\ell/a^{i_\ell} \rightarrow a^{p_\ell}$, $1 \leq \ell \leq m$, in σ_{out} . From Chen et al. (2008) we can have two cases: associating b_0 (when no spikes are produced) with a symbol, or as λ . In this work and as in Ibarra et al. (2010), we only consider the latter.

Definition 6 A *spiking neural P module* (in short, an SN P module) of degree $m \geq 1$, is a tuple of the form $\Pi = (\{a\}, \sigma_1, \dots, \sigma_m, syn, N_{in}, N_{out})$ where $\{a\}, \sigma_1, \dots, \sigma_m, syn$ are as above and $N_{in}, N_{out} \subseteq \{1, 2, \dots, m\}$ indicate the *sets of input* and *output* neurons, respectively.

SN P transducers in Păun et al. (2007) operated on strings over a binary alphabet as well considering b_0 as a symbol. SN P modules in Ibarra et al. (2010) are a special type of SN P systems with extended rules and they generalize SN P transducers. SN P modules behave in the usual way as SN P systems, except that spiking and forgetting rules now both contain no delays. In contrast to SN P systems, SN P modules have the following *distinguishing feature*: at each step, each input neuron $\sigma_i, i \in N_{in}$, takes as input *multiple copies* of a from the environment; Each output neuron $\sigma_o, o \in N_{out}$, produces p spikes to the environment, if a rule $E/a^c \rightarrow a^p$ is applied in σ_o ; Note that $N_{in} \cap N_{out}$ is not necessarily empty.

3 Main results

In this section we amend and improve constructions given in Ibarra et al. (2010) to simulate DFA and DFST using SN P modules. Then, k -DFAO are also simulated with SN P modules. Lastly, SN P modules are related to k -automatic sequences.

3.1 DFA and DFST simulations

We briefly recall the constructions from Theorems 8 and 9 of Ibarra et al. (2010) for SN P modules simulating DFAs and DFSTs. The constructions for both DFAs and DFSTs have a similar structure, which is shown in Fig. 2. Let $D = (Q, \Sigma, \delta, q_1, F)$ be a DFA, where $\Sigma = \{b_1, \dots, b_m\}$, $Q = \{q_1, \dots, q_n\}$. The construction for Theorem 8 of Ibarra et al. (2010) for an SN P Module Π_D simulating D is as follows:

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

where

- $\sigma_1 = \sigma_2 = (n, \{a^n \rightarrow a^n\})$,
- $\sigma_3 = (n, \{a^{2n+i+k}/a^{2n+i+k-j} \rightarrow a^j \mid \delta(q_i, b_k) = q_j\})$,
- $syn = \{(1, 2), (2, 1), (1, 3)\}$.

The structure for Π_D is shown in Fig. 2. Note that $n, m \in \mathbb{N}$, are fixed numbers, and each state $q_i \in Q$ is represented as a^i spikes in σ_3 , for $1 \leq i \leq n$. For each symbol $b_k \in \Sigma$, the representation is a^{n+k} . The operation of Π_D is as follows: σ_1 and σ_2 interchange a^n spikes at every step, while σ_1 also sends a^n spikes to σ_3 .

Suppose that D is in state q_i and will receive input b_k , so that σ_3 of Π_D has a^i spikes and will receive a^{n+k} spikes. In the next step, σ_3 will collect a^n spikes from σ_1 , a^{n+k} spikes from the environment, so that the total spikes in σ_3 is a^{2n+i+k} . A rule in σ_3 with $L(E) = \{a^{2n+i+k}\}$ is applied, and the rule consumes $2n + i + k - j$ spikes, therefore leaving only a^j spikes. A single state transition $\delta(q_i, b_k) = q_j$ is therefore simulated.

With a 1-step delay, Π_D receives a given input $w = b_{i_1}, \dots, b_{i_r}$ in Σ^* and produces a sequence of states $z = q_{i_1}, \dots, q_{i_r}$ (represented by a^{i_1}, \dots, a^{i_r}) such that $\delta(q_{i_\ell}, b_{i_\ell}) = q_{i_{\ell+1}}$, for each $\ell = 1, \dots, r$ where $q_{i_1} = q_1$. Then, w is accepted by D (i.e. $\delta(q_1, w) \in F$) iff $z = \Pi_D(w)$ ends with a state in F (i.e. $q_{i_r} \in F$). Let the language accepted by Π_D be defined as:

$$L(\Pi_D) = \{w \in \Sigma^* \mid \Pi_D(w) \in Q^*F\}.$$

Then, the following is Theorem 8 from Ibarra et al. (2010)

Theorem 1 (Ibarra et al. 2010) *Any regular language L can be expressed as $L = L(\Pi_D)$ for some SN P module Π_D .*

The simulation of DFSTs requires a slight modification of the DFA construction. Let $T = (Q, \Sigma, \Delta, \delta', q_1, F)$ be a DFST, where $\Sigma = \{b_1, \dots, b_k\}$, $\Delta = \{c_1, \dots, c_t\}$, $Q = \{q_1, \dots, q_n\}$. We construct the following SN P module simulating T :

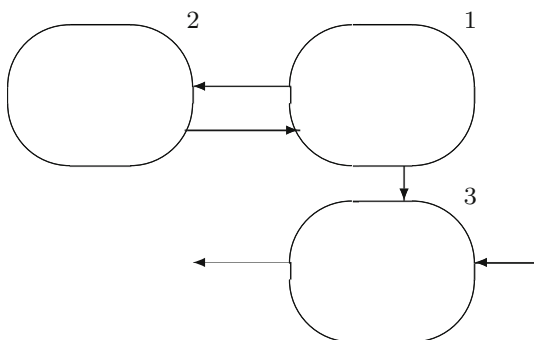


Fig. 2 Structure of SN P modules from Ibarra et al. (2010) simulating DFAs and DFSTs

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

where:

- $\sigma_1 = \sigma_2 = (n, \{a^n \rightarrow a^n\})$,
- $\sigma_3 = (n, \{a^{2n+i+k+t}/a^{2n+i+k+t-j} \rightarrow a^{n+s} \mid \delta'(q_i, b_k) = (q_j, c_s)\})$,
- $syn = \{(1, 2), (2, 1), (1, 3)\}$.

The structure for Π_T is shown in Fig. 2. Note that $n, m, t \in \mathbb{N}$ are fixed numbers. For $1 \leq i \leq n$, $1 \leq k \leq m$, $1 \leq s \leq t$: each state $q_i \in Q$, each input symbol $b_k \in \Sigma$, and each output symbol $c_s \in \Delta$, is represented by a^i , a^{n+t+k} , and a^{n+s} , respectively.

The operation of Π_T given an input $w \in \Sigma^*$ is in parallel to the operation of Π_D ; the difference is that the former produces a $c_s \in \Delta$, while the latter produces a $q_i \in Q$. From the construction of Π_T and the claim in Theorem 1, the following is Theorem 9 from Ibarra et al. (2010):

Theorem 2 (Ibarra et al. 2010) *Any finite transducer T can be simulated by some SN P module Π_T .*

The previous constructions from Ibarra et al. (2010) on simulating DFAs and DFSTs have however, the following technical problem:

Suppose we are to simulate DFA D with at least two transitions, (1) $\delta(q_i, b_k) = q_j$, and (2) $\delta(q_{i'}, b_{k'}) = q_{j'}$. Let $j \neq j', i = k'$, and $k = i'$. The SN P module Π_D simulating D then has at least two rules in σ_3 : $r_1 = a^{2n+i+k}/a^{2n+i+k-j} \rightarrow a^j$, (simulating (1)) and $r_2 = a^{2n+i'+k'}/a^{2n+i'+k'-j'} \rightarrow a^{j'}$ (simulating (2)).

Observe that $2n + i + k = 2n + i' + k'$, so that in σ_3 , the regular expression for r_1 is exactly the regular expression for r_2 . We therefore have a nondeterministic rule selection in σ_3 . However, D being a DFA, transitions to two different states q_j and $q_{j'}$ should be deterministic. Therefore, Π_D is a nondeterministic SN P module that can, at certain steps, incorrectly simulate the DFA D . This nondeterminism also occurs in the DFST simulation.

Next, we amend the problem and modify the constructions for simulating DFAs and DFSTs in SN P modules. Given a DFA D , we construct an SN P module Π'_D simulating D as follows:

$$\Pi'_D = (\{a\}, \sigma_1, syn, \{1\}, \{1\}),$$

where

- $\sigma_1 = (1, \{a^{k(n+1)+i}/a^{k(n+1)+i-j} \rightarrow a^j \mid \delta(q_i, b_k) = q_j\})$,
- $syn = \emptyset$.

We have Π_D containing only 1 neuron, which is both the input and output neuron. Again, $n, m \in \mathbb{N}$ are fixed numbers. Each state q_i is again represented as a^i spikes, for $1 \leq i \leq n$. Each symbol $b_k \in \Sigma$ is now represented as $a^{k(n+1)}$

spikes. The operation of Π'_D is as follows: neuron 1 starts with a^1 spike, representing q_1 in D . Suppose that D is in some state q_i , receives input b_k , and transitions to q_j in the next step. We then have Π'_D combining $a^{k(n+1)}$ spikes from the environment with a^i spikes, so that a rule with regular expression $a^{k(n+1)+i}$ is applied, producing a^j spikes to the environment. After applying such rule, a^j spikes remain in σ_1 , and a single transition of D is simulated.

Note that the construction for Π'_D does not involve nondeterminism, and hence the previous technical problem: Let D have at least two transitions, (1) $\delta(q_i, b_k) = q_j$, and (2) $\delta(q_{i'}, b_{k'}) = q_{j'}$. We again let $j \neq j', i = k'$, and $k = i'$. Note that being a DFA, we have $i \neq k$. Observe that $k(n+1) + i \neq k'(n+1) + i'$. Therefore, Π'_D is deterministic, and has two rules r_1 and r_2 correctly simulating (1) and (2), respectively. We now have the following result.

Theorem 3 Any regular language L can be expressed as $L = L(\Pi'_D)$ for some 1-neuron SN P module Π'_D

For a given DFST T , we construct an SN P module Π'_T simulating T as follows:

$$\Pi'_T = (\{a\}, \sigma_1, \text{syn}, \{1\}, \{1\}),$$

where

$$\begin{aligned} - \sigma_1 &= (1, \{a^{k(n+1)+i+t}/a^{k(n+1)+i+t-j} \rightarrow a^{n+s} \mid \delta'(q_i, b_k) = (q_j, c_s)\}), \\ - \text{syn} &= \emptyset. \end{aligned}$$

We also have Π'_T as a 1-neuron SN P module similar to Π'_D . Again, $n, m, t \in \mathbb{N}$ are fixed numbers, and for each $1 \leq i \leq n, 1 \leq k \leq m$, and $1 \leq s \leq t$: each state $q_i \in Q$, each input symbol $b_k \in \Sigma$, and each output symbol $c_s \in \Delta$, is represented as $a^i, a^{k(n+1)+t}$, and a^{n+s} spikes, respectively. The functioning of Π'_T is in parallel to Π'_D . Unlike Π_T , Π'_T is deterministic and correctly simulates T . We now have the next result.

Theorem 4 Any finite transducer T can be simulated by some 1-neuron SN P module Π'_T .

3.2 k -DFAO simulation and generating automatic sequences

Next, we modify the construction from Theorem 4 specifically for k -DFAOs by: (a) adding a second neuron σ_2 to handle the spikes from σ_1 until end of input is reached, and (b) using σ_2 to output a symbol once the end of input is reached. Also note that in k -DFAOs we have $t \leq n$, since each state must have exactly one output symbol associated with it. Observing k -DFAOs from Definition 3 and DFSTs

from Definition 2, we find a subtle but interesting distinction as follows:

The output of the state after reading the last symbol in the input is the requirement from a k -DFAO, i.e. for every w over some Σ_k , the k -DFAO produces only one $c \in \Delta$ (recall the output function τ); In contrast, the output of DFSTs is a sequence of $Q \times \Delta$ (states and symbols), since $\delta(q_i, b_k) = (q_j, c_s)$. Therefore, if we use the construction in Theorem 4 for DFST in order to simulate k -DFAOs, we must ignore the first $|w| - 1$ symbols in the output of the system in order to obtain the single symbol we require.

For a given k -DFAO $M = (Q, \Sigma, \Delta, \delta, q_1, \tau)$, we have $1 \leq i, j \leq n, 1 \leq s \leq t$, and $1 \leq k \leq m$. Construction of an SN P module Π_M simulating M , is as follows:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \text{syn}, \{1\}, \{2\}),$$

where

$$\begin{aligned} - \sigma_1 &= (1, R_1), \sigma_2 = (0, R_2), \\ - R_1 &= \{a^{k(n+1)+i+t}/a^{k(n+1)+i+t-j} \rightarrow a^{n+s} \mid \delta(q_i, b_k) = (q_j, \tau(q_j) = c_s)\} \\ &\quad \cup \{a^{m(n+1)+n+t+i} \rightarrow a^{m(n+1)+n+t+i} \mid 1 \leq i \leq n\}, \\ - R_2 &= \{a^{n+s} \rightarrow \lambda \mid \tau(q_i) = c_s\} \cup \{a^{m(n+1)+n+t+i} \rightarrow a^{n+s} \mid \tau(q_i) = c_s\}, \\ - \text{syn} &= \{(1, 2)\}. \end{aligned}$$

We have Π_M as a 2-neuron SN P module, and $n, m, t \in \mathbb{N}$ are fixed numbers. Each state $q_i \in Q$, each input symbol $b_k \in \Sigma$, and each output symbol $c_s \in \Delta$, is represented as $a^i, a^{k(n+1)+t}$, and a^{n+s} spikes, respectively. In this case however, we add an end-of-input symbol $\$$ (represented as $a^{m(n+1)+n+t}$ spikes) to the input string, i.e. if $w \in \Sigma^*$, the input for Π_M is $w\$$.

For any $b_k \in \Sigma, \sigma_1$ of Π_M functions in parallel to σ_1 of Π'_D and Π'_T , i.e. every transition $\delta(q_i, b_k) = q_j$ is correctly simulated by σ_1 . The difference however lies during the step when $\$$ enters σ_1 , indicating the end of the input. Suppose during this step σ_1 has a^i spikes, then those spikes are combined with the $a^{m(n+1)+n+t}$ spikes from the environment. Then, one of the n rules in σ_1 with regular expression $a^{m(n+1)+n+t+i}$ is applied, sending $a^{m(n+1)+n+t+i}$ spikes to σ_2 .

The first function of σ_2 is to erase, using forgetting rules, all a^{n+s} spikes it receives from σ_1 . Once σ_2 receives $a^{m(n+1)+n+t+i}$ spikes from σ_1 , this means that the end of the input has been reached. The second function of σ_2 is to produce a^{n+s} spikes exactly once, by using one rule of the form $a^{m(n+1)+n+t+i} \rightarrow a^{n+s}$. The output function $\tau(\delta(q_1, w\$))$ is therefore correctly simulated. We can then have the following result.

Theorem 5 Any k -DFAO M can be simulated by some 2-neuron SN P module Π_M .

Next, we establish the relationship of SN P modules and automatic sequences.

Theorem 6 *Let a sequence $\mathbf{a} = (a_n)_{n \geq 0}$ be k -automatic, then it can be generated by some 2-neuron SN P module Π .*

k -automatic sequences have several interesting robustness properties. One property is the capability to produce the same output sequence given that the input string is read in reverse, i.e. for some finite string $w = a_1 a_2 \dots a_n$, we have $w^R = a_n a_{n-1} \dots a_2 a_1$. It is known [e.g. Allouche and Shallit (2003)] that if $(a_n)_{n \geq 0}$ is a k -automatic sequence, then there exists a k -DFAO M such that $a_n = \tau(\delta(q_0, w^R))$ for all $n \geq 0$, and all $w \in \Sigma_k^*$, where $[w]_k = n$. Since the construction of Theorem 5 simulates both δ and τ , we can include robustness properties as the following result shows.

Theorem 7 *Let $\mathbf{a} = (a_n)_{n \geq 0}$ be a k -automatic sequence. Then, there is some 2-neuron SN P module Π where $\Pi(w^R \S) = a_n$, $w \in \Sigma_k^*$, $[w]_k = n$, and $n \geq 0$.*

4 Final remarks

We have shown that a single neuron in an SN P module is enough to simulate a DFA or DFST, and this is the optimal result in terms of the number of neurons per module (improving and amending some constructions in Ibarra et al. (2010)). In this simulating SN P module with one neuron, a rule simulates a transition in the simulated finite automata, i.e. given a simulated DFA or DFST with m number of transitions, the simulating SN P module with neuron i has $|R_i| = m$. The SN P module simulating a k -DFAO contains two neurons: the general idea is that the first neuron simulates δ while the second neuron simulates τ of the simulated k -DFAO. We were then able to generate automatic sequences using SN P modules, as well as transfer some robustness properties of k -DFAOs to the simulating module.

In Chen et al. (2008), strict inclusions for the types of languages characterized by SN P systems with extended rules having one, two, and three neurons were given. Then in Păun et al. (2007), it was shown that there is no SN P transducer that can compute nonerasing and nonlength preserving morphisms: for all $a \in \Sigma$, the former is a morphism h such that $h(a) \neq \lambda$, while the latter is a morphism h where $|h(a)| \geq 2$. It is known [e.g. in Allouche and Shallit (2003)] that the Thue-Morse morphism is given by $\mu(0) = 01$ and $\mu(1) = 10$. It is interesting to further investigate SN P modules with respect to other classes of sequences, morphisms, and finite transition systems. Another technical note is that in Păun et al. (2007) a time step without a spike entering or leaving the system was considered as a symbol of the alphabet, while in Ibarra et al. (2010) (and in this work) it was considered as λ .

We also leave as an open problem a more systematic analysis of input/output encoding size and system complexity: in the constructions for Theorems 3–4, SN P modules consist of only one neuron for each module, compared to three neurons in the constructions of Ibarra et al. (2010). However, the encoding used in our results is more involved, i.e. with multiplication and addition of indices (instead of simply addition of indices in Ibarra et al. (2010)). On the practical side, SN P modules might also be used for computing functions, as well as other tasks involving (streams of) input-output transformations. Practical applications might include image modification or recognition, sequence analyses, online algorithms, et al. For example, perhaps improving or extending the work done in Díaz-Pernil et al. (2013).

Some preliminary work on SN P modules and morphisms was given in Cabarle et al. (2012). From finite sequences, it is interesting to extend SN P modules to infinite sequences. In Freund and Oswald (2008), extended SN P systems¹ were used as acceptors of ω -languages. SN P modules could also be a way to “go beyond Turing” by way of *interactive computations*, as in interactive components or transducers given in Goldin et al. (2006). While the syntax of SN P modules may prove sufficient for these “interactive tasks”, or at least requiring only minor modifications, a (major) change in the semantics is probably necessary.

Acknowledgments Cabarle is supported by a scholarship from the DOST-ERDT of the Philippines. Adorna is funded by a DOST-ERDT grant, the Semirara Mining Corp. professorial chair of the College of Engineering, UP Diliman, and the UP Diliman Gawad Tsanselor 2015 grant. M.J. Pérez-Jiménez acknowledges the support of the Project TIN2012-37434 of the “Ministerio de Economía y Competitividad” of Spain, co-financed by FEDER funds. Fruitful discussions with Miguel Ángel Martínez-del Amor are also acknowledged.

References

- Allouche J-P, Shallit J (2003) Automatic sequences: theory, applications. Cambridge University Press, Cambridge
- Cabarle FGC, Buño KC, Adorna HN (2012) Spiking neural P systems generating the Thue-Morse sequence. In: Asian conference on membrane computing (2012) pre-proceedings, pp 15–18 Oct. Wuhan, China
- Cabarle FGC, Adorna HN, Pérez-Jiménez MJ, Song T (2015) Spiking neural P systems with structural plasticity. *Neural Comput Appl* 26(8):1905–1917
- Chen H, Ionescu M, Ishdorj T-O, Păun A, Păun G, Pérez-Jiménez MJ (2008) Spiking neural P systems with extended rules: universality and languages. *Natural Comput* 7:147–166

¹ or ESN P systems, in short, are generalizations of SN P systems almost to the point of becoming tissue P systems. ESN P systems are thus generalizations also of (and not to be confused with) SN P systems with extended rules.

- Díaz-Pernil D, Peña-Cantillana F, Gutiérrez-Naranjo MA (2013) A parallel algorithm for skeletonizing images by using spiking neural P systems. *Neurocomputing* 115:81–91
- Freund R, Oswald M (2008) Regular ω -languages defined by finite extended spiking neural P systems. *Fundam Inform* 81(1–2): 65–73
- Goldin D, Smolka S, Wegner P (eds) (2006) *Interactive computation: the new paradigm*. Springer, Berlin
- Ibarra O, Pérez-Jiménez MJ, Yokomori T (2010) On spiking neural P systems. *Nat Comput* 9:475–491
- Ionescu M, Păun G, Yokomori T (2006) Spiking neural P systems. *Fundam Inform* 71(2,3):279–308
- Leporati A, Zandron C, Ferretti C, Mauri G (2007) Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis G, Kefalas P, Păun G, Rozenberg G, Salomaa A (eds) *Membrane computing: 8th international workshop, WMC 2007 Thessaloniki, Greece, June 25–28, 2007 revised selected and invited papers*, Springer, Berlin/Heidelberg, pp 336–352. doi:[10.1007/978-3-540-77312-2_21](https://doi.org/10.1007/978-3-540-77312-2_21)
- Maass W (2002) Computing with spikes. *Found Inform Process* TELEMATIK 8(1):32–36
- Maass W, Bishop C (eds) (1999) *Pulsed neural networks*. MIT Press, Cambridge
- Neary T (2010) A boundary between universality and non-universality in extended spiking neural P systems. In: Martin-Vide C, Fernau H, Dediu AH (eds) *Language and automata theory and applications: 4th international conference, LATA 2010, Trier, Germany, May 24–28, 2010, Proceedings*. Springer, Berlin/Heidelberg, pp 475–487. doi:[10.1007/978-3-642-13089-2_40](https://doi.org/10.1007/978-3-642-13089-2_40)
- Pan L, Păun G, Pérez-Jiménez MJ (2011) Spiking neural P systems with neuron division and budding. *Sci China Inform Sci* 54(8):1596–1607
- Păun G (2002) *Membrane computing: an introduction*. Springer, Berlin
- Păun A, Păun G (2007) Small universal spiking neural P systems. *BioSystems* 90(1):48–60
- Păun G, Pérez-Jiménez MJ, Rozenberg G (2007) Computing morphisms by spiking neural P systems. *J Found Comput Sci* 8(6):1371–1382
- Păun G, Pérez-Jiménez MJ (2009) Spiking neural P systems. Recent results, research topics. In: Condon A et al (eds) *Algorithmic bioprocesses*. Springer, Berlin
- Păun G, Rozenberg G, Salomaa A (eds) (2010) *The Oxford handbook of membrane computing*. OUP, Oxford
- Song T, Pan L (2015) Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. *IEEE Trans NanoBiosci* 14(1):38–44
- Song T, Zou Q, Liu X, Zeng X (2015) Asynchronous spiking neural P systems with rules on synapses. *NeuroComputing* 152(2015): 1439–1445
- The P systems webpage <http://ppage.psystems.eu/>
- Zeng X, Pan L, Pérez-Jiménez MJ (2013) Small universal simple spiking neural P systems with weights. *Sci China Inform Sci* 57(9):1–11
- Zeng X, Zhang X, Song T, Pan L (2014) Spiking neural P systems with thresholds. *Neural Comput* 26(7):1340–1361
- Zhang X, Pan L, Păun A (2015) On the universality of axon P systems. *IEEE Trans Neural Netw Learn Syst* 26(11):2816–2829