Spiking Neural dP Systems: Balance and Homogeneity

Kelvin C. Buño*
Francis George C. Cabarle
Jherico Gabriel Q. Torres
Department of Computer Science (Algortihms & Complexity)
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines

ABSTRACT

This work explores some properties of Spiking Neural dP Systems, namely: (1) balance and, (2) homogeneity. This class of Spiking Neural dP Systems uses Extended SNP systems with Request Rules. As a case study, this work presents two SNdP systems that accept the language $L_{ww} = \{ww \mid w \in \{b_1, \cdots, b_k\}^n, n \geq 1\}$, where these SNdP systems differ in the balance of the input partition, the homogeneity of components of the system, or the number of SNP components. An analysis of the communication cost and running time is given for the two presented SNdP system.

KEYWORDS

Membrane Computing, Spiking Neural P Systems, dP Systems, SNdP systems

1 INTRODUCTION

In the field of computer science, *natural computing* is an area that aims to produce models of computation inspired by nature. One further branch of natural computing is *membrane computing*, inspired by the membrane structure of living cells, which is meant to provide computational models along the lines of computability theory. [18]

P systems are nondeterministic maximally parallel computing model. Communications play vital role in its computations. Communication complexity of P systems was sugested as a research topic in [9] and [17]. In [2], communication cost is associated with an object, denoted as e, representing the energy required to transport objects in and out of the membrane. These are the antiport and symport rules. Antiport rules allows membranes to send output objects and receive input objects at the same time, while symport rules only allow sending objects only in one direction at a time (either in or out, but not at the same time). A related measure for symport/antiport P systems namely *communication difference (Comdif)* was discussed in [10]. *Comdif* is the difference of numbers of input and output objects in an antiport rule. This measure of communication is static. The idea of using communication complexity of P systems as a dynamic measure was initiated in [2]. And in applying the idea of communication complexity started in 1979 by Yao[22], dP systems was introduced in [19]. dP systems has been used for solving NP-complete problems; specifically the satisfiability problem. In [4], dP systems with active membranes was used to solve the N-Queens problem, which was represented using boolean formula. Hence, the problem of solving N-Queens became a satisfiability problem. In [1], distributed tissue P systems with evolutional communication has been used for solving SAT. And in [3], dP systems with active membranes was once again used for solving SAT, but with a different approach.

Our particular P system model focus for this work is the Spiking Neural P system (SNP system). SNP systems was first introduced in [12], a P system that mimics the way neural systems in the brain function by use of electrical impulses sent through neurons via synapses. Since the introduction of SNP systems, much work has been done regarding their computing power, efficiency, and applications as in [5, 6, 14–16], the SNP systems chapter of the handbook in [21], and references therein.

An extension of the spiking rules were introduced in [8], and in [11], request rules were introduced to allow the system to interact with the environment. The generating and accepting power of SNP systems with request rules was analysed in [11], with respect to how the output (for generating) or input (for accepting) spikes of the system are interpreted. In particular, the interpretation of a time step without an input or output spike is referred to as restricted mode, while the interpretation of a time step without an input or output spike as an empty string is referred to as unrestricted mode. Similar to SNP systems with standard rules, SNP working in restricted mode are unable to recognize some context-sensitive non-context-free languages such as $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$.

In addition to the introduction of request rules, input partition was also introduced for SNP systems. Using SNP systems as components of dP system, Spiking Neural dP (SNdP) systems was introduced in [11], with some further investigations in [7]. In [11], it was shown that the use of input distribution enabled the SNdP system to accept languages such as L_{ww} without using unrestricted mode of computation.

The focus of the paper is to analyse two properties of SNdP systems: balance and homogeneity. Balance refers to the the input of each component of the SNdP sytem; a balanced system means that each component receives the same length of input strings from the environment while a nonbalanced system does not need each component to receive input strings of the same length. Homogeneity refers to the neurons of each component: a homogeneous system means that each component has the same set of neurons while a nonhomogeneous system has each component have a different

*corresponding authors: kcbuno@up.edu.ph, fccabarle@up.edu.ph.

set of neurons. Two different SNdP systems are presented, a 2-component balanced and homogeneous, and a 3-component nonbalanced and nonhomogeneous. These two SNdP systems are made to recognize strings from the language L_{ww} and the cost of their communication rules are computed. The 2-component balanced and nonhomogeneous system presented in [11] will be also be considered for comparison purposes.

The paper is arranged as follows: Section 2 gives the defines the concepts and notations used throughout the paper, Section 3 presents the main results of this work, presenting the 3 SNdP systems recognizing L_{ww} and then followed by their analyses and comparison, and finally Section 4 discusses future works from this work.

2 PRELIMINARIES

The reader is assumed to be familiar with the fundamentals of formal language theory. Let Σ be an alphabet, the Kleene closure of Σ, denoted as Σ^*, is the set of all finite words over the alphabet Σ. We denote by λ the word of length zero (the empty string), and by Σ^+ the set of all non-empty words. A multiset M over Σ, is a mapping from Σ to the set of non-negative integers.

DEFINITION 1. *Let P be a non-empty finite set. A collection of $\{P_1, ..., P_n\}$ is called a partition P if and only if for all $1 \leq i, j \leq n$, and $i \neq j$, P_i and P_j are disjoint, and $\bigcup_{i=1}^{n} P_i = P$.*

*A partition $\{P_1, ..., P_n\}$ of P is called a **balanced partition** if and only if for each set P_i, $1 \leq i \leq n$, the size of the set differs by at most one element from all other sets in the partition. That is, $||P_i| - |P_j|| \leq 1$, for all i, j, $1 \leq i, j \leq n$, $i \neq j$. Otherwise, the partition is an **unbalanced partition**.*

2.1 dP Systems

DEFINITION 2. *A dP scheme of degree $n \geq 1$ is a construct[19]:*

$$\Delta = (O, \Pi_1, ..., \Pi_n, R),$$

where:

- *O is an alphabet of objects;*
- *$\Pi_1, ..., \Pi_n$ are cell-like P systems with O as the alphabet of objects and the skin membranes are labeled with $s_1, ..., s_n$ respectively;*
- *R is a finite set of inter-component communication rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n, i \neq j$, and $u, v \in O^*$, with $uv \neq \lambda$; $|uv|$ is called the length of the rule $(s_i, u/v, s_j)$.*

The systems $\Pi_1, ..., \Pi_n$ are called the components of Δ. Each component can take some input and perform computations independently. The system accepts if all components end in a halting configuration. Each component can also communicate symbols with other components as defined by the rules in R.

A configuration in Δ for some computation step $k \geq 0$ is a distribution of multisets over the membranes of each component Π_i, for all $i \in \{1, ..., n\}$. For dP systems using P system components with a dynamic membrane structure, such as P systems with active membranes, it is also important to include the membrane structure in the configuration. The initial configuration of Δ is denoted as δ_0.

2.2 Estimating the Cost of Communication in dP Systems

Another complexity measure to consider in dP scheme is the communication cost. The following defines the communication cost for a given computation step of the system[2, 13, 19]:

DEFINITION 3. *Let Δ be a dP scheme, and $\delta : \delta_0 \Rightarrow \delta_1 \Rightarrow ... \Rightarrow \delta_h$ be a halting computation in Δ, and R the set of inter-component communication rules, with each rule of the form $(s_i, u/v, s_j)$. Then for each $i = 0, 1, ..., h-1$, we have the following complexity measures:*

- *$ComN(\delta_i \Rightarrow \delta_{i+1}) = 1$, if at least one inter-component communication rule, $r \in R$, is used in this transition, and $ComN(\delta_i \Rightarrow \delta_{i+1}) = 0$ otherwise;*
- *$ComR(\delta_i \Rightarrow \delta_{i+1})$ is the number of inter-component communication rules used in this transition;*
- *$ComW(\delta_i \Rightarrow \delta_{i+1})$ is the sum of lengths of all inter-component communication rules used in this transition. Recall that the length of a rule $r \in R$ is $|uv| \geq 1$.*

DEFINITION 4. *Let the set of strings accepted by Δ be denoted as $L(\Delta)$. For $ComX \in \{ComN, ComR, ComW\}$, we define:*

- *$ComX(\delta) = \sum_{i=0}^{h-1} ComX(\delta_i \Rightarrow \delta_{i+1})$, for δ which is a halting computation.*
- *$ComX(w, \Delta) = min\{ComX(\delta) \mid \delta$ is a computation of Δ that accepts the string $w\}$*
- *$ComX(\Delta) = max\{ComX(w, \Delta) \mid w \in L(\Delta)\}$*

2.3 Extended Spiking Neural P Systems with Request Rules

For this work, Extended Spiking Neural P Systems with request rules will be used as components for dP Systems. The following is a formal definition of this model:

DEFINITION 5 (SNP SYSTEM). *[11] An extended Spiking Neural P System with request rules is a construct of the form*

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

where:

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

(2) $\sigma_1, \cdots, \sigma_m$ are pairs $\sigma_i = (n_i, \mathcal{R}_i), 1 \leq i \leq m$, called neurons, where $n_i \geq 0$ and $n_i \in \mathbb{N} \cup \{0\}$ represents the initial spikes in σ_i, and $\mathcal{R}_i$ is a finite set of rules of the form:

(a) $E/a^c \rightarrow a^p$, where E is a regular expression over O, and $c \geq p \geq 1$ (spiking rules);

(b) $E/\lambda \leftarrow a^r$, where E is a regular expression over O, and $r \geq 1$ (request rules);

(c) and $a^s \rightarrow \lambda$, with $s \geq 1$ (forgetting rules) such that there is no spiking rule $E/a^c \rightarrow a^p$; d or request rule $E/\lambda \leftarrow a^r$ such that $a^s \in L(E)$.

(3) syn is the synapse set, a nonreflexive relation on $\{1, \cdots, m\} \times \{1, \cdots, m\}$*;*
(4) $in, out \in \{1, 2, \cdots, m\}$ *are the indices of the input and output neurons, respectively.*

Applications of rules are as follows:

- (spiking rules) if neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \rightarrow a^p \in \mathcal{R}_i$ can be applied. The application of this rule consumes or removes c spikes from σ_i, so that only $k - c$ spikes remain in σ_i. The neuron sends a spike to every σ_j such that $(i, j) \in syn$. The output neuron has a synapse not directed to any other neuron, only to the environment.
- (request rules) if neuron σ_i contains k spikes, $a^k \in L(E)$, then the rule $E/\lambda \leftarrow a^r$ can be applied. The application of this rule produces r spikes, so that the number of spikes in σ_i is now $r + k$. Spikes produced using request rules are considered as input from the environment.
- (forgetting rules) if neuron σ_i contains exactly $s \geq 1$ spikes, then all spikes are removed from σ_i, leaving 0 spikes.

The *non-determinism* in SNP systems occurs when, given two rules $E_1/a^{c_1} \rightarrow a^{b_1}$ and $E_2/a^{c_2} \rightarrow a^{b_2}$, it is possible to have $L(E_1) \cap L(E_2) \neq \emptyset$. In this situation, only one rule will be non-deterministically chosen and applied. SNP systems are *globally parallel* but are *locally sequential*. That is, an SNP system assumes a global clock for all its neurons and are synchronous. If any neuron can fire a rule it must do so, but each neuron can only fire at most one rule each time step. All neurons fire at the same time.

Note that if a spiking rule can be applied, then there is no forgetting rule that can be applied, and vice versa i.e. if a spiking and forgetting rule have regular expressions E_{spik} and E_{forg} respectively in the same neuron, then $L(E_{spik}) \cap L(E_{forg}) = \emptyset$.

Definition 6. *Let* Π *be an Extended SNP system with Request Rules. A configuration* C_i *of* Π *at time step* i *is a vector* $C_i = \langle r_1, \cdots, r_m \rangle$*, where each element of the vector represents the number of the symbol* a *(spikes) in neuron* σ_j*, with* $0 \leq r_j \leq m$*.*

The initial configuration of Π *is defined as* $C_0 = \langle n_1, \cdots, n_m \rangle$*.*

Definition 7. *Let* Π *be an Extended SNP system with Request Rules. A configuration* C_i *yields a configuration* C_{i+1}*, denoted as* $C_i \Rightarrow C_{i+1}$*, if and only if* C_{i+1} *is obtained from* C_i *by applying spiking rules, request rules, or forgetting rules in a globally parallel and locally sequential manner.*

Definition 8. *Let* Π *be an Extended SNP system with Request Rules. If from a configuration* C_i*,* $i \geq 0$*, there are no more applicable rules of any form (i.e. there does not exist* C_{i+1} *such that* $C_i \Rightarrow C_{i+1}$*), then configuration* C_i *is referred to as a halting configuration. A halting configuration is denoted as* C_h*.*

Definition 9. *Let* Π *be an Extended SNP system with Request Rules. A computation of* Π *is a sequence of configurations* $C_\Pi : C_0 \Rightarrow C_1 \Rightarrow ...$*, where* C_0 *is the initial configuration of* Π*.*

A computation C_Π *is a halting computation if and only if* $C_\Pi : C_0 \Rightarrow ... \Rightarrow C_h$*, where* C_h *is a halting configuration.*

In [8, 11], a possible result of a computation extended SNP system with request rules as strings. For an extended SNP system with request rules Π, one neuron is designated as the input neuron (this is the only neuron in Π with request rules), denoted as σ_{in}. For a string over an alphabet $\Sigma = \{b_1, ..., b_k\}$, for some $k \geq 1$, a symbol b_i can be associated with a step of a computation when i spikes are requested by σ_{in}. The sequence of spikes that enter Π through σ_{in} can then be associated with a string over Σ. Note here that for this paper, we do not associate no spike entering the system to any symbol in Σ, i.e. we use unrestricted mode in our work. The result of a computation of Π is formally defined as follows.

Definition 10. *Let* Π *be an Extended SNP system with Request Rules with a designated input neuron* σ_{in}*. For a halting computation* C_Π*, suppose that a sequence of spikes* $\langle i_1, ..., i_n \rangle$*, for some* $n \geq 1$*, enter* Π *through* σ_{in}*, and for all* $1 \leq j \leq n$*,* $1 \leq i_j \leq k$*. Then,* Π *accepts the string* $b_{i_1}...b_{i_k}$*.*

The set of all strings over $\{b_1, ..., b_k\}$ *accepted by* Π *is denoted as* $L(\Pi)$*.*

Note here that strings associated with sequences of spikes from a non-halting computation are not accepted.

2.4 Spiking Neural dP Systems

Definition 11 (SNdP system from [11]). *A Spiking Neural dP System (SNdP system) is a construct of the form* $\Delta = (O, \Pi_1, \cdots, \Pi_n, esyn)$*, where:*

1. $O = \{a\}$ *is the singleton alphabet.*
2. $\Pi_i = (O, \sigma_{i,1}, \cdots, \sigma_{i,n_i}, syn, in_i)$ *is an extended SNP system with request rules present only in neuron* σ_{in_i}
   - $\sigma_{i,j} = (n_{i,j}, R_{i,j})$*, where* $n_{i,j}$ *is the number of spikes initially present in the neuron and* $R_{i,j}$ *is the finite set of rules of the neuron,* $1 \leq j \leq n_i$*.*
   - *Only the input neuron of each component (indicated by* in_i*) will have request rules.*
3. *esyn, is the set of external synapses between neurons* Π_i *with the restriction that between any two sysems* Π_i, Π_j*, there exists at most one neuron of* Π_i *with at most one synapse to a neuron of* Π_j *and vice versa. All rules defined in a neuron with an external synapse are considered as intercomponent communication rule.*

The systems $\Pi_i, 1 \leq i \leq n$ are called *components* of Δ. Each component, Π can take an input through the use of request rules, do computations using the spiking and forgetting rules of the neurons, and communicate with other components through the synapses in *esyn*.

When r spikes are taken from the environment, a symbol b_r is associated with that step, and the strings that can be formed by and introduced in the system are over an alphabet $\Sigma = \{b_1, \cdots, b_k\}$, with k being the maximum

number of spikes that can be obtained by the request rule of a component.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \cdots x_n$ over the alphabet Σ if the components $\Pi_1, ..., \Pi_n$, beginning from their initial configurations and working in a synchronous, nondeterministic way, bring from the environment the substrings $x_1, ..., x_n$, respectively, and halts eventually. The set of all strings over Σ accepted by Δ is denoted as $L(\Delta)$.

SNdP systems are synchronized by a universal clock that exists for all components and neurons to make sure that the time is marked in the same way throughout the whole system.

Balance and Homogeneity. This work analyses two properties of SNdP systems: *balance* and *homogeneity*.

Balance refers to the input partition of the SNdP system. Let Δ be an SNdP system of degree n and $\Sigma = \{b_1, ..., b_k\}$ be an alphabet. Let $L(\Delta)$ denote the language of Δ over Σ. For a string $x \in \Sigma^*$, let $x = x_1...x_n$, where each $x_i \in \Sigma^*$. Then the set $\{x_1, ..., x_n\}$ is referred to as an input partition of x. The partition $\{x_1, ..., x_n\}$ is balanced if $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$. Otherwise, the partition $\{x_1, ..., x_n\}$ is said to be nonbalanced.[20].

A homogeneous SNP system is an SNP system where all neurons have the same set of rules[23]. In SNdP systems, however, homogeneity does not apply to the individual neurons of each component, but rather to the components of the SNdP itself. That is, there is a one-to-one correspondence between the components of a homogeneous SNdP system. Each component has the same set of neurons, the same set of starting spikes, spiking rules, request rules, and forgetting rules. Likewise, in a nonhomogeneous SNdP system, components can have different sets of neurons.

As a case study, the aforementioned balance and homogeneity are applied to the language $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$.

2.5 An example of an SNdP system with balanced inputs and nonhomogeneous components

Figure 1 shows an SNdP system from [11] that recognizes the language, $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$.

Neuron $\sigma_{(1,1)}$ can bring in some spikes from the environment, say r_1 spikes, while $\sigma_{(2,1)}$ can bring in r_2 spikes. These spikes are moved one-by-one up to at most a total of k steps, where k is the cardinality of the alphabet. They are moved to $\sigma_{(1,2)}$ and $\sigma_{(1,3)}$ respectively, through the rules $a^4a^+/a \to a$, and are duplicated in the neurons $\sigma_{(1,4)}$ and $\sigma_{(1,5)}$, where they are removed by the forgetting rule $a^2 \to \lambda$.

If $r_1 = r_2$, then the neurons will reach a computation where the spikes will be forgotten in $\sigma_{(1,4)}$ and $\sigma_{(1,5)}$ will use the rule $a^6 \to a^3$ to send spikes to $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$ to start the process all over again and read one further symbol of the string.

However, if $r_1 \neq r_2$, then one of $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$ send one spike while the other sends three, causing four spikes to arrive at neuron $\sigma_{(1,4)}$, which then sends spikes to $\sigma_{(1,6)}$ and $\sigma_{(1,1)}$, two neurons that send forever spikes to each other, creating a computation that never halts.

Ultimately, after going through the whole string, if the two substrings received by the input neurons are equal, then the computation will stop.

$r - 1$ steps are needed for using the rules $a^4a^+/a \to a$ and one step for the rule $a^4 \to a^3$, then two more steps for sending three spikes to neurons $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$.

3 RESULTS

In [11], an SNdP accepting the language $L_{ww} = \{ww \mid w \in \{b_1, b_2, \cdots, b_k\}^n, n \geq 1\}$ was presented. This system uses a balanced partition, and uses two components that have different sets of neurons. Thus, for the purpose of our work, we categorize this particular SNdP accepting L_{ww} as a Balanced and Nonhomogeneous system.

For the following, an SNdP with a balanced partition, using two similar components, accepting L_{ww} is presented.

3.1 Balanced and Homogeneous SNdP

PROPOSITION 1. *$L_{ww} = \{ww \mid w \in \{b_1, b_2, \cdots, b_k\}^n, n \geq 1\}$ can be recognized by a 2-component SNdP system, Δ, with balanced input and homogeneous components, where $ComN(\Delta) = kn$, $ComR(\Delta) = 2kn$, and $ComW(\Delta) = 2kn + 4n + 2$, and Δ uses a total of 16 neurons in 2 components.*

Proof: Let $\Sigma_k = \{b_1, \cdots, b_k\}$ and $w' \in \Sigma_k^+$. The input string, w', is partitioned into two substrings in this way: $w' = w_1 w_2$, where $||w_1| - |w_2|| \leq 1$.

The Balanced and Homogeneous SNdP system that can recognize the above language using the above partition is the following:

$$\Delta = (\{a\}, \Pi_1, \Pi_2, \{((2,2),(1,4)),((1,2),(2,4))\})$$

Figure 2 shows the 2-component SNdP system with balanced input partition and homogeneous components recognizing L_{ww}. r spikes produced using the request rules represents the character b_r from the alphabet of w, $\{b_1, b_2, \cdots, b_k\}$. The first component, Π_1, takes in w_1 while the second component, Π_2 takes in w_2. The system then tries to check if both components received the same inputs strings from the environment. Because the input is partitioned in a balance manner, both components would be taking an input of length n from the environment, so that $|w'| = 2n$. The system is homogeneous, so both components have the same number of neurons and these neurons have the same rules.

For brevity of discussing the flow of the computation, we only refer to the computation of one component (unless both components are involved for a computation step). As the components are homogeneous, they would have the same flow of computation. Let $\alpha \in \{1, 2\}$.

At the start of the computation, $\sigma_{(\alpha,1)}$ will get r spikes from the environment using the request rule $a^2/\lambda \leftarrow a^r$, $1 \leq r \leq k$. Then, $\sigma_{(\alpha,1)}$ will send the $r - 1$ spikes one-by-one using the spiking rule $(a^3)a^+/a \to a$ to $\sigma_{(\alpha,2)}$. For the last

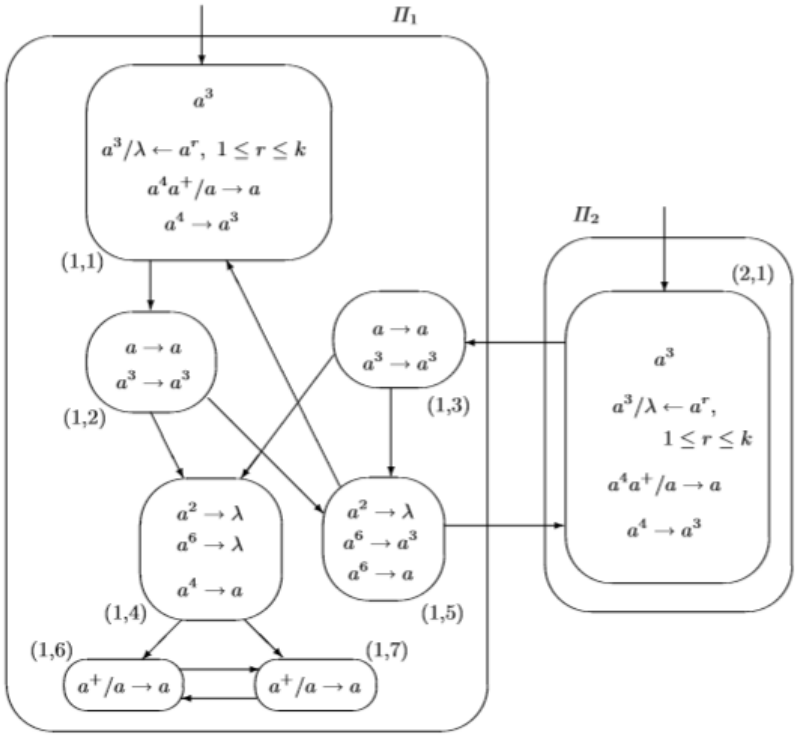

Figure 1: A balanced, nonhomogeneous SNdP system (presented in [11]) recognizing $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$

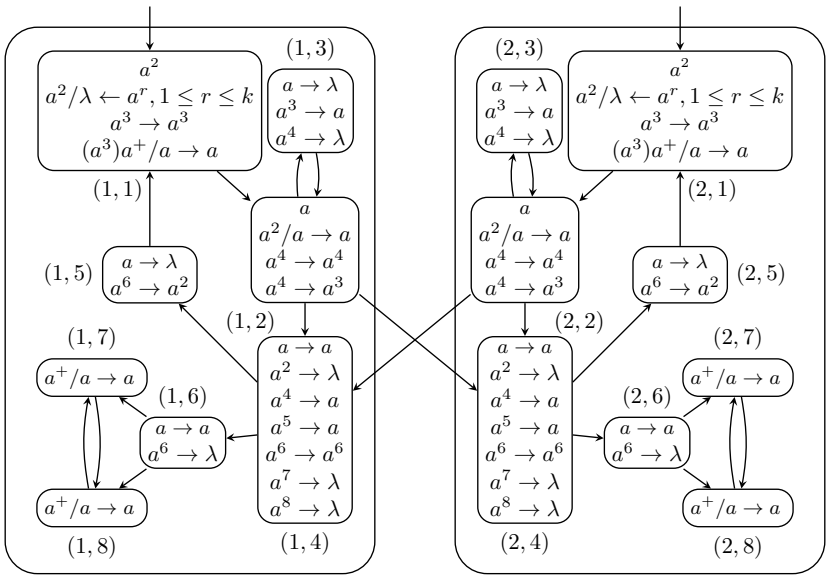

Figure 2: The SNdP system for the proof of Proposition 1

spike, the rule $a^3 \to a^3$ will be used instead, consuming all remaining spikes in $\sigma_{(\alpha,1)}$.

For the first $r-1$ spikes, neuron $\sigma_{(1,2)}$ will send its spikes to $\sigma_{(1,4)}$ and $\sigma_{(2,4)}$, using spiking rule $a^2/a \to a$. Likewise, $\sigma_{(2,2)}$ will also send its spikes to $\sigma_{(1,4)}$ and $\sigma_{(2,4)}$. For the last spike of the current symbol, $\sigma_{(\alpha,2)}$ will non-deterministically choose between rules $a^4 \to a^4$ xor $a^4 \to a^3$ to apply. Neuron $\sigma_{(\alpha,3)}$ are used to ensure that neuron $\sigma_{(\alpha,2)}$ will have one

spike when it uses rule $a^4 \to a^3$. The number of spikes sent by $\sigma_{(1,2)}$ and $\sigma_{(2,2)}$ to the other components will be interpreted as follows:

- 1 spike from $\sigma_{(\alpha,2)}$ indicates that component Π_α is still processing the symbol it requested from the environment.
- 3 spikes indicate that component Π_α has finished processing the last spike of its current symbol. In addition, 3 spikes mean that Π_α wishes to request a new symbol from the environment.
- 4 spikes indicate that component Π_α has finished processing the last spike of its current symbol. In addition, 4 spikes mean that Π_α will not request anymore symbols from the environment.

The purpose of neuron $\sigma_{(\alpha,4)}$ is to receive the spike sent from the other component's neuron 2 and, together with the spike sent by $\sigma_{(\alpha,2)}$, decide if the current symbols processed by the components are the same. The following are the responses of $\sigma_{(\alpha,4)}$ based on the sum of spikes received from $\sigma_{(1,2)}$ and $\sigma_{(2,2)}$:

- Positive Response - Neuron $\sigma_{(\alpha,4)}$ will either enable the request of new symbols, or halt the computation and hence, the system, Δ, will accept the input string $w' = w_1 w_2$.
  - (2 spikes) Both components are still processing the spikes of their respective current symbols, so computation should continue. The 2 spikes are erased by a forgetting rule.
  - (6 spikes) Both components have processed the last spike of their respective current symbols, and that both components wishes to request a new symbol from the environment. Neuron $\sigma_{(\alpha,4)}$ will use rule $a^6 \to a^6$ to send 6 spikes to neurons $\sigma_{(\alpha,5)}$, to continue the symbol request process by returning 2 spikes to neuron $\sigma_{(\alpha,1)}$. Neuron $\sigma_{(\alpha,6)}$ which will erase the 6 spikes.
  - (7 spikes) Both components have processed the last spike of their respective current symbols, but one component wishes to request a new symbol, while the other wishes to end the system computation. For this case, we opt to just end the computation if at least one component does not wish to request a new symbol. Hence, the 7 spikes are forgotten to stop further request of new symbols. This ends the computation of Δ.
  - (8 spikes) Both components have processed the last spike of their respective current symbols, and both components wishes to end the computation. Hence, the 8 spikes are forgotten to stop further request of new symbols. This ends the computation of Δ.
- Negative Response - Neuron $\sigma_{(\alpha,4)}$ will cause a nonhalting computation, making the system reject the input string.
  - (4 spikes, 5 spikes) One component has processed the last spike of its request symbol, while the other component has not. This indicates that the two requested symbols of the two components are not the same. Hence, the input string $w' \notin L_{ww}$, and must be rejected. Neuron $\sigma_{(\alpha,4)}$ will send one spike to neuron $\sigma_{(\alpha,5)}$, which will forget the spike, and to neuron $\sigma_{(\alpha,6)}$, which will relay the spike to neurons $\sigma_{(\alpha,7)}$ and $\sigma_{(\alpha,8)}$. Neurons $\sigma_{(\alpha,7)}$ and $\sigma_{(\alpha,8)}$ will exchange the spikes indefinitely, causing a nonhalting computation.
  - (1 spike) These occurs when in the previous computation step, $\sigma_{(\alpha,4)}$ received either 4 spikes or 5 spikes. Neuron $\sigma_{(\alpha,4)}$ will fire 1 spike, but this will not affect the nonhalting computation anymore.

The algorithm makes use of two components with eight neurons each. Suppose the input string to Δ, w', is of length $2n$. Partitioning w' to $w_1 w_2$, each substring is of length n. Analysing the running time of an accepting computation, for $\alpha \in \{1, 2\}$, for every symbol, b_r, $1 \leq r \leq k$, of input string w_α to component Π_α, it takes 1 step to request r spikes from the environment, and takes r steps before the final spike of b_r arrives at neuron $\sigma_{(\alpha,2)}$. Then, 1 step to move the last spike to $\sigma_{(\alpha,4)}$. And then, two more steps to produce 2 spikes back in $\sigma_{(\alpha,1)}$. Since the running time is affected by which symbols are requested from the environment, we consider the worst case in which the component will always request the maximum k spikes. Hence, for the partitioned input string w_α of length n, the first $n-1$ symbols will take $(k+4)(n-1)$ and the last symbol of w_α requires $k+3$ steps to process. Thus, the total running time of a halting and accepting computation is $(kn + 4n - 1)$ steps.

To analyse the communication complexity between the two components, for each symbol $b_r \in \{b_1, ..., b_k\}$, note that the number of times each component communicates to the other component through the external synapse is r, which is k at most. Both components send spikes at the same time, hence the number of communication steps per symbol, b_r, processed is at most k. Hence, for an input of length n, there are at most kn communications between the two components. Therefore, $ComN(\Delta) = kn$.

For each communication step, two intercomponent communication rules (one for each component). Therefore, $ComR(\Delta) = 2kn$.

For each character b_r, the component sends $r + 2$ spikes, and both components send to each other at the same time. For the last symbol of the input substring, w_α, the number of spikes sent is $r + 3$. Taking again the maximum possible number of spikes per character, which is k, the number of spikes required to be communicated is $(k+2)(n-1)+k+3 = kn + 2n + 1$ per component. Hence the total weight of the communication is $2(kn + 2n + 1) = 2kn + 4n + 2$. Therefore, $ComW(\Delta) = 2kn + 4n + 2$.

In summary, we have that $ComN(\Delta) = kn$, $ComR(\Delta) = 2kn$, and $ComW(\Delta) = 2kn + 4n + 2$.

□

3.2 3-component Nonbalanced and Nonhomogeneous SNdP

PROPOSITION 2. *$L_{ww} = \{ww \mid w \in \{b_1, b_2, \cdots, b_k\}^n, n \geq 1\}$ can be recognized by a 3-component SNdP system, Δ, with nonbalanced input partition and nonhomogeneous components, with $ComN(\Delta) = ComR(\Delta) = (k+1)n$, and $ComW(\Delta) = (k+2)n$, and Δ uses a total of 13 neurons across all components.*

Proof: Let $\Sigma_k = \{b_1, \cdots, b_k\}$. The input string, say $w' \in \Sigma_k^{2n}$, $n \geq 1$, will be partitioned into two substrings in this way: $w' = w_1 w_2$, where $||w_1| - |w_2|| \leq 1$. The first component, Π_1, takes in w_1 while the second and third components, Π_2 and Π_3, takes in w_2. w_2 is further partitioned between these two components in this way: Let $w_2 = x_1 x_2 \cdots x_n$ where $k = |w_2|$. If n is odd, $y_1 = x_1 x_3 x_5 \cdots x_n$ and $y_2 = x_2 x_4 \cdots x_{n-1}$. If n is even, $y_1 = x_1 x_3 x_5 \cdots x_{n-1}$ while $y_2 = x_2 x_4 \cdots x_n$.

$$\Delta = (\{a\},\ \Pi_1,\ \Pi_2,\ \Pi_3,\ \{((1,4),(3,1)),\ ((1,7),(2,1)),\ ((2,2),(1,3)),\ ((3,2),(1,6))\})$$

Figure 3 shows the 3-component SNdP system using nonbalanced partition and nonhomogeneous components recognizing L_{ww}, where x is $\lceil \frac{|w|}{2} \rceil$, around the half length of w, and y is $|w| - x$, around the other half of w, and $1 \leq r \leq k$. r spikes represents the character b_r from the alphabet of w, $\{b_1, b_2, \cdots, b_k\}$.

The first component, Π_1, takes in w_1 while the second component takes y_1 and the third component takes y_2. The input of the system is nonbalanced, so the components do not necessarily have to take inputs of equal length from the environment. And since the system is nonhomogeneous, the system does not need to have components with the same neurons. The first neuron on modules 2 and 3 is a counter that is initialized to have $2x$ and $2y$ spikes, respectively.

At the beginning of the computation, $\sigma_{(1,7)}$ has two spikes. It uses these two spikes to send a spike each to $\sigma_{(1,1)}$ and $\sigma_{(2,1)}$. This causes the counter in $\sigma_{(2,1)}$ to decrement and send a spike to $\sigma_{(2,2)}$ using the rule $a(aa)^+/a \to a$ and for $\sigma_{(1,1)}$ to get an input from the environment using its request rule $a/\lambda \leftarrow a^r$. $\sigma_{(1,1)}$ then sends these spikes one by one to $\sigma_{(1,2)}$ and $\sigma_{(1,5)}$ using either the rules $(aa)a^+/a \to a$ if it is not yet the last spike or the rule $a^2 \to a^2$ if it is the last spike, consuming all spikes in the neuron in the process. However, since $\sigma_{(1,5)}$ has no spikes at first, it will just forget if it gets one spike. When $\sigma_{(1,1)}$ sends its first spike to $\sigma_{(1,2)}$, $\sigma_{(2,2)}$ gets a spike from the environment. Both $\sigma_{(1,2)}$ and $\sigma_{(2,2)}$ then sends the spikes they get one by one to $\sigma_{(1,3)}$, $\sigma_{(1,2)}$ using either the rule $a^3/a \to a$ or the rule $a^4 \to a^2$ which consumes all spikes in $\sigma_{(1,2)}$ if $\sigma_{(1,1)}$ sends two spikes, meaning it received the last character. $\sigma_{(2,2)}$ sends spikes in the same manner as $\sigma_{(1,1)}$. If $\sigma_{(1,3)}$ receives the same spikes from both $\sigma_{(1,2)}$ and $\sigma_{(2,2)}$, it just forgets these spikes using either $a^2 \to \lambda$ or $a^4 \to \lambda$. Otherwise, it uses $a^3 \to a$ to send a spike to $\sigma_{(1,8)}$ to cause it to have infinite spikes with $\sigma_{(1,9)}$.

When $\sigma_{(1,1)}$ sends its last two spikes, it will give two spikes to $\sigma_{(1,5)}$ and will cause $\sigma_{(1,2)}$ to use all of its spikes to send two spikes to $\sigma_{(1,4)}$ which will be used to send one spike to $\sigma_{(1,1)}$ and $\sigma_{(3,1)}$. Neuron $\sigma_{(3,1)}$ assumes its function as a counter and begins decrementing, sneding a spike to $\sigma_{(3,2)}$ using the rule $a(aa)^+/a^3 \to a$. This causes computations similar to the one done between the first and second components but this time using a different set of neurons. Because this time it is $\sigma_{(1,5)}$ that has two spikes and $\sigma_{(1,2)}$ has none, the latter will just forget if it obtains one spike while the former will do the computations of sending either one or two spikes to $\sigma_{(1,6)}$ and $\sigma_{(1,7)}$. As with $\sigma_{(1,3)}$, if $\sigma_{(1,6)}$ receives the same spikes, it will forget them, but receiving different spikes causes it to send to $\sigma_{(1,8)}$, which, again, causes infinite spikes in the system.

These computations go on, with Π_1 alternating between computing with Π_2 and Π_3. If both counters finish counting down to 0, the computation will halt, signaling that the whole ww string has been read and accepted.

The algorithm makes use of three components: one with nine neurons, and two with two neurons each. For every character of input, it takes 1 step for either $\sigma_{(1,4)}$ or $\sigma_{(1,7)}$ to send a signal to the counter in one of the neurons, 1 step to decrement from the counter and for $\sigma_{(1,1)}$ to get the input from the environment, then 1 step to send to $\sigma_{(1,2)}$ and for $\sigma_{(2,2)}$ or $\sigma_{(3,2)}$ to get an input from the environment. Then, the input spikes are decremented one by one and sent to either $\sigma_{(1,3)}$ or $\sigma_{(1,6)}$. For a character r where $1 \leq r \leq n$, with n as the maximum number of steps. As the last spike of the character is used, the neuron that will send to the counter is also reinitialized. This shows that it takes a maximum of $n+3$ steps per character. Thus, for any input string of length with any length n, It will take at most $kn+3n$ steps to reach a halting computation.

To analyse the communication complexity between the three components, note that the number of times a component sends to the other depends on the number of spikes r it receives from the environment, which, again, is k at most. There is also one communication between components sent when decrementing from the counter. Thus, the number of communications used between three components is at most $k+1$. And for an input string $w' = w_1 w_2$ of length $2n$, there are $(k+1)n$ communications between the three components. Hence, $ComN(\Delta) = (k+1)n$. Additionally, for each communication step, only one intercomponent communication rule is used. Note here that since Π_2 and Π_3 alternates between the odd and even symbols of w_2, only one is actively communicating with Π_1 at a time. Hence, $ComR(\Delta) = ComN(\Delta) = (k+1)n$.

Observe that for a spike r, the $\sigma_{(2,2)}$ or $\sigma_{(3,2)}$ sends 1 spike to the first component for up to $r-1$ times and then sends 2 for the last spike. This means that each character causes one component to send $(r-1)+2 = r+1$ spikes. One spike is also sent between communications to decrement the counter per character. And because the maximum r for each character is k, that means per character, the total weight of

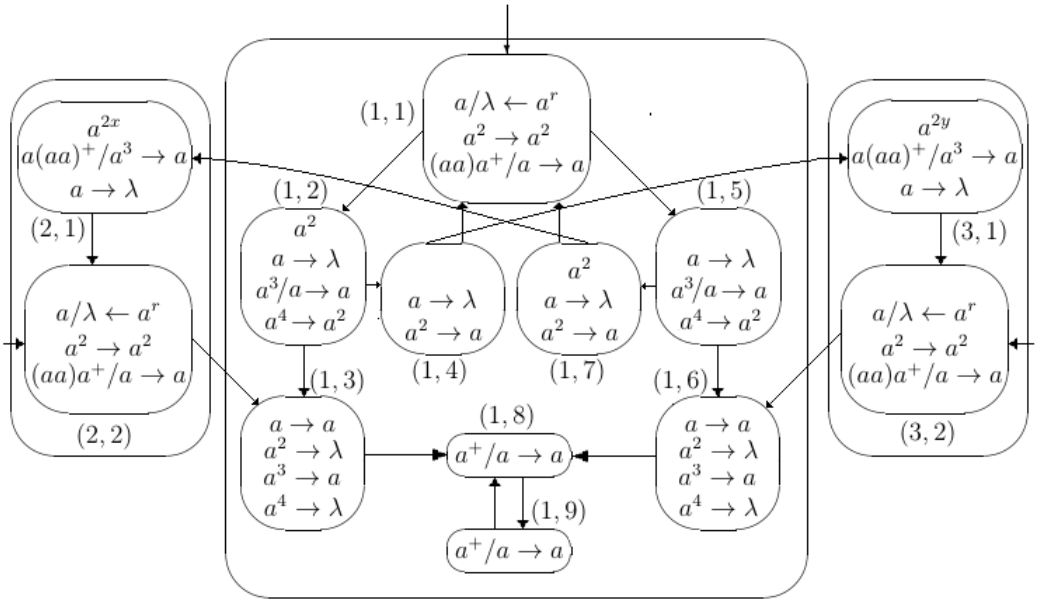

Figure 3: The SNdP system for the proof of Proposition 2

communication is $k+1+1 = k+2$. Then, for a string w' of length $2n$, the total communication cost of $kn+2n$ spikes. Hence, $ComW(\Delta) = kn+2n$.

In summary, we have that $ComN(\Delta) = ComR(\Delta) = (k+1)n$, and $ComW(\Delta) = (k+2)n$.

□

3.3 Comparison Between Models

Looking at the three models, i.e., the balanced and nonhomogeneous system, the balanced and homogeneous systems, and the nonbalanced nonhomogeneous system, we can compare them in terms of their communication costs as well as the neurons needed per component.

For the balanced and nonhomogeneous system presented in [11], we see that per character, the second component communicates with the first component r times because it sends the spikes one by one, and afterwards, the first component communicates with the second component once. Since $r \leq k$, then the number of communications between the components in a given computation is $k+1$. For a string of length n, this means that there are $kn+n$ communications. When the weights of the communications are considered, we see that the second component sends to the first component 1 spike $r-1$ times, and 3 spikes once per character for a total of $r-1+3 = r+2$ spikes per character with k as the maximum r, and it does this for n characters. The second component then sends for a communication cost of $n(k+2) = kn+2n$. The first component, on the other hand, sends 3 spikes for the first $n-1$ characters and just 1 spike for the last character, which means it sends $3(n-1)+1 = 3n-2$ spikes in total. Adding the two, the total communication cost is $kn+5n-2$. The number of time steps needed to finished computing one character is $k+2$. For n character, this means a running time of $kn+2n$.

The balanced and nonhomogeneous system, then, has a $ComN(\Delta) = kn+k$ and $ComW(\Delta) = kn+5k-2$. The balanced and homogeneous system has $ComN(\Delta) = kn$ and $ComW(\Delta) = 2kn+4k$ and the nonbalanced and nonhomogeneous system has $ComN(\Delta) = \lfloor \frac{n}{2} \rfloor k$ and $ComW(\Delta) = \lfloor \frac{n}{2} \rfloor (k+2)$. Here, we see that the nonbalanced and nonhomogeneous system gives the least communication cost at almost a quarter of the balanced and nonhomogeneous system, while the balanced and homogeneous system gives the most, which is almost double that of the balanced and homogeneous system.

In terms of the number of neurons of the components, the balanced and nonhomogeneous used the least neurons, with 7 neurons on the first component and 1 neuron on the second component for a total of 8 neurons. The balanced and homogeneous system, being homogeneous, used the same number of neurons per component, which is 8, for a total of 16 neurons.

Table 2 summarizes the analyses between the models.

4 FINAL REMARKS

This work was only able to model SNdP that are either balanced and homogeneous or nonbalanced and nonhomogeneous. As shown in Table 1, constructing an SNdP system with a nonbalanced partition and homogeneous components is still left open. Some things to note for future reference are difficulties in communicating between components when they do not have inputs of equal lengths, as well as the difficulty

Table 1: Balance and Homogeneity of SNdP Systems recognizing L_{ww}

	Balanced Input Partition	Nonbalanced Input Partition
Homogeneous Components	Section 3.1	Open Problem
Nonhomogeneous Components	In [11]	Section 3.2

Table 2: Comparison between models

	Number of Communications	Weighted Cost	Number of Neurons	Running Time
2-component Balanced Nonhomogeneous[11]	$nk + k$	$nk + 5k - 2$	8	$kn + 2n$
2-component Balanced Homogeneous (in Section 3.1)	kn	$2kn + 4k$	16	$kn + 4n - 1$
3-component Nonbalanced Nonhomogeneous (in Section 3.2)	$nk + k$	$kn + 2k$	13	$kn + 3n$

of making sure that neurons have the same rules and are able to use them effectively.

In this work, we measured the communication cost of the presented SNdP systems as defined in Section 2. Based on Table 2, compared to the work in [11], the SNdP in Section 3.1 has a lower number of communications but higher weighted communication cost and running time. The weighted communication cost, number of neurons, and running time of the SNdP in Section 3.2 are also lower than that of the SNdP in Section 3.1, even if it has more components. This may indicate that SNdP would benefit more from having nonhomogeneous components, having a "central" component doing most of the processing, with auxilliary input components.

This work mainly looked into SNdP systems that recognize the language $\{ww \mid w \in \{b_1, b_2, \cdots, b_n\}^n, n \geq 1\}$. However, this can be generalized further into the family of languages $\{w^k, k \geq 2 \mid w \in \{b_1, b_2, \cdots, b_n\}^n, n \geq 2\}$. Some ideas to do this is to follow a pattern similar to the components of the homogeneous and balanced SNdP systems in this work. There can be one component for each w and the spikes counter and rules can be adjusted accordingly to accommodate more spikes being exchanged in between components.

In the introduction of dP systems[19], dP systems should have constant communication measures with respect ot the length of the input. This is important for another property of dP systems, known as parallelizability. Parallelizability can be viewed as a measure of how we scale-up the solution (in terms of number of components). Having a non-constant communication measure would make a dP system with more components less desirable, as most of the resource will go to communication, rather than the internal and independent computation of the individual components. For further future works, we would like to look into how we can construct a SNdP solution with communication measures that is not a function of the length of the input, similar to the dP systems in [1, 3, 4].

ACKNOWLEDGEMENTS

K. Buño is supported by the Atty. Raul C. Villanueva Professorial Chair since 2019 until present. F. Cabarle is supported by the Dean Ruben Garcia Professorial Chair since 2017 until present, and RLC grant 2019–2020. All authors also acknowledge support from the UPD CoE.

REFERENCES

[1] Henry Adorna, Linqiang Pan, and Bosheng Song. 2018. On Distributed Solution to SAT by Membrane Computing. *International Journal of Computers Communications & Control* 13, 3 (2018), 303–320. https://doi.org/10.15837/ijccc.2018.3.3217

[2] H. Adorna, G. Păun, and M.J. Pérez-Jiménez. 2010. On Communication Complexity in Evolution-Communication P systems. *Romanian Journal of Information Science and Technology* 13, 2 (2010), 113–130. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84868036820&partnerID=40&md5=edde040b73b4dafcd73f70c574826799

[3] Kelvin Buño and Henry Adorna. 2020. Distributed computation of a k P systems with active membranes for SAT using clause completion. *Journal of Membrane Computing* 2, 2 (01 Jun 2020), 108–120. https://doi.org/10.1007/s41965-020-00040-4

[4] Kelvin C. BuÃśo, Francis George C. Cabarle, Marj Darrel Calabia, and Henry N. Adorna. 2018. Solving the N-Queens problem using dP systems with active membranes. *Theoretical Computer Science* 736 (2018), 1 – 14. https://doi.org/10.1016/j.tcs.2017.12.013

[5] F.G.C. Cabarle, H.N. Adorna, and M.J. Pérez-Jiménez. 2016. Notes on spiking neural P systems and finite automata. *Natural Computing* 15, 4 (2016), 533–539. https://doi.org/10.1007/s11047-016-9563-4

[6] Francis George C. Cabarle, Henry Adorna, and Miguel A. Martinez-del Amor. 2011. An Improved GPU Simulator for Spiking Neural P Systems. In *Proceedings of the 2011 Sixth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA '11)*. IEEE Computer Society, USA, 262âĂŞ267. https://doi.org/10.1109/BIC-TA.2011.37

[7] Francis George C. Cabarle and Henry N. Adorna. 2012. Some Notes on Spiking Neural dP Systems and Petri Nets. In *Theory and Practice of Computation: Workshop on Computation: Theory and Practice Quezon City, Philippines, September 2011 Proceedings*, Shin-ya Nishizaki, Masayuki Numao, Jaime Caro, and Merlin Teodosia Suarez (Eds.). Springer Japan, Tokyo, 62–77. https://doi.org/10.1007/978-4-431-54106-6_6

[8] Haiming Chen, Mihai Ionescu, Tseren-Onolt Ishdorj, Andrei Păun, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2008. Spiking neural P systems with extended rules: universality and languages. *Natural Computing* 7, 2 (01 Jun 2008), 147–166. https://doi.org/10.1007/s11047-006-9024-6

[9] Erzsébet Csuhaj-Varjú. 2005. P Automata. In *Membrane Computing: 5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004, Revised Selected and Invited Papers*, Giancarlo Mauri, Gheorghe Păun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 19–35. https://doi.org/10.1007/978-3-540-31837-8_2

[10] Erzsébet Csuhaj-Varjú, Maurice Margenstern, György Vaszil, and Sergey Verlan. 2007. On Small Universal Antiport P Systems. *Theor. Comput. Sci.* 372, 2-3 (March 2007), 152–164. https://doi.org/10.1016/j.tcs.2006.11.023

[11] Mihai Ionescu, Gheorghe Păun, Mario J. Pérez-Jiménez, and Takashi Yokomori. 2011. Spiking Neural dP Systems. *Fundam. Inf.* 111, 4 (Dec. 2011), 423–436. http://dl.acm.org/citation.cfm?id=2361516.2361521

[12] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. 2006. Spiking Neural P Systems. *Fundam. Inf.* 71, 2,3 (Feb. 2006), 279–308. http://dl.acm.org.ezproxy.engglib.upd.edu.ph/citation.cfm?id=1227505.1227513

[13] Richelle Ann B. Juayong and Henry N. Adorna. 2020. A survey of results on evolution–communication P systems with energy. *Journal of Membrane Computing* 2, 1 (01 Mar 2020), 59–69. https://doi.org/10.1007/s41965-020-00034-2

[14] L.F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, and A. Riscos-Núñez. 2012. A P-lingua based simulator for spiking neural P systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7184 LNCS (2012), 257–281. https://doi.org/10.1007/978-3-642-28024-5_18

[15] T. Neary. 2010. On the computational complexity of spiking neural P systems. *Natural Computing* 9, 4 (2010), 831–851.

[16] L. Pan and G. PÇŐun. 2010. Spiking neural P systems: An improved normal form. *Theoretical Computer Science* 411, 6 (2010), 906–918. https://doi.org/10.1016/j.tcs.2009.11.010

[17] Gheorghe Paun. 2004. Further open problems in membrane computing. In *Proceedings of the Second Brainstorming Week on Membrane Computing, 354-365. Sevilla, ETS de Ingeniería Informática, 2-7 de Febrero, 2004*. Fénix Editora, 354 – 365.

[18] Gheorghe Păun. 2002. M*embrane Computing: An Introduction*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-56196-2

[19] Gheorghe Păun and Mario J. Pérez-Jiménez. 2010. Solving Problems in a Distributed Way in Membrane Computing: dP Systems. I*nternational Journal of Computers, Communications and Control* 5 (2010), 238–250. http://www.journal.univagora.ro/?page=article_details&id=408

[20] Gheorghe Păun and Mario J. Pérez-Jiménez. 2012. An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Science* 431 (2012), 4 – 12. https://doi.org/10.1016/j.tcs.2011.12.053

[21] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. 2010. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA.

[22] Andrew Chi-Chih Yao. 1979. Some Complexity Questions Related to Distributive Computing(Preliminary Report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing* (Atlanta, Georgia, USA) *(STOC '79)*. ACM, New York, NY, USA, 209–213. https://doi.org/10.1145/800135.804414

[23] Xiangxiang Zeng, Xingyi Zhang, and Linqiang Pan. 2009. Homogeneous Spiking Neural P Systems. *Fundamenta Informaticae* 97 (2009), 275–294. https://doi.org/10.3233/FI-2009-200 1-2.