Sequential Spiking Neural P Systems with Structural Plasticity Based on Max/Min Spike Number

Francis George C. Cabarle[1], Henry N. Adorna[1], Mario J. Pérez-Jiménez[2]

[1]Algorithms & Complexity Lab, Department of Computer Science
University of the Philippines Diliman
Diliman, 1101, Quezon City, Philippines;
[2]Department of Computer Science and AI
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
email: fccabarle@up.edu.ph, hnadorna@dcs.upd.edu.ph, marper@us.es

Abstract. Spiking neural P systems (in short, SNP systems) are parallel, distributed, and nondeterministic computing devices inspired by biological spiking neurons. Recently, a class of SNP systems known as SNP systems with structural plasticity (in short, SNPSP systems) were introduced. SNPSP systems represent a class of SNP systems that have dynamism applied to the synapses, i.e. neurons can use plasticity rules to create or remove synapses. In this work, we impose the restriction of sequentiality on SNPSP systems, using four modes: max, min, max-pseudo, and min-pseudo sequentiality. We also impose a normal form for SNPSP systems as number acceptors and generators. Conditions for (non)universality are then provided. Specifically, acceptors are universal in all modes, while generators need a nondeterminism source in two modes, which in this work is provided by the plasticity rules.

Key words: Membrane computing, Spiking neural P systems, Structural plasticity, Sequential systems, Turing universality

1 Introduction

Membrane computing, a branch of natural computing, aims to abstract ideas from living cells for computing use [25]. More recently, the actively investigated area of spiking neurons[1] was introduced in membrane computing as spiking neural P (in short, SNP) systems [8]. SNP systems consist of neurons (often drawn as ovals), and synapses between neurons (arcs between ovals). Neurons are placed in vertices of a directed graph: they process only one type of indistinct signal called a *spike* (denoted by the symbol a) by firing or applying *spiking rules*. SNP systems use time to encode information using spikes, e.g. the value $n = t' - t$

[1] See e.g. [13] and [14] and references therein.

is computed by neuron σ_1 if σ_1 produced consecutive spikes at steps t and t', where $t \neq t', t' > t$. Time therefore plays a crucial role in information processing, and is not merely a background for computation.

Many theoretical results on computability or computational complexity have been investigated in SNP systems[2] e.g., SNP systems are known to be Turing universal as: number generating or accepting devices in [8] and [28]; language generating devices in [5] and [21]; function computing devices in [21] and [24]. SNP systems have also been used to efficiently solve computationally hard problems in [12] and [22], trading space for time.

SNP systems often use parallel, nondeterministic, or synchronous features in their computations. These features are powerful "ingredients" for achieving Turing universality, and many works (with biological inspiration) have investigated restrictions by removing at least one of these features, e.g. deterministic SNP systems have been considered since [8], asynchronous SNP systems as in [4], [6], and [30], and sequential SNP systems as in [6], [7], [10], and [31]. In SNP systems, neurons apply their rules in parallel but at most one rule is applied once in each neuron. Other ways of applying rules have also been considered, such as in [9][21] where rules were applied in a maximally parallel (i.e. exhaustive) way. A generalized rule application was introduced in [35], where neurons can nondeterministically choose the number of times a rule is applied. More neuroscience ideas and motivations have also been included, hence many variants of SNP systems have been introduced, e.g. the use of weighted synapses [20], neuron division and budding [22], the use of astrocytes [23], and anti-spikes [18].

Other biologically inspired restrictions imposed on SNP systems and its variants are the notions of *simplicity* and *homogeneity*: a neuron is simple if it precisely has one rule; an SNP system is simple if all neurons are simple; an SNP system is homogeneous if all of its neurons precisely have the same set of rules; Simple SNP systems usually require more neurons to achieve universality due to the simplicity of their neurons [31][32]. Homogeneous SNP systems can become more "compact", i.e. they can require less neurons, due to the fact that they can have more than one rule in a neuron as in [11] and [33]. Both notions can have interesting biological and computational interpretations, for theoretical and practical use: the connectivity of the neurons in the system is important for achieving a certain level of computing power; also, neurons do not need to be very complex (in the case of simple systems) or do not need to be very varied (in the case of homogeneous systems).

Another restriction imposed on SNP systems and its variants is the type(s) of rule(s) present in a neuron: *standard* (spiking) rules (at each step, at most one spike is sent from a σ_1 to a σ_2); *extended* (spiking) rules (at each step, more than one spike can be sent from a σ_1 to a σ_2); spiking rules with delay (if σ_1 sends a spike at step t, σ_2 receives the spike at $t + d, d \geq 1$); forgetting rules (rules that remove spikes from the system). Extended rules allow for more compact systems in terms of neuron count, due to the ability to produce more than one spike each step as in [5] and [24]. Both forgetting rules and rules with delays were used

[2] An overview in [29] and the SNP systems chapter in [28].

in [8], while in [19] for example, it was shown that universality is still achieved given the normal form of having no forgetting rules and rules with delays.

The SNP system variants we consider in this work are SNP systems with *structural plasticity*, in short, SNPSP systems.[3] Structural plasticity refers to any changes in the connectivity between neurons, i.e. synapses can be created or removed. Synapses can also be *rewired*, i.e. if synapse (a, b) exists between neurons σ_a and σ_b, σ_a can remove (a, b) and create novel synapse (a, c) to neuron σ_c instead. SNPSP systems also provide a response to an open problem in [29] where "dynamism" is only applied to synapses, in contrast to both neuron and (implicit) synapse dynamism for example in [22]. SNPSP systems represent the class of SNP systems that explicitly focus on synapse graph dynamism only, since the collection of neurons in the system remains static. Other than standard rules, the only other type of rule in SNPSP systems are plasticity rules: rules that allow neurons to create, remove, or rewire synapses.

In this work, the restriction we impose on SNPSP systems is sequentiality. Specifically, and as in [7], we *induce* sequential operation based on the number of spikes stored in a neuron. For example, in the max sequentiality or *maxs* mode, if neurons σ_a, σ_b, and σ_c contain 1, 3, and 2 spikes at the same step respectively, only σ_b is allowed to apply its rule. If however σ_c also stored 3 spikes, only one among σ_b or σ_c will apply its rule (nondeterministically chosen). Our results also exhibit (biologically and computationally) two interesting features, also known as a normal form, as in [19] and [31]: a neuron has two rules (the maximum per neuron) if and only if it is purely plastic, i.e. only contains plasticity rules; if a neuron has a standard rule, then it is simple. These two features are interesting because while certain biological neurons seem to have more specific or complex functions (e.g. they create or remove synapses), other neurons seem more simple or generic (e.g. they simply function as spike repositories or relays). More related normal forms in this work include: almost simple SNP or ASSNP systems in [31], where the system has only one neuron that is not simple; simple and homogeneous SNP systems with astrocytes or SHSNPA systems in [23], where SHSNPA systems maintained universality despite being simple and homogeneous, due to the use of additional neuroscience structures (the astrocytes).

Another distinction of SNPSP systems, again biologically and computationally motivated, is an alternative source for nondeterminism. A more common source of nondeterminism in SNP systems is rule-level (in short, nd_{rule}): if more than one rule can be applied in a step, only one is chosen nondeterministically. In SNPSP systems, there is instead synapse-level nondeterminism (in short, nd_{syn}): selecting which synapses a neuron will create or remove. Our results show the conditions, with or without nd_{syn} (among other levels, except nd_{rule}) by which our systems become (non)universal. Our systems, under the normal form mentioned above, are investigated under four modes: max, min, max-pseudo, and min-pseudo sequentiality.

We organize this work as follows: in Section 2 we provide some prerequisites for our results; Section 3 introduces the syntax and semantics of SNPSP systems;

[3] Introduced in [1] and improved and extended in [2].

our (non)universality results for the four modes are presented in Section 4; We provide final remarks and further directions in Section 5.

2 Preliminaries

It is assumed that the readers are familiar with the basics of membrane computing[4] and formal language theory (available in many monographs). We only briefly mention notions and notations which will be useful throughout the paper. 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 $V^+ = V^* - \{\lambda\}$. If $V = \{a\}$, we simply write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$. If $a \in V$, we write $a^0 = \lambda$, and we write the language associated with a regular expression E as $L(E)$.

In proving computational universality, we use the notion of register machines. A register machine is a construct $M = (m, I, l_0, l_h, R)$, where m is the number of registers, I is a finite set of instruction labels, l_0 is the start label, l_h is the halt label, and R is a finite set of instructions. Every label $l_i \in I$ uniquely labels only one instruction in R. Instructions in R have the following forms, given the value n stored in register r:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$, increase n by 1, then nondeterministically go to either l_j or l_k;
- $l_i : (\texttt{SUB}(r), l_j, l_k)$, if $n \geq 1$, then subtract 1 from n and go to l_j, otherwise perform no operation on r and go to l_k;
- $l_h : \texttt{HALT}$, the halt instruction.

We say M computes or generates a number n as follows: M starts with all its registers empty. The register machine then applies its instructions starting with the instruction labeled l_0. Without loss of generality, we assume that l_0 labels an ADD instruction, and that the content of the output register (e.g. register 1) is never decremented, only added to during computation, i.e. no SUB instruction is applied to it. If M reaches the halt instruction l_h, then the number n stored during this time in register 1 is said to be computed by M. We denote the set of all numbers computed by M as $N(M)$. It was proven that register machines compute all sets of numbers computed by a Turing machine, therefore characterizing the set NRE [17].

Register machines can also work in an accepting mode: a number n is stored in register 1 of M (all other registers are empty). If M halts, given this initial configuration, then n is said to be accepted or computed by M. It is also known that if M is deterministic, where an ADD instruction is simply written as $l_i : (\texttt{ADD}(r), l_j)$, M still characterizes NRE. As a convention, when comparing the power of two number generating devices D_1 and D_2, the number zero is ignored, i.e. $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$. This convention

[4] A good introduction is [26] and the P systems webpage at `http://ppage.psystems.eu/`, with a handbook in [28].

corresponds to the common practice in language and automata theory to ignore the empty string.

3 Spiking neural P systems with structural plasticity

In this section we define SNP systems with structural plasticity. Initial motivations and results are included in the seminal paper in [8]. A spiking neural P system with structural plasticity (SNPSP systems) of degree $m \geq 1$ is a construct of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out)$, where:

- $O = \{a\}$ is the singleton alphabet (a is called spike);
- $\sigma_1, \ldots, \sigma_m$ are neurons of the form $(n_i, R_i), 1 \leq i \leq m$; $n_i \geq 0$ indicates the initial number of spikes in σ_i; R_i is a finite rule set of σ_i with two forms:
  1. Spiking rule: $E/a^c \to a$, where E is a regular expression over O, $c \geq 1$;
  2. Plasticity rule: $E/a^c \to \alpha k(i, N)$, where E is a regular expression over O, $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $k \geq 1$, and $N \subseteq \{1, \ldots, m\} - \{i\}$;
- $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$ (synapses between neurons);
- $out \in \{1, \ldots, m\}$ indicates the output neuron.

Given neuron σ_i (we also say neuron i or simply σ_i) we denote the set of neuron labels with σ_i as their presynaptic (postsynaptic, resp.) neuron as $pres(i)$, i.e. $pres(i) = \{j | (i, j) \in syn\}$ (as $pos(i) = \{j | (j, i) \in syn\}$, resp.). Spiking rule semantics in SNPSP systems are similar with SNP systems in [8]. We do not use forgetting rules or rules with delays. Spiking rules are applied as follows: If neuron σ_i contains b spikes and $a^b \in L(E)$, with $b \geq c$, then a rule $E/a^c \to a \in R_i$ can be applied. Applying such a rule means consuming c spikes from σ_i, thus only $b - c$ spikes remain in σ_i. Neuron i sends one spike to every neuron with a label in $pres(i)$ at the same step as rule application. If a rule $E/a^c \to a$ has $L(E) = \{a^c\}$, we simply write this as $a^c \to a$.

Plasticity rules are applied as follows. If at step t we have that σ_i has $b \geq c$ spikes and $a^b \in L(E)$, a rule $E/a^c \to \alpha k(i, N) \in R_i$ can be applied. The set N is a collection of neurons to which σ_i can create a synapse to, or remove a synapse from, using the applied plasticity rule. The rule consumes c spikes and performs one of the following, depending on α:

- If $\alpha := +$ and $N - pres(i) = \emptyset$, or if $\alpha := -$ and $pres(i) = \emptyset$, then there is nothing more to do, i.e. c spikes are consumed but no synapses are created or removed. Notice that with these semantics, a plasticity rule can replace a forgetting rule, i.e. the former can be used to consume spikes without producing any spike.
- for $\alpha := +$, if $|N - pres(i)| \leq k$, deterministically create a synapse to every σ_l, $l \in N_j - pres(i)$. If however $|N - pres(i)| > k$, nondeterministically select k neurons in $N - pres(i)$, and create one synapse to each selected neuron.
- for $\alpha := -$, if $|pres(i)| \leq k$, deterministically delete all synapses in $pres(i)$. If however $|pres(i)| > k$, nondeterministically select k neurons in $pres(i)$, and delete each synapse to the selected neurons.

If $\alpha := \pm$ ($\alpha := \mp$, respectively), create (delete, resp.) synapses at step t and then delete (create, resp.) synapses at step $t+1$. Only the application priority of synapse creation or deletion is changed, but the semantics of synapse creation and deletion remain the same as when $\alpha \in \{+, -\}$. Neuron i can receive spikes from t until $t+1$, but σ_i can only apply another rule at time $t+2$.

An important note is that for σ_i applying a rule with $\alpha \in \{+, \pm, \mp\}$, creating a synapse always involves a sending of one spike when σ_i connects to a neuron. This single spike is sent at the time the synapse creation is applied, i.e. whenever synapse (i, j) is created between σ_i and σ_j during synapse creation, we have σ_i immediately transferring one spike to σ_j.

Let t be some step during a computation: we say σ_i is *activated* at step t if there is at least one $r \in R_i$ that can be applied (i.e. the stored spikes in σ_i satisfy the regular expression of r); we say σ_i is *active* at t if for some step $t' < t$, σ_i was activated by applying $r \in R_i$, and is currently fulfilling the requirements for the application of r. Also, we say σ_i is *simple* if $|R_i| = 1$. Given an SNPSP system Π, we can have the following levels of nondeterminism: *system-level*, if given at least two activated neurons, Π chooses exactly one neuron to fire; *rule-level*, if at least one neuron has at least two rules with regular expressions E_1 and E_2 such that $E_1 \neq E_2$ and $L(E_1) \cap L(E_2) \neq \emptyset$; *synapse-level*, if at least one neuron can nondeterministically create or delete a synapse;

By default SNP and SNPSP systems are locally sequential (at most one rule is applied per neuron) but globally parallel (all activated neurons must apply a rule). Note that the application of rules in neurons are synchronized, i.e. a global clock is assumed and if a neuron can apply a rule then it must do so. A configuration of an SNPSP system is based on (a) distribution of spikes in neurons, and (b) neuron connections based on *syn*. For some step t, we can represent: (a) as $\langle s_1, \ldots, s_m \rangle$ where $s_i, 1 \leq i \leq m$, is the number of spikes contained in σ_i; for (b) we can derive *pres*(i) and *pos*(i) from *syn*, for a given σ_i. The initial configuration therefore is represented as $\langle n_1, \ldots, n_m \rangle$, with the possibility of a disconnected graph. A computation is defined as a sequence of configuration transitions, from an initial configuration, and following rule application semantics. A computation halts if the system reaches a halting configuration, i.e. a configuration where no rules can be applied.

A result of a computation can be defined in several ways in SNP systems literature. For sequential SNPSP systems, we use the following as in [11]: we only consider the first two consecutive steps t_1 and t_2 that σ_{out} spikes. Their difference minus one, i.e. the number $n = (t_2 - t_1) - 1$, is said to be computed by Π. We refer to Π as generator, if Π computes in this manner. Π can also work as an acceptor, as follows: n spikes are stored in a defined neuron in an initial configuration. Π then accepts n if the computation halts.

The following two features are used in our systems as the normal form: (i) only purely plastic neurons (i.e. neurons with plasticity rules only) have at most 2 rules (the maximum in any neuron of the system), and (ii) neurons with standard rules are simple. We denote the family of sets computed by nondeterministic SNPSP systems (in the mentioned normal form) as generators as $N_{2,gen}SNPSP^{\gamma}$: subscript 2 indicates the first 2 spikes of σ_{out} as the result; we replace 2 and *gen* with *acc* for acceptors; we have the mode $\gamma \in \{maxs, maxps, mins, minps\}$ (details given shortly); Given an SNPSP system Π, we denote the computed set by Π as $N_m(\Pi)$, $m \in \{gen, acc\}$. To further parametrize our results in this work we have the following: $+syn_k$ ($-syn_j$, resp.) where at most k (j, resp.) synapses are created (deleted, resp.) each step; $nd_\beta, \beta \in \{syn, rule, sys\}$ indicate additional levels of nondeterminism source; $rule_m$ indicates at most m rules (either standard or plasticity) per neuron; Since our results for k and j for $+syn_k$ and $-syn_j$ are equal, we write them instead in the compressed form $\pm syn_k$, where $\pm$ in this sense is not the same as when $\alpha := \pm$.

Induced sequentiality is as follows: In one step of an SNPSP system Π, only the neuron with the maximum (minimum, resp.) number of stored spikes becomes activated in *maxs* (*mins*, resp) mode. If there is more than one such neuron, then exactly one is chosen among them (i.e. nd_{sys} is used). In *max pseudosequential* or *maxps* (*min pseudosequential* or *minps*, resp.) mode, Π allows all neurons with the maximum (minimum, resp.) stored spikes to become activated (i.e. no nd_{sys}). In all four modes, the system is sequential, induced by the global maximum (minimum, resp.) spike number. The nd_{sys} was used in [7] (denoted as nondeterminism at the level of the system) and [10]. In [7], they denoted nd_{rule} as nondeterminism at the level of neurons. Note that if γ for example takes the value *maxs*, nd_{sys} is implied so that nd_{sys} is omitted from writing.

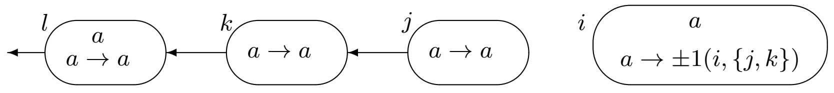


Fig. 1. An SNPSP system Π_{ej}.

To illustrate the notions and semantics in SNPSP systems, we take as an example the SNPSP system Π_{ej} in Figure 1, and describe its computations. The initial configuration is as follows: spike distribution is $\langle 1, 0, 0, 1\rangle$ for the neuron order σ_i, σ_j, σ_k, σ_l, respectively; $syn = \{(j,k), (k,l)\}$; output neuron is σ_l, indicated by the outgoing synapse to the environment.

Let t be the step when σ_i and σ_l become activated, i.e. they can apply their rules because the regular expressions are satisfied. At t then we have the following: σ_l applies its spiking rule and sends the first (out of two) spike to the environment; since $k = 1 < |\{j,k\}|$, σ_i nondeterministically selects whether to create synapse (i,j) or (i,k); if (i,j) ((i,k), resp.) is created, a spike is sent from σ_i to σ_j (σ_k, resp.) at t due to the immediate sending of a spike during synapse creation, and σ_l fires for the second and final time at $t+3$ ($t+2$, resp.); also, if (i,j) is created then $syn' := syn \cup \{(i,j)\}$, otherwise $syn'' := syn \cup \{(i,k)\}$. At $t+1$, σ_i deletes the synapse created at t (since $\alpha := \pm$), and we have syn again. By definition, σ_i and σ_l become activated at t, while only σ_i is active at

$t+1$ (while it deletes a synapse). The output of Π_{ej} is either $(t+3-t)-1$, or $(t+2-t)-1$, so that $N_2(\Pi_{ej}) = \{1,2\}$. Computations of Π_{ej} generating $\{1\}$ and $\{2\}$ are given in Table 1 and Table 2, respectively, where (!) indicates the spiking of the output neuron.

time	σ_i	σ_j	σ_k	σ_l	syn
t_0	1	0	0	1	syn
t	0	0	1	0(!)	$syn'' = syn \cup \{(i,k)\}$
$t+1$	0	0	0	1	syn
$t+2$	0	0	0	0(!)	syn

Table 1. Computation of Π_{ej} generating $\{1\}$, with $syn = \{(j,k),(k,l)\}$.

time	σ_i	σ_j	σ_k	σ_l	syn
t_0	1	0	0	1	syn
t	0	1	0	0(!)	$syn' = syn \cup \{(i,j)\}$
$t+1$	0	0	1	0	syn
$t+2$	0	0	0	1	syn
$t+3$	0	0	0	0(!)	syn

Table 2. Computation of Π_{ej} generating $\{2\}$, with $syn = \{(j,k),(k,l)\}$.

4 Main results

In this section we consider SNPSP systems as acceptors or generators of sets of numbers in four modes: *maxs*, *maxps*, *mins*, and *minps*. It is known from [2] that SNPSP systems as generators or acceptors are universal, operating in the "usual way" in membrane computing, i.e. neurons operate in parallel. Using the four modes, we show in this section that we can still achieve universality despite the restriction of induced sequentiality and the normal form in Section 3.

4.1 Sequential SNPSP systems based on max spike number

We first consider acceptors and generators in *maxs* and *maxps* modes: in *maxs* mode, at most one neuron with the most stored spikes is nondeterministically chosen to fire; in *maxps* mode, all neurons with the most number of spikes are allowed to fire. Our (non)universality results under a normal form are as follows (with details of the parameters provided shortly): for *maxs*, acceptors and generators are universal; for *maxps*, acceptors are universal while generators can become universal with nondeterminism provided in this work by nd_{syn}.

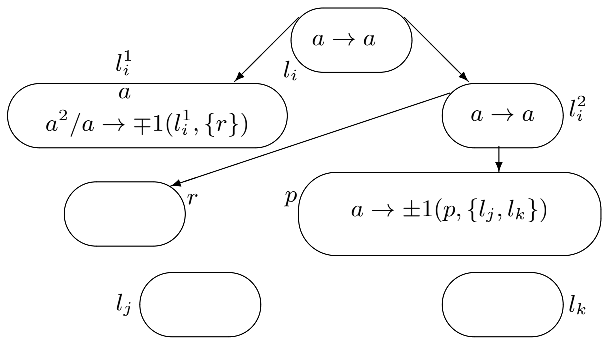
Max sequentiality ($maxs$) mode

Theorem 1 $NRE = N_{2,gen}SNPSP^{maxs}(rule_2, \pm syn_k, nd_{syn}), k \geq 1$.

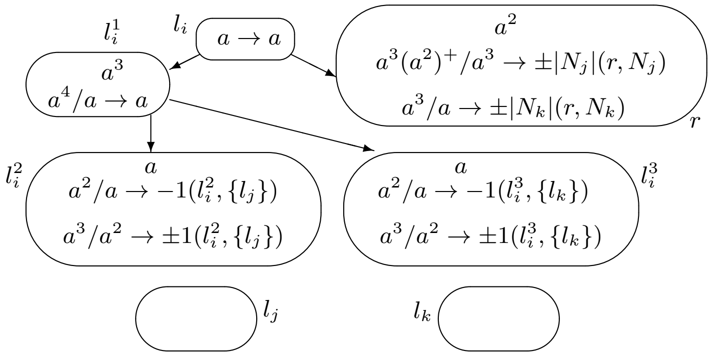
Proof. It is enough to simulate a register machine M by means of an SNPSP system Π, with restrictions given in the Theorem statement. Before we construct Π, we provide a general description of the computation as follows: each register r in M is associated with a σ_r in Π. If r stores the value n, then σ_r stores $2n+2$ spikes. We will construct modules for Π that will simulate addition, subtraction, and halting operations in M. If a rule with label l_i is applied in M, then the associated neuron σ_{l_i} in Π is activated in order to simulate the operation performed by l_i. In the initial configuration, all neurons are empty, except neuron σ_{l_0} which contains one spike and is associated with the initial instruction l_0 of M.

Module ADD: The module simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$ is shown in Figure 2. Once σ_{l_i} is activated it sends one spike each to $\sigma_{l_i^1}$ and $\sigma_{l_i^2}$. At this step $\sigma_{l_i^1}$ stores two spikes, while $\sigma_{l_i^2}$ stores one spike. Let step t be the step where $\sigma_{l_i^1}$ is activated: $\sigma_{l_i^1}$ applies its only rule so it consumes one spike and deletes the synapse (l_i^1, r) if it exists. At step $t+1$ we have the following: $\sigma_{l_i^1}$ becomes an active neuron to continue the application of its plasticity rule applied at t, while $\sigma_{l_i^2}$ is now the only activated neuron. At $t+1$, $\sigma_{l_i^2}$ sends one spike each to σ_r and σ_p, while $\sigma_{l_i^1}$ creates synapse (l_i^1, r), thus sending one spike to σ_r.

The spikes stored in σ_r at $t+1$ increase by two, and σ_p is activated at the next step. We omit further details of σ_r at this point, but we note that no rule of σ_r is activated while σ_r stores an even number of spikes. Since σ_p is activated, it will nondeterministically select whether to create synapse (p, l_j) or (p, l_k). Once σ_p selects one synapse to create, either σ_{l_j} or σ_{l_k} is activated. The ADD module correctly simulates the addition operation: the spikes stored in σ_r are increased by two, simulating the increase of the value stored in r by one; then, either σ_{l_j} or σ_{l_k} is activated nondeterministically, simulating the nondeterministic application of either l_j or l_k in M.


$a \to a$
l_i
l_i^1
a
$a^2/a \to \mp 1(l_i^1, \{r\})$
$a \to a$
l_i^2
r
p
$a \to \pm 1(p, \{l_j, l_k\})$
l_j
l_k
Fig. 2. Module ADD simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$ in the proof of Theorem 1.

Module SUB: The module simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$ is shown in Figure 3. This is the only module in which σ_r becomes activated. We define N_k (N_j, resp.) as $N_k = \{l_i^3 | l_i \text{ is a } \texttt{SUB} \text{ instruction on } r\}$ ($N_j = \{l_i^2 | l_i \text{ is a } \texttt{SUB} \text{ instruction on } r\}$, resp.). Once activated, σ_{l_i} sends one spike each to $\sigma_{l_i^1}$ and σ_r. Let t be the step when σ_{l_i} and σ_r collect four and $2n+3$ spikes, respectively. At t we can have both $\sigma_{l_i^1}$ and σ_r activated, but due to *maxs* mode, only one neuron becomes activated. Depending on the value of n, either $\sigma_{l_i^1}$ or σ_r spikes, and we have the following two possible cases:



l_i^1
l_i
$a \to a$
a^2
$a^3(a^2)^+/a^3 \to \pm|N_j|(r, N_j)$
$a^3/a \to \pm|N_k|(r, N_k)$
r
a^3
$a^4/a \to a$
l_i^2
a
$a^2/a \to -1(l_i^2, \{l_j\})$
$a^3/a^2 \to \pm1(l_i^2, \{l_j\})$
a
$a^2/a \to -1(l_i^3, \{l_k\})$
$a^3/a^2 \to \pm1(l_i^3, \{l_k\})$
l_i^3
l_j
l_k

Fig. 3. Module SUB simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$ in the proof of Theorem 1.

Case $n = 0$: This case corresponds to σ_r storing three spikes at t, so $\sigma_{l_i^1}$ becomes activated. In this case $\sigma_{l_i^1}$ consumes one spike and sends one spike each to $\sigma_{l_i^2}$ and $\sigma_{l_i^3}$ at t. At $t+1$, σ_r is the activated neuron with the most stored spikes. Recall that in this case M must deterministically execute l_k, and this operation is simulated as follows: the rule of σ_r with $L(E) = \{a^3\}$ is applied and $|N_k|$ synapses are created and then deleted (in the next step). For the specific case where $N_k = \{l_i^3\}$ so that $|N_k| = 1$, σ_r consumes one spike and synapse (r, l_i^3) is created and then deleted. The spikes stored in σ_r return to two, simulating the "no operation" when register r stores $n = 0$. Now at $t+2$, $\sigma_{l_i^3}$ stores three spikes and applies its rule with $L(E) = \{a^3\}$, so synapse (l_i^3, l_k) is created and then deleted. Before σ_{l_k} is activated, *maxs* mode dictates that $\sigma_{l_i^2}$ must apply its rule with $L(E) = \{a^2\}$ to delete any synapse from it.

Case $n \geq 1$: This case corresponds to σ_r storing at least five spikes at t, so σ_r is the neuron that fires. In this case M must deterministically execute l_j which is simulated as follows: the rule of σ_r with $L(E) = \{a^3(a^2)^+\}$ is applied and $|N_j|$ synapses are created and then deleted (in the next step). For the specific case where $N_j = \{l_i^2\}$ so that $|N_j| = 1$, σ_r consumes three spikes and synapse (r, l_i^2) is created and then deleted. The stored spikes in σ_r after removing three spikes return to an even count, simulating the decrease of the value stored in r by one. At $t+1$, $\sigma_{l_i^1}$ is the activated neuron with the most stored spikes, and it sends one spike each to $\sigma_{l_i^2}$ and $\sigma_{l_i^3}$. At $t+2$, $\sigma_{l_i^2}$ now stores three spikes and applies its rule with $L(E) = \{a^2\}$, so synapse (l_i^2, l_j) is created and then deleted. Before σ_{l_j} is activated, $\sigma_{l_i^3}$ must apply its rule with $L(E) = \{a^2\}$ to delete any synapse from it.

In both cases, the number of stored spikes in $\sigma_{l_i^1}$, $\sigma_{l_i^2}$, and $\sigma_{l_i^3}$ are restored to the original number of spikes before σ_{l_i} was activated, i.e. the same SUB module can be used again. We consider now the general case where more than one $l_i : (\mathtt{SUB}(r), l_j, l_k)$ operation is performed on the same register r. We need to be certain that there is no interference with several SUB modules acting on the same σ_r.

By definition of N_j and N_k, only the correct SUB module simulating l_x on r is allowed to continue because either l_x^2 or l_x^3 receives three spikes. The other neuron, including neurons l_y^2 and l_y^3 for l_y also operating on r, only receive two spikes: *maxs* mode will nondeterministically allow all such neurons to remove one spike from their two spikes using their plasticity rule with $L(E) = \{a^2\}$. Hence, the SUB module correctly simulates the subtraction operation: spikes stored in σ_r are reduced by two, simulating the decrease by one of the nonzero value stored in r, then σ_{l_j} is activated to simulate l_j in M; if r stored the value zero, then the two spikes in σ_r are restored and σ_{l_k} is activated to simulate l_k in M.

Module FIN: Once M arrives at l_h, M halts its computation and is simulated by the FIN module illustrated in Figure 4. Neuron l_h is activated, sending one spike to σ_{out}. Let t be the step when σ_{out} first spikes to the environment. At t, σ_{out} has six spikes and consumes two, reducing its spikes to four. At $t+1$, if register 1 stores the value $n = 1$, then σ_1 contains $2n + 3 = 5$ spikes. Due to *maxs* mode, both σ_{out} and σ_1 can be activated but only σ_1 is allowed to spike. At $t+1$ we have σ_1 reducing its spikes to three, and deleting the nonexisting synapse $(1, out)$. At $t+2$, σ_{out} spikes (because it is the only activated neuron) for the second and final time to the environment. The interval between the first and second spikes minus one is $((t+2) - t) - 1 = 1$, exactly the value stored in register r. Note that the four spikes in σ_1 at $t+2$ do not allow σ_1 to become activated.

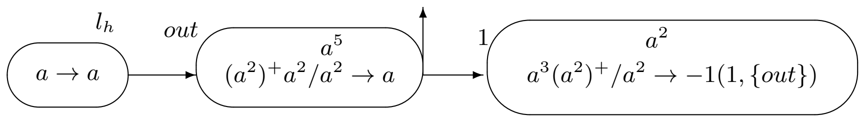


Fig. 4. Module FIN in the proof of Theorem 1.

If register r stores $n \geq 2$, then σ_1 contains at least seven spikes. Again, σ_1 has more spikes than σ_{out}, and σ_1 reduces its spikes by two each step. The second spike of σ_{out} is produced at step $t+n+1$, so that the interval between the first and second spikes minus one is $((t+n+1) - t) - 1 = n$, exactly the value stored in r. The Theorem parameters are satisfied in every neuron of every module: at most two rules, at most $k \geq 1$ synapses created (deleted), and nd_{sys} and nd_{syn} exist. This completes the proof. $\square$

Remarks: The idea of an active neuron, used in our proofs, is from [7] and their activated neuron and implicit active neuron: In one step of their ADD module in $maxs$ mode, a neuron is activated while another neuron with delay is active. In our ADD module, we apply a plasticity rule instead of a rule with delay. If we further restrict the system and remove nd_{syn}, we still achieve universality with a minor trade-off: $k \geq 2$ instead of $k \geq 1$.

Theorem 2 $NRE = N_{2,gen}SNPSP^{maxs}(rule_2, \pm syn_k), k \geq 2$.

Proof. In the proof for Theorem 1, only the ADD module in Figure 2 has nd_{syn}. We use in this case the new ADD module in Figure 5, which is a modified version of the ADD module in Figure 2. Since only nd_{sys} remains, and σ_{p_1} and σ_{p_2} can be activated at the same step, only one of them will fire: if σ_{p_1} (σ_{p_2}, resp.) fires, using its plasticity rule with $L(E) = \{a\}$ it sends one spike to σ_{l_j} (σ_{l_k}, resp.) and σ_{p_2} (σ_{p_1}, resp.). Before σ_{l_j} (σ_{l_k}, resp.) is activated, σ_{p_2} (σ_{p_1}, resp.) uses its rule with $L(E) = \{a^2\}$ to remove its two spikes and delete synapse (p_2, r) ((p_1, r), resp.). Notice however that the synapse $(m, r), m \in \{p_1, p_2\}$ never exists, and r in this case can be replaced by any neuron that neuron m will never have a synapse to. All Theorem parameters are satisfied: only nd_{sys} exist, and in this case $k \geq 2$ from the new ADD module. $\square$

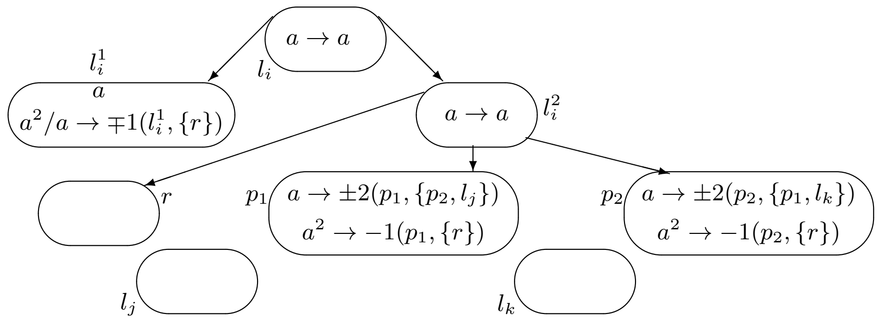


Fig. 5. Module ADD simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$ in the proof of Theorem 2.

Theorem 3 $NRE = N_{acc}SNPSP^{maxs}(rule_2, \pm syn_k), k \geq 1$.

Proof. Since a register machine M working as an acceptor characterizes NRE using only addition instructions of the form $l_i : (\texttt{ADD}(r), l_j)$, we modify the ADD module in Figure 5 as follows: we remove σ_{p_1}, σ_{p_2}, σ_{l_k}, and add synapse (l_i^2, l_j); We store $2n + 2$ spikes in σ_1, associated with n (the value to be accepted or rejected) stored in register 1 of M. We use the SUB module in Figure 3, and $k \geq 1$ instead of $k \geq 2$. The FIN module is not required: we accept n if σ_{l_h} is activated (meaning M has reached l_h) and reject n otherwise. $\square$

Max pseudo-sequentiality (*maxps*) mode

Theorem 4 $NRE = N_{2,gen}SNPSP^{maxps}(rule_2, \pm syn_k, nd_{syn}), k \geq 1$.

Proof. All modules in the proof of Theorem 1 can be used in *maxps* mode. In the SUB module in Figure 3 in particular, if $\sigma_{l_i^2}$ ($\sigma_{l_i^3}$, resp.) has three spikes, then $\sigma_{l_i^3}$ ($\sigma_{l_i^2}$, resp.) and every neuron with a label in $N_j - \{l_i^2\}$ ($N_k - \{l_i^3\}$, resp.) only have two spikes: these neurons with two spikes remove their spikes in parallel by applying their rule with $L(E) = \{a^2\}$. □

Remarks: In the case of deterministic generators, without a source of nondeterminism, they only generate singleton sets given an initial configuration (hence they are not universal) i.e. $NRE \neq D_{gen}SNPSP^{maxps}(rule_*, \pm syn_*)$. Similar to deterministic register machines as acceptors, we can have universality if the previous result is considered using SNPSP systems as acceptors instead.

Theorem 5 $NRE = D_{acc}SNPSP^{maxps}(rule_2, \pm syn_k), k \geq 1$.

Proof. This result holds since all modules in Theorem 3 are deterministic. □

4.2 Sequential SNPSP systems based on min spike number

Next we consider acceptors and generators in *mins* and *minps* modes: *mins* mode dictates that at most one neuron (with the least number of spikes stored) is nondeterministically chosen to become activated step; for *minps* mode, all the neurons with the least number of spikes stored become activated. Our (non)universality results under a normal form are as follows (with details of the parameters provided shortly): acceptor and generator systems in *mins* mode are universal; generator systems in *minps* mode are not universal; generator systems with nd_{syn}, or acceptor systems in *minps* mode, are universal.

Min sequentiality (*mins*) mode

Theorem 6 $NRE = N_{2,gen}SNPSP^{mins}(rule_2, \pm syn_k), k \geq 2$.

Proof. We construct modules of an SNPSP system Π which simulates a register machine M. Modules of Π will again perform addition, subtraction, and halting operations corresponding to the same operations in M. If register r stores the value n, then σ_r stores $2n + 2$ spikes. Simulating instruction l_i means σ_{l_i} in a module is activated, so that the module performs the operation of l_i.

Module ADD: The ADD module is illustrated in Figure 6. After σ_{l_i} becomes activated, it sends one spike each to σ_r and $\sigma_{l_i^1}$. Due to *mins* mode, only $\sigma_{l_i^1}$ is allowed by the system to apply a rule because it has only one spike (σ_r has at least three spikes). After $\sigma_{l_i^1}$ spikes, one spike is sent to σ_r, σ_{p_1}, and σ_{p_2}. In this way, two spikes are added to σ_r simulating the increment in register r. The *mins* mode will either allow σ_{p_1} or σ_{p_2} to become activated first: if σ_{p_1} (σ_{p_2}, resp.) becomes activated first, it creates two synapses so that σ_{l_j} and σ_{p_2} (σ_{l_k} and σ_{p_1}, resp.) receive one spike each; Before σ_{l_j} (σ_{l_k}, resp.) is allowed to apply a rule, *mins* mode dictates that σ_{p_2} (σ_{p_1}, resp.) apply its rule first because σ_{p_2} (σ_{p_1}, resp.) only has two spikes. Once σ_{p_2} (σ_{p_1}, resp.) removes its two spikes, σ_{l_j} (σ_{l_k}, resp.) can proceed to simulating instruction l_j (l_k, resp.). The `ADD` instruction is correctly simulated in Π.

Module SUB: The SUB module is illustrated in Figure 7. After σ_{l_i} spikes, $\sigma_{l_i^1}$ and σ_r each contain two and $2n+3$ spikes, respectively. Again, *mins* mode allows $\sigma_{l_i^1}$ to spike before σ_r. Let t be the step that $\sigma_{l_i^1}$ spikes, i.e. the step when $\sigma_{l_i^1}$ becomes activated: at t, $\sigma_{l_i^1}$ deletes all synapses from itself; at $t+1$, $\sigma_{l_i^1}$ remains active, then creates synapses and sends one spike each to $\sigma_{l_i^2}$ and $\sigma_{l_i^3}$; Also at $t+1$, σ_r becomes activated. As in Theorem 1, we define the sets N_j and N_k as follows: $N_k = \{l_i^3 | l_i$ is a `SUB` instruction on $r\}$ ($N_j = \{l_i^2 | l_i$ is a `SUB` instruction on $r\}$, resp.).

If $r \geq 1$ ($r = 0$, resp.), then σ_r creates a synapse and sends one spike each to every neuron with a label in N_j (N_k, resp.) at $t+1$. Also, if $r = 0$ then σ_r will contain $3 - 1 = 2$ spikes again, otherwise if $r \geq 1$ then σ_r will contain $2n + 3 - 3 = 2n$ spikes again. At $t+2$, $\sigma_{l_i^2}$ ($\sigma_{l_i^3}$, resp.) contains two spikes, while $\sigma_{l_i^3}$ and every neuron with a label in $N_j - \{l_i^2\}$ ($\sigma_{l_i^2}$ and every neuron with a label in $N_k - \{l_i^3\}$, resp.) only contain one spike. At $t+2$, *mins* mode dictates that these neurons containing one spike must remove their spikes, using their rule with $\alpha := -$. At $t+3$, σ_{l_j} (σ_{l_k}, resp.) is activated by $\sigma_{l_i^2}$ ($\sigma_{l_i^3}$, resp.) because it now contains three spikes. The `SUB` instruction is correctly simulated in Π, and no interference occurs among several `SUB` instructions operating on the same register r.

Module FIN: The FIN module is illustrated in Figure 8. Once σ_{l_h} is activated, σ_{out} then contains four spikes. At the next step, σ_{out} is activated and sends the first spike to the environment (we denote this step as t). At $t+1$, σ_1 contains at least five spikes, since $n \geq 1$. It takes $n-1$ steps for σ_1, using its rule with $\alpha := -$, to reduce its spikes to five if $n \geq 2$. Once σ_1 stores five spikes, it creates a synapse and sends one spike to σ_{out} at $t+n$. At $t+n+1$, σ_{out} sends a spike to the environment for the second and final time. The value $((t+n+1)-t)-1) = n$ is computed, which is precisely the value stored in register 1 of M. Note that the second spike sent by σ_{out} to σ_1 never activates any rule in σ_1.

Clearly, only nd_{sys} exists, at most two rules exist in each neuron, and at least $k = 2$ synapses are created each step. This completes the proof. □

Theorem 7 $NRE = N_{acc}SNPSP^{mins}(rule_2, \pm syn_k), k \geq 2.$

Proof. To simulate an acceptor register machine M using addition instructions of the form l_i : (`ADD`$(r), l_j)$, we modify the ADD module in Figure 6 as follows: we remove σ_{p_1}, σ_{p_2}, σ_{l_k}, and add synapse (l_i^1, l_j). We store $2n+2$ spikes in σ_1, associated with the value n in register 1 of M. The SUB module is the module in Figure 7, and a FIN module is not required. We accept n if σ_{l_h} becomes activated, and reject n otherwise. □

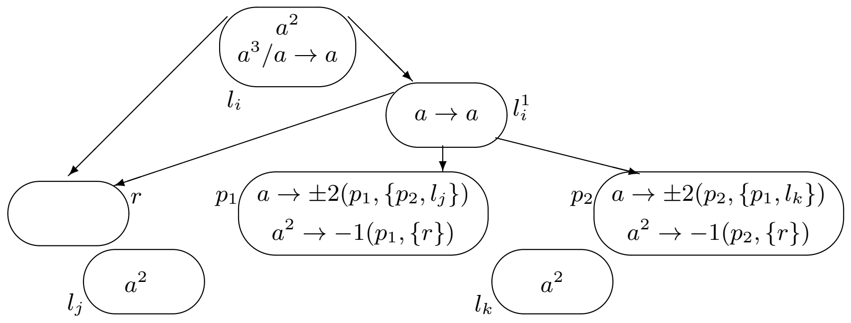
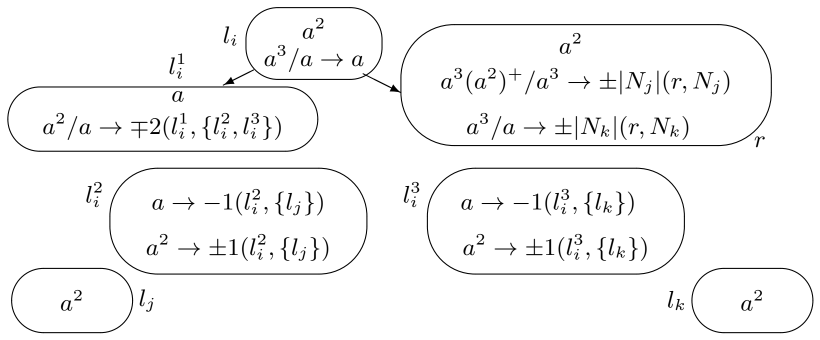
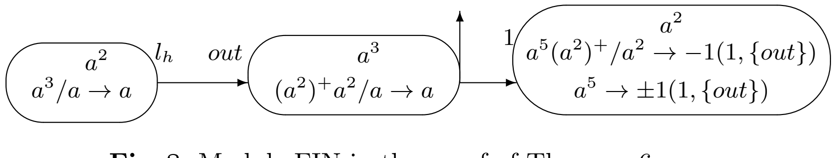


Fig. 6. Module ADD simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$ in the proof of Theorem 6.



l_i
a^2
$a^3/a \to a$
l_i^1
a
$a^2/a \to \mp 2(l_i^1, \{l_i^2, l_i^3\})$
a^2
$a^3(a^2)^+/a^3 \to \pm|N_j|(r, N_j)$
$a^3/a \to \pm|N_k|(r, N_k)$
r
l_i^2
$a \to -1(l_i^2, \{l_j\})$
$a^2 \to \pm 1(l_i^2, \{l_j\})$
l_i^3
$a \to -1(l_i^3, \{l_k\})$
$a^2 \to \pm 1(l_i^3, \{l_k\})$
a^2
l_j
l_k
a^2

Fig. 7. Module SUB simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$ in the proof of Theorem 6.



a^2
$a^3/a \to a$
l_h
out
a^3
$(a^2)^+a^2/a \to a$
1
a^2
$a^5(a^2)^+/a^2 \to -1(1, \{out\})$
$a^5 \to \pm 1(1, \{out\})$

Fig. 8. Module FIN in the proof of Theorem 6.

Min pseudo-sequentiality (*minps*) mode

Without a source of nondeterminism, deterministic SNPSP system as generators in *minps* mode are not universal, i.e. $NRE \neq D_{gen}SNPSP^{minps}(rule_*, \pm syn_*)$. This nonuniversality parallels the nonuniversality of deterministic generators in *maxps* mode. As in Theorem 4 for generators in *maxps* mode, generators in *minps* mode can become universal using nd_{syn}, as the next result shows.

Theorem 8 $NRE = N_{2,gen}SNPSP^{minps}(rule_2, \pm syn_k, nd_{syn}), k \geq 2.$

Proof. We modify the ADD module in Figure 6, to remove nd_{sys} and maintain only nd_{syn}, as follows: Replace σ_{p_1} and σ_{p_2} with a single neuron σ_p, and add synapse (l_i^1, p); the rule set of σ_p is $R_p = \{a \to \pm 1(p, \{l_j, l_k\})\}$.

The SUB and FIN modules in Figure 7 and 8, respectively, can be used in *minps* mode. In particular, the SUB module is still correct because if $\sigma_{l_i^2}$ ($\sigma_{l_i^3}$, resp.) contains two spikes, then $\sigma_{l_i^3}$ and every neuron with a label in $N_j - \{l_i^2\}$ ($\sigma_{l_i^2}$ and every neuron with a label in $N_k - \{l_i^3\}$, resp.) contain one spike each. The neurons that contain one spike each must remove their spikes in parallel, before the next instruction is correctly simulated. □

Theorem 9 $NRE = N_{acc}SNPSP^{minps}(rule_2, \pm syn_k), k \geq 2.$

Proof. All modules in the proof of Theorem 7 can be used in *minps* mode. □

5 Final remarks

We have shown that SNPSP systems as acceptors are more powerful than as generators, since the former do not need any source of nondeterminism. SNPSP systems maintain Turing universality despite the following restrictions: induced sequentiality, and under a normal form. The normal form is that purely plastic neurons have at most two rules (the maximum in the system), and neurons with standard rules are simple. Acceptors are universal in all four modes, while generators are not universal under *maxps* and *minps* modes. Replacing the commonly used nd_{rule} (e.g. in [8] and [7]) with nd_{syn} as a nondeterminism source, we are able to: allow generators in *maxps* or *minps* mode to become universal; reduce the rules in each neuron to at most two, while making the modules more compact, i.e. require fewer neurons.

Our results provide a family of *uniform modules*, i.e. at the expense of using nd_{syn}, our modules can be used in both *maxs* and *maxps* modes, or *mins* and *minps* modes. In [10], the problem of constructing a family of uniform ADD, SUB, and FIN modules that can be used in both *mins* and *minps* modes was open. Our work thus provides a positive hint to this open problem. The systems in [10] only contain spiking and forgetting rules, hence they use nd_{sys} or nd_{rule} instead of nd_{syn} as nondeterminism sources. Meanwhile, the sequential systems in [34] are investigated in the same four modes as in this work but with a different rule application semantic: instead of applying at most one rule once each step in a neuron (the common way of applying rules in SNP systems), the rules are applied in an exhaustive or maximally parallel way.

It is also interesting to consider the notion of homogeneous systems. In [11], two types of neurons are enough for universality: one for *maxs*, and another for *maxps*. It is then proved that homogeneous SNP (HSNP) systems as acceptors and generators are universal in both modes. Since neurons are homogeneous, the connectivity of the neurons is important. For SNPSP systems, perhaps a pseudo-homogeneous construction is reasonable, where only the set N is different among the rules of neurons.

Another restriction we investigate next, and which is related to sequentiality, is the removal of synchronization, i.e. asynchronous SNPSP systems [3]. However, much is left to be investigated for SNPSP systems. We here only list a few established results in membrane computing and SNP systems which can be applied to SNPSP systems: investigating the type of regular expression in standard or plasticity rules, e.g. bounded, unbounded, or general rules as in [4] and [30]; constructing small universal systems as in [24] and [32]; language generation as in [5]; processing (in)finite sequences as in [27]; exhaustive use of rules as in [21]. Further investigations on the structure or connectivity on SNP and SNPSP systems are also interesting. In [36] for example, a neural-like P system known as Axon P systems are proven to be universal despite having only a linear arrangement of the nodes. Such investigations can lead to further interesting results for building theoretical or practical neural systems with dynamic synapse graphs. Lastly, it is interesting to include SNPSP systems in the *P-Lingua* tool as in [15][16], which includes simulators for many SNP system variants. P-Lingua and other tools based based on P-Lingua, can then be used to perform experimental simulations based on SNPSP systems theory.

Acknowledgments

Cabarle is supported by a scholarship from the DOST-ERDT of the Philippines. Adorna is funded by a DOST-ERDT grant and the Semirara Mining Corp. professorial chair of the College of Engineering, UP Diliman. 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. The authors are thankful for the useful comments from three anonymous reviewers who helped improve this work.

References

1. Cabarle, F.G.C, Adorna, H., Ibo, N.: Spiking neural P systems with structural plasticity. Pre-proc. of 2nd Asian Conference on Membrane Computing, Chengdu, China, pp. 13 - 26, 4-7 November (2013)
2. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural P systems with structural plasticity (to appear). Neural Computing and Applications doi:10.1007/s00521-015-1857-4 (2015)

3. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Asynchronous Spiking Neural P Systems (to appear) 14th Unconventional Computation and Natural Computation, 31 Aug-04 Sep, Auckland, New Zealand (2015)
4. Cavaliere, M., Egecioglu, O., Woodworth, S., Ionescu, I., Păun, G.: Asynchronous spiking neural P systems: Decidability and undecidability. DNA 2008, LNCS vol. 4848, pp. 246 - 255 (2008)
5. Chen, H., Ionescu, M., Ishdorj, T.-I., Păun, A., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with extended rules: universality and languages. Natural Computing, vol. 7, pp. 147 - 166 (2008)
6. Ibarra, O.H., Woodworth, S.: Spiking neural P systems: some characterizations. FCT 2007, LNCS vol. 4639, pp. 23 - 37 (2007)
7. Ibarra, O.H., Păun, A., Rodríguez-Patón, A.: Sequential SNP systems based on min/max spike number. Theor. Com. Sci., vol. 410, pp. 2982 - 2991 (2009)
8. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae, vol. 71(2,3), pp. 279 - 308 (2006)
9. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems with an Exhaustive Use of Rules. J. of Unconventional Computing, vol. 3(2), pp. 135-153 (2007)
10. Jiang, K., Song, T., Pan, L.: Universality of sequential spiking neural P systems based on minimum spike number. Theor. Com. Sci., vol. 499, pp. 88 - 97 (2013)
11. Jiang, K., Song, T., Chen, W., Pan, L.: Homogeneous spiking neural P systems working in sequential mode induced by maximum spike number. J. of Computer Mathematics, vol. 90(4), pp. 831 - 844 (2013)
12. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving Numerical NP-Complete Problems with Spiking Neural P Systems. Eleftherakis et al. (Eds.): WMC8 2007, LNCS vol. 4860, pp. 336 - 352 (2007)
13. Maass, W., Bishop, C. (Eds.): Pulsed Neural Networks, MIT Press (1999)
14. Maass, W.: Computing with spikes, in: Special Issue on Foundations of Information Processing of TELEMATIK, vol. 8(1) pp. 32 - 36 (2002)
15. Macías-Ramos, F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua based Simulator for Spiking Neural P Systems. In: Gheorghe, M. et al. (eds.) CMC12 LNCS vol. 7184, pp. 257-281 (2012)
16. Macías-Ramos, F., Pérez-Jiménez, M.J., Song, T., Pan, L.: Extending Simulation of Asynchronous Spiking Neural P Systems in P-Lingua. Fundamenta Informaticae. vol. 136(3), pp. 253-267 (2015)
17. Minsky, M.: Computation: Finite and infinite machines. Englewood Cliffs, NJ: Prentice Hall (1967)
18. Pan, L., Păun, G.: Spiking neural P systems with anti-spikes. J. of Computers, Communication, & Control. vol. IV(3), pp 273 - 282 (2009)
19. Pan, L., Păun, G.: Spiking Neural P Systems: An improved normal form. Theor. Com. Sci. vol. 411(6), pp. 906 - 918 (2010)
20. Pan, L., Wang, J., Hoogeboom, H.J.: Spiking Neural P Systems with Weights. Neural Computation, vol. 22, pp. 2615 - 2646 (2010)
21. Pan, L., Zeng, X.: Small Universal Spiking Neural P Systems Working in Exhaustive Mode. IEEE Trans. NanoBiosci., vol. 10(2), pp. 99 - 105 (2011)
22. Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Science China Information Sciences. vol. 54(8) pp. 1596 - 1607 (2011)
23. Pan, L., Wang, J., Hoogeboom, J.H.: Spiking Neural P Systems with Astrocytes. Neural Computation vol. 24, pp. 805 - 825 (2012)

24. Păun, A., Păun, G.: Small universal spiking neural P systems. Biosystems, vol. 90, pp. 48 - 60 (2007)
25. Păun, G.: Computing with membranes. J. of Computer and System Science, vol. 61(1), pp. 108 - 143 (1999)
26. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002)
27. Păun, G., Pérez-Jiménez, M.J., Rozenberg, G.: Spike trains in spiking neural P systems. J. of Foundations of Computer Science, vol. 17(4), pp. 975 - 1002 (2006)
28. Păun, G., Rozenberg, G., Salomaa, A., Eds.: The Oxford Handbook of Membrane Computing. Oxford Univ. Press. (2009)
29. Păun, G., Pérez-Jiménez, M.J.: Spiking Neural P Systems. Recent Results, Research Topics. A. Condon et al. (eds.), Algorithmic Bioprocesses, Springer (2009)
30. Song, T., Pan, L., Păun, G.: Asynchronous spiking neural P systems with local synchronization. Information Sciences, vol. 219(10), pp. 197 - 207 (2013)
31. Song, T., Pan, L., Jiang, K., Song, B., Chen, W.: Normal Forms for Some Classes of Sequential Spiking Neural P Systems. IEEE Trans. NanoBiosci., vol. 12(3), pp. 1536 - 1241 (2013)
32. Zeng, X., Pan, L., Pérez-Jiménez, M.J.: Small universal simple spiking neural P systems with weights. Science China Information Sciences, vol. 57, pp. 1 - 11 (2014)
33. Zeng, X., Zhang, X., Pan, L.: Homogeneous Spiking Neural P Systems. Fundamenta Informaticae, vol. 97, pp. 1 - 20 (2009)
34. Zeng, X. Luo, B., Pan, L.: On Some Classes of Sequential Spiking Neural P Systems. Neural Computation, vol. 26, pp. 974-997 (2014)
35. Zhang, X., Wang, B., Pan, L.: Spiking neural P systems with a generalized use of rules. Neural Computation, vol. 26(12), pp. 2925-2943 (2014)
36. Zhang, X., Pan, L., Păun, A.: On the Universality of Axon P Systems. IEEE Trans. Neural Networks and Learning Systems, doi:10.1109/TNNLS.2015.2396940 (2015)