Asynchronous Spiking Neural P Systems with Structural Plasticity*

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
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
fccabarle@up.edu.ph, hnadorna@dcs.upd.edu.ph, marper@us.es

Summary. Spiking neural P (in short, SNP) systems are computing devices inspired by biological spiking neurons. In this work we consider SNP systems with structural plasticity (in short, SNPSP systems) working in the asynchronous (in short, *asyn* mode). SNPSP systems represent a class of SNP systems that have dynamic synapses, i.e. neurons can use plasticity rules to create or remove synapses. We prove that for *asyn* mode, bounded SNPSP systems (where any neuron produces at most one spike each step) are not universal, while unbounded SNPSP systems with weighted synapses (a weight associated with each synapse allows a neuron to produce more than one spike each step) are universal. The latter systems are similar to SNP systems with extended rules in *asyn* mode (known to be universal) while the former are similar to SNP systems with standard rules only in *asyn* mode (conjectured not to be universal). Our results thus provide support to the conjecture of the still open problem.

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

1 Introduction

Spiking neural P systems (in short, SNP systems) are parallel, distributed, and nondeterministic devices introduced into the are of membrane computing in [7]. Neurons are often drawn as ovals, and they process only one type of object, the *spike* signal represented by a. Synapses between neurons are the arcs between ovals: neurons are then placed on the vertices of a directed graph. Since their introduction, several lines of investigations have been produced, e.g. (non)deterministic

* An improved version of this article will appear at the 14th Unconventional Computation and Natural Computation (2015), Auckland, New Zealand.

computing power in [7][14]; language generation in [4]; function computing devices in [11]; solving computationally hard problems in [9]. Many neuroscience inspirations have also been included for computing use, producing several variants (to which the previous investigation lines are also applied), e.g. use of weighted synapses [16], neuron division and budding [9], the use of astrocytes [10]. Furthermore, many restrictions have been applied to SNP systems (and variants), e.g. asynchronous SNP systems as in [6], [3], and [15], and sequential SNP systems as in [6].

In this work the variant we consider are SNP systems with structural plasticity, in short, SNPSP systems. SNPSP systems were first introduced in [1], then extended and improved in [2]. The biological motivation for SNPSP systems is structural plasticity, one form of neural plasticity, and distinct from the more common functional (Hebbian) plasticity. SNPSP systems represent a class of SNP systems using plasticity rules: synapses can be created or deleted so the synapse graph is dynamic. The restriction we apply to SNPSP systems is asynchronous operation: imposing synchronization on biological functions is sometimes "too much", i.e. not alway realistic. Hence, the asynchronous mode of operation is interesting to consider. Such restriction is also interesting mathematically, and we refer the readers again to [6], [3], and [15] for further details.

In this work we prove that (*i*) asynchronous bounded (i.e. there exists a bound on the number of stored spikes in any neuron) SNPSP systems are not universal, (*ii*) asynchronous weighted (i.e. a positive integer weight is associated with each synapse) SNPSP systems, even under a normal form (provided below), are universal. The open problem in [3] whether asynchronous bounded SNP systems with standard rules are universal is conjectured to be false. Also, asynchronous SNP systems with extended rules are known to be universal [5]. Our results provide some support to the conjecture, since neurons in SNPSP systems produce at most one spike each step (similar to standard rules) while synapses with weights function similar to extended rules (more than one spike can be produced each step). This work is organized as follows: Section 2 provides preliminaries for our results; syntax and semantics of SNPSP systems are given in Section 3; our (non)universality results are given in Section 4. Lastly, 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 (a good introduction is [13] with recent results and information in the P systems webpage (`http://ppage.psystems.eu/`) and a recent handbook [14]) and formal language theory (available in many monographs). We only briefly mention notions and notations which will be useful throughout the paper.

We denote the set of positive integers as $\mathbb{N} = \{1, 2, \ldots\}$. 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 is a symbol in V, we write $a^0 = \lambda$ and we write the language generated by a regular expression E over V 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 the set of instruction labels, l_0 is the start label, l_h is the halt label, and R is the set of instructions. Every label $l_i \in I$ uniquely labels only one instruction in R. Register machine instructions have the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$, increase n by 1, then nondeterministically go to 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.

Given a register machine M, 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 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 the first (also the output) register 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 NRE [8]. A strongly monotonic register machine is one restricted variant: it has only one register which is also the output register. The register initially stores zero, and can only be incremented by 1 at each step. Once the machine halts, the value stored in the register is said to be computed. It is known that strongly monotonic register machines characterize $SLIN$, the family of length sets of regular languages.

3 Spiking neural P systems with structural plasticity

In this section we define SNP systems with structural plasticity. Initial motivations and results for SNP systems are included in the seminal paper in [7]. A *spiking neural P system with structural plasticity* (SNPSP system) 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: $E/a^c \rightarrow a$, where E is a regular expression over O, $c \geq 1$;
  2. Plasticity rule: $E/a^c \rightarrow \alpha k(i, N)$, 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);
- $out \in \{1,\ldots,m\}$ indicate 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 [7]. In this work we do not use forgetting rules (rules of the form $a^s \to \lambda$) or rules with delays of the form $E/a^c \to a; d$ for some $d \geq 1$. 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 label in $pres(i)$ at the same step as rule application. A nonzero delay d means that if σ_i spikes at step t, then neurons receive the spike at $t+d$. Spikes sent to σ_i from t to $t+d-1$ are lost (i.e. σ_i is closed), and σ_i can receive spikes (i.e. σ_i is open) and apply a rule again at $t+d$ and $t+d+1$, respectively. 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 connect to or disconnect from using the applied plasticity rule. The rule application 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 functions similar to 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 $\alpha \in \{\pm, \mp\}$: create (respectively, delete) synapses at step t and then delete (respectively, create) synapses at step $t+1$. Only the priority of application of synapse creation or deletion is changed, but the application is similar to $\alpha \in \{+,-\}$. Neuron i is always open 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 an *embedded* 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 σ_i *attaches* to σ_j using a synapse during synapse creation, we have σ_i immediately transferring one spike to σ_j.

Let t be a step during a computation: we say a σ_i is *activated* at step t if there is at least one $r \in R_i$ that can be applied; σ_i is *simple* if $|R_i| = 1$, with a nice biological and computing interpretation, i.e. some neurons do not need to be complex, but merely act as spike repositories or relays. We have the following nondeterminism levels: *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 initially Π has at least one σ_i with a plasticity rule where $k < |N - pres(i)|$; *neuron-level*, if at least one activated neuron with rule r can choose to apply its rule r or not (i.e. asynchronous).

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). The application of rules in neurons are usually synchronized, i.e. a global clock is assumed. However, in the asynchronous ($asyn$, in short) mode we release this synchronization so that neuron-level nondeterminism is implied. 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, or $syn = \emptyset$. 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. no rules can be applied and all neurons are open.

A result of a computation can be defined in several ways in SNP systems literature. For SNP systems in $asyn$ mode however, and as in [3] [5] [15], the output is obtained by counting the total spikes sent out by σ_{out} to the environment (in short, Env) upon reaching a halting configuration. We refer to Π as generator, if Π computes in this asynchronous manner. Π can also work as an acceptor but this is not given in this work.

For our universality results, the following simplifying features are used in our systems as the normal form: (i) plasticity rules can only be found in purely plastic neurons (i.e. neurons with plasticity rules only), (ii) neurons with standard rules are simple, and (iii) we do not use forgetting rules or rules with delays. We denote the family of sets computed by asynchronous SNPSP systems (under the mentioned normal form) as generators as $N_{tot}SNPSP^{asyn}$: subscript tot indicates the total number of spikes sent to Env as the result; Other parameters are as follows: $+syn_k$ ($-syn_j$, respectively) where at most k (j, resp.) synapses are created (deleted, resp.) each step; $nd_\beta, \beta \in \{syn, rule, neur\}$ 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 $\alpha := \pm$. A bound p on the number of spikes stored in any neuron of the system is denoted as $bound_p$. We omit nd_{neur} from writing since it is implied in $asyn$ mode.

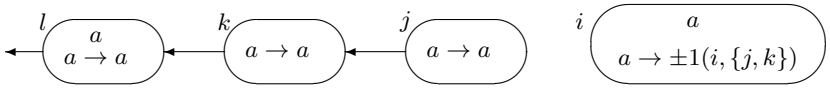

Fig. 1. An SNPSP system Π_{ej}.

To illustrate the notions and semantics in SNPSP systems, we take as an example the SNPSP system Π_{ej} of degree 4 in Fig. 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 Env.

Given the initial configuration, σ_i and σ_l can become activated. Due to *asyn* mode however, they can decide to apply their rules at a later step. If σ_l applies its rule before it receives a spike from σ_i, then it will spike to Env twice so that $N_{tot}(\Pi_{ej}) = \{2\}$. Since $k = 1 < |\{j, k\}|$ and $pres(i) = \emptyset$, 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.) due to the embedded sending of a spike during synapse creation. Let this be step t. If (i, j) is created then $syn' := syn \cup \{(i, j)\}$, otherwise $syn'' := syn \cup \{(i, k)\}$. At $t + 1$, σ_i deletes the created synapse at t (since $\alpha := \pm$), and we have syn again. Note that if σ_l does not apply its rule and collects two spikes (one spike from σ_i), the computation is aborted or blocked, i.e. no output is produced since $a^2 \notin L(a)$.

4 Main results

In this section we use at most two nondeterminism sources: nd_{neur} (in *asyn* mode), and nd_{syn}. Recall that in *asyn* mode, if σ_i is activated at step t so that an $r \in R_i$ can be applied, σ_i can choose to apply r or not. If σ_i did not choose to apply r, σ_i can continue to receive spikes so that for some $t' > t$, it is possible that: r can never be applied again, or some $r' \in R_i, r' \neq r$, is applied.

For the next result, each neuron can store only a bounded number of spikes (see for example [3][6][7] and references therein). In [6], it is known that bounded SNP systems with extended rules in *asyn* mode characterize $SLIN$, but it is open whether such result holds for systems with standard rules only. In [3], a negative answer was conjectured for the following open problem: are asynchronous SNP systems with standard rules universal? First, we prove that bounded SNPSP systems in *asyn* mode characterize $SLIN$, hence they are not universal.

Lemma 1 $N_{tot}SNPSP^{asyn}(bound_p, nd_{syn}) \subseteq SLIN, p \geq 1$.

Proof. Taking any asynchronous SNPSP system Π with a given bound p on the number of spikes stored in any neuron, we observe that the number of possible configurations is finite: Π has a constant number of neurons, and that the number of spikes stored in each neuron are bounded. We then construct a right-linear

grammar G, such that Π generates the length set of the regular language $L(G)$. Let us denote by $\mathcal{C}$ the set of all possible configurations of Π, with C_0 being the initial configuration. The right-linear grammar $G = (\mathcal{C}, \{a\}, C_0, P)$, where the production rules in P are as follows:

(1) $C \to C'$, for $C, C' \in \mathcal{C}$ if Π has a transition $C \Rightarrow C'$ in which the output neuron does not spike;
(2) $C \to aC'$, for $C, C' \in \mathcal{C}$ if Π has a transition $C \Rightarrow C'$ in which the output neuron spikes;
(3) $C \to \lambda$, for any $C \in \mathcal{C}$ in which Π halts.

Due to the construction of G, Π generates the length set of $L(G)$, hence the set is semilinear. □

Lemma 2 $SLIN \subseteq N_{tot}SNPSP^{asyn}(bound_p, nd_{syn}), p \geq 1$.

The proof is based on the following observation: A set Q is semilinear if and only if Q is generated by a strongly monotonic register machine M. It suffices to construct an SNPSP system Π with restrictions given in the theorem statement, such that Π simulates M. Recall that M has precisely register 1 only (it is also the output register) and addition instructions of the form $l_i : (\texttt{ADD}(1), l_j, l_k)$. The ADD module for Π is given in Fig. 2. Next, we describe the computations in Π.

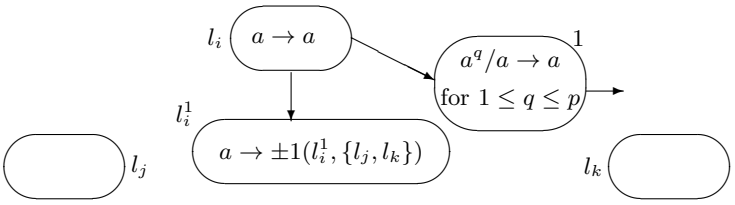

Fig. 2. Module ADD simulating $l_i : (\texttt{ADD}(1) : l_j, l_k)$ in the proof of Lemma 2.

Once ADD instruction l_i of M is applied, σ_{l_i} is activated and it sends one spike each to σ_1 and $\sigma_{l_i^1}$. At this point we have two possible cases due to *asyn* mode, i.e. either σ_1 spikes to Env before $\sigma_{l_i^1}$ spikes, or after. If σ_1 spikes before $\sigma_{l_i^1}$, then the number of spikes in Env is immediately incremented by 1. After some time, the computation will proceed if $\sigma_{l_i^1}$ applies its only (plasticity) rule. Once $\sigma_{l_i^1}$ applies its rule, either σ_{l_j} or σ_{l_k} becomes nondeterministically activated.

However, if σ_1 spikes after $\sigma_{l_i^1}$ spikes, then the number of spikes in Env is not immediately incremented by 1 since σ_1 does not consume a spike and fire to Env. The next instruction, either l_j or l_k, is then simulated by Π. Furthermore, due to *asyn* mode, the following "worst case" computation is possible: σ_{l_h} becomes activated (corresponding to l_h in M being applied, thus halting M) before σ_1 spikes. In this computation, M has halted and has applied an m number of ADD instructions since the application of l_i. Without loss of generality we can have the

arbitrary bound $p > m$, for some positive integer p. We then have the output neuron σ_1 storing m spikes. Since the rules in σ_1 are of the form $a^q/a \to a$, $1 \leq q \leq p$, σ_1 consumes one spike at each step it decides to apply a rule, starting with rule $a^m/a \to a$, until rule $a \to a$. Thus, Π will only halt once σ_1 has emptied all spikes it stores, sending m spikes to Env in the process.

The FIN module is not necessary, and we add σ_{l_h} without any rule (or maintain $pres(l_h) = \emptyset$). Once M halts by reaching instruction l_h, a spike in Π is sent to neuron l_h. Π is clearly bounded: every neuron in Π can only store at most p spikes, at any step. We then have Π correctly simulating the strongly monotonic register machine M. This completes the proof. □

From Lemma 1 and 2, we can have the next result.

Theorem 1 $SLIN = N_{tot}SNPSP^{asyn}(bound_p, nd_{syn}), p \geq 1$.

Next, in order to achieve universality, we add an additional ingredient to asynchronous SNPSP systems: weighted synapses. The ingredient of weighted synapses has already been introduced in SNP systems literature, and we refer the reader to [16] (and references therein) for computing and biological motivations. In particular, if σ_i applies a rule $E/a^c \to a^p$, and the weighted synapse (i, j, r) exists (i.e. the weight of synapse (i, j) is r) then σ_j receives $p \times r$ spikes.

It seems natural to consider weighted synapses for asynchronous SNPSP systems: since asynchronous SNPSP systems are not universal, we look for other ways to improve their power. SNPSP systems with weighted synapses (in short, WSNPSP systems) are defined in a similar way as SNPSP systems, except for the plasticity rules and the synapse set. Plasticity rules in σ_i are now of the form

$$E/a^c \to \alpha k(i, N, r),$$

where $r \geq 1$, and E, c, α, k, N are as previously defined. Every synapse created by σ_i using a plasticity rule with weight r receives the weight r. Instead of one spike sent from σ_i to a σ_j during synapse creation, $j \in N$, r spikes are sent to σ_j. The synapse set is now of the form

$$syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times \mathbb{N}.$$

We note that SNPSP systems are special cases of SNPSP systems with weighted synapses where $r = 1$, and when $r = 1$ we omit it from writing. In weighted SNP systems with standard rules, the weights can allow neurons to produce more than one spike each step, similar to having extended rules. In this way, our next result parallels the result that asynchronous SNP systems with extended rules are universal in [5]. However, our next result uses nd_{syn} with $asyn$ mode, while in [5] their systems use nd_{rule} with $asyn$ mode. We also add the additional parameter l in our universality result, where the synapse weight in the system is at most l. Our universality result also makes use of the normal form given in Section 3.

Theorem 2 $N_{tot}WSNPSP^{asyn}(rule_m, \pm syn_k, weight_l, nd_{syn}) = NRE, m \geq 9, k \geq 1, l \geq 3$.

Proof. We construct an asynchronous SNPSP system with weighted synapses Π, with restrictions given in the theorem statement, to simulate a register machine M. The general description of the simulation is as follows: each register r of M corresponds to σ_r in Π. If register r stores the value n, σ_r stores $2n$ spikes. Simulating instruction $l_i : (\texttt{OP}(r) : l_j, l_k)$ of M in Π corresponds to σ_{l_i} becoming activated. After σ_{l_i} is activated, the operation OP is performed on σ_r, and σ_{l_j} or σ_{l_k} becomes activated. We make use of modules in Π to perform addition, subtraction, and halting of the computation.

Module ADD: The module is shown in Fig. 3. At some step t, σ_{l_i} sends a spike to $\sigma_{l_i^1}$. At some $t' > t$, $\sigma_{l_i^1}$ sends a spike: the spike sent to σ_r is multiplied by two, while 1 spike is received by $\sigma_{l_i^2}$. For now we omit further details for σ_r, since it is never activated with an even number of spikes.

At some $t'' > t'$, $\sigma_{l_i^2}$ nondeterministically creates (then deletes) either (l_i^2, l_j) or (l_i^2, l_k). The chosen synapse then allows either σ_{l_j} or σ_{l_k} to become activated. The ADD module thus increments the contents of σ_r by 2, simulating the increment by 1 of register r. Next, only one among σ_{l_j} or σ_{l_k} becomes nondeterministically activated. The addition operation is correctly simulated.

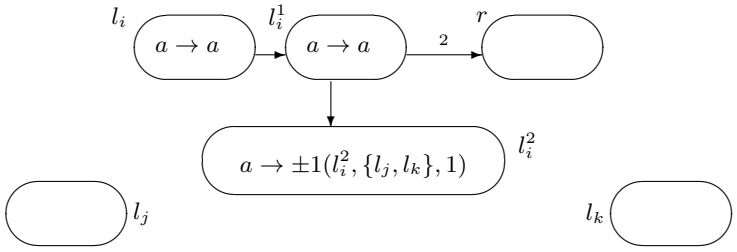

Fig. 3. Module ADD simulating $l_i : (\texttt{ADD}(r) : l_j, l_k)$ in the proof of Theorem 2.

Module SUB: The module is shown in Fig. 4. Let $|S_r|$ be the number of instructions with form $l_i : (\texttt{SUB}(r), l_j, l_k)$, and $1 \leq s \leq |S_r|$. $|S_r|$ is the number of SUB instructions operating on register r, and we explain in a moment why we use a size of a set for this number. Clearly, when no SUB operation is performed on r, then $|S_r| = 0$, as in the case of register 1. At some step t, σ_{l_i} spikes, sending 1 spike to σ_r, and $4|S_r| - s$ spikes to $\sigma_{l_i^1}$ (the weight of synapse (l_i, l_i^1)).

$\sigma_{l_i^1}$ has rules of the form $a^p \to -1(l_i^1, \{r\}, 1)$, for $3|S_r| \leq p < 8|S_r|$. When one of these rules is applied, it performs similar to a forgetting rule: p spikes are consumed and deletes a nonexisting synapse (l_i^1, r). Since $\sigma_{l_i^1}$ received $4|S_r| - s$ spikes from σ_{l_i}, and $3|S_r| \leq 4|S_r| - s < 8|S_r|$, then one of these rules can be applied. If $\sigma_{l_i^1}$ applies one of these rules at $t' > t$, no spike remains. Otherwise, the $4|S_r| - s$ spikes can combine with the spikes from σ_r at a later step.

In the case where register r stores $n = 0$ (respectively, $n \geq 1$), then instruction l_k (respectively, l_j) is applied next. This case corresponds to σ_r applying the

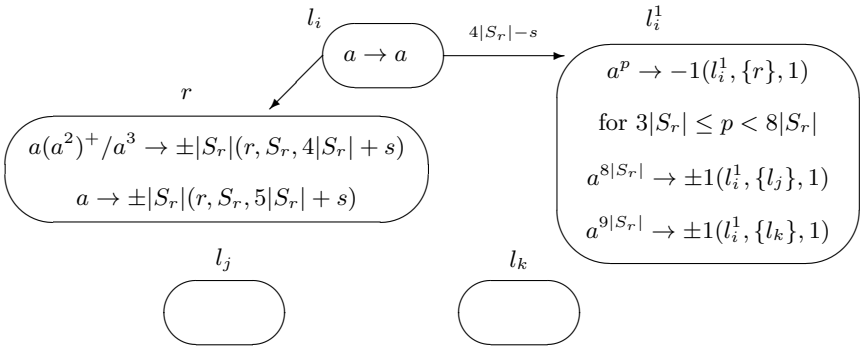

Fig. 4. Module SUB simulating $l_i : (\texttt{SUB}(r) : l_j, l_k)$ in the proof of Theorem 2.

rule with $E = a$ (respectively, $E = a(a^2)^+$), which at some later step allows σ_{l_k} (respectively, σ_{l_j}) to be activated.

For the moment let us simply define $S_r = \{l_i^1\}$. For case $n = 0$ (respectively, $n \geq 1$), σ_r stores 0 spikes (respectively, at least 2 spikes), so that at some $t'' > t$ the synapse $(r, l_i^1, 5|S_r| + s)$ (respectively, $(r, l_i^1, 4|S_r| + s)$) is created and then deleted. $\sigma_{l_i^1}$ then receives $5|S_r| + s$ spikes (respectively, $4|S_r| + s$ spikes) from σ_r. Note that we can have $t'' \geq t'$ or $t'' \leq t'$, due to *asyn* mode, where t' is again the step that $\sigma_{l_i^1}$ applies a rule. If $\sigma_{l_i^1}$ previously removed all of its spikes using its rules with $E = a^p$, then it again removes all spikes from σ_r because $3|S_r| \leq x < 8|S_r|$, where $x \in \{4|S_r| + s, 5|S_r| + s\}$. At this point, no further rules can be applied, and the computation aborts, i.e. no output is produced. If however $\sigma_{l_i^1}$ did not remove its spikes previously, then it collects a total of either $8|S_r|$ or $9|S_r|$ spikes. Either σ_{l_j} or σ_{l_k} is then activated by $\sigma_{l_i^1}$ at a step after t''.

To remove the possibility of "wrong" simulations when at least two SUB instructions operate on register r, we give the general definition of S_r: $S_r = \{l_v^1 | l_v$ is a SUB instruction on register $r\}$. In the SUB module, a rule application in σ_r creates (and then deletes) an $|S_r|$ number of synapses: one synapse from σ_r to all neurons with label $l_v^1 \in S_r$. Again, each neuron with label l_v^1 can receive either $4|S_r| + s$, or $5|S_r| + s$ spikes from σ_r, and $4|S_r| - s$ spikes from σ_{l_v}.

Let l_i be the SUB instruction that is currently being simulated in Π. In order for the correct computation to continue, only $\sigma_{l_i^1}$ must not apply a rule with $E = a^p$, i.e. it must not remove any spikes from σ_r or σ_{l_i}. The remaining $|S_r| - 1$ neurons of the form l_v^1 must apply their rules with $E = a^p$ and remove the spikes from σ_r. Due to *asyn* mode, the $|S_r| - 1$ neurons can choose not to remove the spikes from σ_r: these neurons can then receive further spikes from σ_r in future steps, in particular they receive either $4|S_r| + s'$ or $5|S_r| + s'$ spikes, for $1 \leq s' \leq S_r$; these neurons then accumulate a number of spikes greater than $8|S_r|$ (hence, no rule with $E = a^p$ can be applied), but not equal to $8|S_r|$ or $9|S_r|$ (hence, no plasticity rule can be applied). Similarly, if these spikes are not removed, and spikes from $\sigma_{l_{v'}}$ are received, $v \neq v'$ and $l_{v'} \in S_r$, no rule can again be applied: if $l_{v'}$ is the s'th SUB instruction operating on register r, then $s \neq s'$ and $\sigma_{l_{v'}}$ accumulates a number

of spikes greater than $8|S_r|$ (the synapse weight of $(l_{v'}, l^1_{v'})$ is $4|S_r| - s'$), but not equal to $8|S_r|$ or $9|S_r|$. No computation can continue if the $|S_r|-1$ neurons do not remove their spikes from σ_r, so computation aborts and no output is produced. This means that only the computations in Π that are allowed to continue are the computations that correctly simulate a `SUB` instruction in M.

The SUB module correctly simulates a `SUB` instruction: instruction l_j is simulated only if r stores a positive value (after decrementing by 1 the value of r), otherwise instruction l_k is simulated (the value of r is not decremented).

Module FIN: The module FIN for halting the computation of Π is shown in Fig. 5. The operation of the module is clear: once M reaches instruction l_h and halts, σ_{l_h} becomes activated. Neuron l_h sends a spike to σ_1, the neuron corresponding to register 1 of M. Once the number of spikes in σ_1 become odd (of the form $2n+1$, where n is the value stored in register 1), σ_1 keeps applying its only rule: at every step, 2 spikes are consumed, and 1 spike is sent to Env. In this way, the number n is computed since σ_1 will send precisely n spikes to Env.

The ADD module has nd_{syn}: initially it has $pres(l_i^2) = \emptyset$, and its $k = 1 < |N|$. We also observe the parameter values: m is at least 9 by setting $|S_r| = 1$, then adding the two additional rules in $\sigma_{l_i^1}$; k is clearly at least 1; lastly, the synapse weight l is at least 3 by again setting $|S_r| = 1$. This completes the proof. □

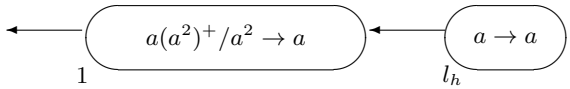

Fig. 5. Module FIN in the proof of Theorem 2.

5 Conclusions and final remarks

In [5] it is known that asynchronous SNP systems with extended rules are universal, while the conjecture is that asynchronous SNP systems with standard rules are not [3]. In Theorem 1, we showed that asynchronous bounded SNPSP systems are not universal where, similar to standard rules, each neuron can only produce at most one spike each step. In Theorem 2, asynchronous WSNPSP systems are shown to be universal. In WSNPSP systems, the synapse weights perform a function similar to extended rules in the sense that a neuron can produce more than one spike each step. Our results thus provide support to the conjecture about the nonuniversality of asynchronous SNP systems with standard rules. It is also interesting to realize the computing power of asynchronous unbounded (in spikes) SNPSP systems.

It can be argued that when $\alpha \in \{\pm, \mp\}$, the synapse creation (resp., deletion) immediately followed by a synapse deletion (resp., creation) is another form of synchronization. Can asynchronous WSNPSP systems maintain their computing power, if we further restrict them by removing such semantic? Another interesting

question is as follows: in the ADD module in Theorem 2, we have nd_{syn}. Can we still maintain universality if we remove this level, so that nd_{neur} in *asyn* mode is the only source of nondeterminism? In [5] for example, the modules used *asyn* mode and nd_{rule}, while in [15], only *asyn* mode was used (but with the use of a new ingredient called local synchronization).

In Theorem 2, the construction is based on the value $|S_r|$. Can we have a uniform construction while maintaining universality? i.e. can we construct a Π such that $N(\Pi) = NRE$, but is independent on the number of `SUB` instructions of M? Then perhaps parameters m and l in Theorem 2 can be reduced.

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.

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. 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)
4. 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)
5. Cavaliere, M., Ibarra, O., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems. Theor. Com. Sci. vol. 410, pp. 2352 - 2364 (2009)
6. Ibarra, O.H., Woodworth, S.: Spiking neural P systems: some characterizations. FCT 2007, LNCS vol. 4639, pp. 23 - 37 (2007)
7. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae, vol. 71(2,3), pp. 279-308 (2006)
8. Minsky, M.: Computation: Finite and infinite machines. Englewood Cliffs, NJ: Prentice Hall (1967)
9. Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Science China Information Sciences. vol. 54(8) pp. 1596 - 1607 (2011)

10. Pan, L., Wang, J., Hoogeboom, J.H.: Spiking Neural P Systems with Astrocytes. Neural Computation vol. 24, pp. 805 - 825 (2012)
11. Păun, A., Păun, G.: Small universal spiking neural P systems. Biosystems, vol. 90, pp. 48 - 60 (2007)
12. Păun, G.: Computing with membranes. J. of Computer and System Science, vol. 61(1), pp. 108 - 143 (1999)
13. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002)
14. Păun, G., Rozenberg, G., Salomaa, A., Eds.: The Oxford Handbook of Membrane Computing. Oxford Univ. Press. (2009)
15. Song, T., Pan, L., Păun, G.: Asynchronous spiking neural P systems with local synchronization. Information Sciences, vol. 219(10), pp. 197 - 207 (2013)
16. Wang, J., Hoogeboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Weights. Neural Computation, vol. 22(10), pp. 2615 - 2646 (2010)