Theoretical Computer Science 410 (2009) 2982–2991

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Sequential SNP systems based on min/max spike number[☆]

Oscar H. Ibarra[a,*], Andrei Păun[b,c,d], Alfonso Rodríguez-Patón[c]

[a] *Department of Computer Science, University of California, Santa Barbara, CA 93106, USA*
[b] *Bioinformatics Department, National Institute of Research and Development for Biological Sciences, Splaiul Independenţei, Nr. 296, Sector 6, Bucharest, Romania*
[c] *Universidad Politécnica de Madrid - UPM, Facultad de Informática, Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, Spain*
[d] *Department of Computer Science/IfM, Louisiana Tech University, P.O. Box 10137, Ruston, LA 71272, USA*

ARTICLE INFO

Keywords:
Membrane computing
Sequentiality
Universality
Spike number
Spiking neural P systems

ABSTRACT

We consider the properties of spiking neural P (SNP) systems that work in a sequential manner. These SNP systems are a class of computing devices recently introduced as a bridge between spiking neural nets and membrane computing. The general sequentiality of these systems was considered previously; now we focus on the sequentiality induced by the spike number: at each step, the neuron with the maximum (or minimum) number of spikes among the neurons that are active (can spike) will fire. This strategy corresponds to a global view of the whole network that makes the system sequential. We study the properties of this type of a restriction (i.e. considering the case of sequentiality induced by the function maximum defined on numbers of spikes as well as the case of the sequentiality induced by the function minimum similarly defined on numbers of spikes). Several universality results are obtained for the cases of maximum and minimum induced sequentiality.

1. Introduction

Spiking neural P systems (in short, SNP systems) were recently introduced in [5], and then investigated in [2,11], thus incorporating in membrane computing [10] ideas from spiking neurons, see, e.g., [1,6,7]. In this paper we consider a new restriction on the rule application (or neuron firing) in SNP systems. Several authors have recently noticed that the maximal parallelism way of rule application (which is widely used in membrane systems) is rather non-realistic in some cases. This fact motivated the consideration of various "strategies" and changes in the rule application in membrane systems (or neuron firing in SNP systems); for details we refer the interested reader to [4,9].

Here we consider the spiking restriction on neurons in the following way: if at any step there is more than one active neuron (one that can spike according to their pre-defined rules) then only the neuron(s) containing the maximum (or, in other cases, the minimum) number of spikes (among the currently active neurons) will be able to fire. This is in contrast with the maximal parallel application of the rules, in which case all the active neurons will fire at that specific step. To exemplify the firing mechanism of the new strategy, let us consider four neurons: n_1, n_2, n_3, n_4 that are the only active neurons at this step (according to their internal rules and the contents of spikes for each of them). In such a case we would find the maximum number of spikes stored in n_1 through n_4, say we have the values 5, 3, 7, 1. Then obviously the neuron n_3 holds the maximum number of spikes, and only n_3 will fire at the next step. After the spiking of n_3, we update the number of

[☆] Work supported in part by NSF Grants CCF-0430945, CCF-0523572 and CCF-0524136, support from INBRE Program of the NCRR (a division of NIH), support from CNCSIS grant RP-13, support from CNMP grant 11-56 /2007, support from Spanish Ministry of Science and Education (MEC) under project TIN2006-15595, and support from the Comunidad de Madrid (grant No. CCG07-UPM/TIC-0386 to the LIA research group).

* Corresponding author. Tel.: +1 805 893 4171; fax: +1 805 893 8553.
E-mail addresses: ibarra@cs.ucsb.edu (O.H. Ibarra), apaun@latech.edu (A. Păun), arpaton@fi.upm.es (A. Rodríguez-Patón).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.03.004

the spikes in the whole system according to its synapses, and at the next step the neurons n_1, n_2, n_4 together with n_3 will be checked if they can fire (as they may have been rendered inactive by an increment in their number of spikes stored). If there is a tie for the maximum number of spikes stored in the active neurons, then there are two distinct strategies that will be considered in the following: max-pseudo-sequentiality (when all the neurons containing the maximum will fire) and max-sequentiality (when only one of the neurons containing the maximum will fire).

The main motivation behind this spiking strategy is the observation that in a population of cells of the same type (neurons in this case) which have similar types of rules (the spiking rules in our case) one can notice that the cells containing larger numbers of a specific molecule species are more active (spike faster/more frequently) than the cells containing lower numbers of the same molecule. Another observation is the fact that the neurons that receive a large number of spikes are more probable to spike than the neurons that do not receive many spikes. The same modeling path was taken also when the *integrate-and-fire* models were defined for neurons, which leads to the neurons that receive more spikes to fire faster than the neurons that receive lower numbers of spikes.

The restriction proposed above makes the spiking of the neurons in the system almost sequential (more than one neuron can spike simultaneously only in the special case described above when there is a tie for the number of spikes contained). Because of this, we will call this application strategy *pseudo-sequential* with respect to maximum. One can also consider the sequential strategy which resolves the ties by choosing for the next spiking neuron nondeterministically one of the neurons containing the maximum number of spikes at that moment (out of the active neurons). This second strategy will be called in the following *max-sequentiality*. We will also study in the current paper the dual notion induced by the minimum function defined on the number of spikes in neurons. We will obtain universality results also for the *min-sequentiality* and *min-pseudo-sequentiality* cases.[1]

We will consider also the difference between these devices from the point of view of generators versus acceptors; specifically, we notice a major difference between systems with deterministic neurons working as generators as opposed to acceptors. We see that the acceptors are universal whereas the generators are only able to generate one single value (thus are non-universal).

2. Spiking neural P systems

The original definition of spiking neural P systems was given in [5]; the interested reader can find in the reference above the motivation, basic results etc. Let us recall the basic definition in the following.

A *spiking neural membrane system* (abbreviated as an SNP system), of degree $m \geq 1$, is a construct of the form $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
(2) $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
  (a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
  (b) R_i is a finite set of *rules* of the following two forms:
    (1) $E|a^c \to a; d$, where E is a regular expression over a, $c \geq 1$, and $d \geq 0$;
    (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that for each rule $E|a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$[2];
(3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
(4) $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (i.e., σ_{i_0} is the output neuron).

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and their usage is described in the following. If the neuron σ_i contains k spikes, and $a^k \in L(E)$, $k \geq c$, then the rule $E|a^c \to a; d$ can be applied. The application of this rule means consuming (removing) c spikes (thus only $k - c$ remain in σ_i), the neuron fires and produces a spike after d time units (as usual in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, etc. If the rule is used in the step t of the computation and $d \geq 1$, then we have the following setting: in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is *closed* (this corresponds to the refractory period from neurobiology), so that it cannot receive new spikes (if a neuron has a synapse to a closed, refractory, neuron and tries to send a spike along it, then that particular spike is lost). In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$).

The rules of type (2) are the *forgetting* rules; they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i.

If a rule $E|a^c \to a; d$ of type (1) has $E = a^c$, then we will write it in the following simplified form: $a^c \to a; d$. Furthermore, if $d = 0$ the rule will omit the delay in the notation: thus a rule $(aa)^*|a \to a; 0$ will be written from now on as $(aa)^*|a \to a$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *can* be used at the next step. Since two firing rules, $E_1|a^{c_1} \to a; d_1$ and $E_2|a^{c_2} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in

[1] The paper is an extended and improved version of the work reported in [3], where, among others, the min-sequentiality case was not considered.
[2] By $L(E)$ we denote the language associated with the regular expression E.

a neuron, and in that case only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

We note that the rules are used in the sequential manner in each neuron, but neurons were previously considered to function in parallel with each other. It is important to notice that the applicability of a rule is established based on the *total* number of spikes contained in the neuron. Thus, e.g., if a neuron σ_i contains 5 spikes, and R_i contains the rules $(aa)^*|a \to a$, $a^3 \to a$, $a^2 \to \lambda$, then none of these rules can be used: a^5 is not in $L((aa)^*)$ and not equal to a^3 or a^2. However, if the rule $a^5|a^2 \to a$ is in R_i, then it can be used: two spikes are consumed (thus three remain in σ_i), and one spike is produced and sent immediately ($d = 0$) to all neurons linked by a synapse to σ_i, and the process continues.

Starting from a fixed initial distribution of spikes in the neurons (initial configuration) and using the rules in a synchronized manner (a global clock is assumed), the system evolves. A computation is a sequence of transitions starting from the initial configuration. A transition is maximally parallel in the sense that all neurons that can apply at least one rule must use such a forgetting or spiking rule. However, in any neuron, at most one rule is allowed to fire. Further details about the motivation of the definition as well as the properties of the original model of SNP systems can be found in [5].

One can associate a set of numbers with Π in several ways. We follow here the idea of [5] and we consider the intervals between the very first two consecutive spikes of the output neuron as the numbers computed by such a device. Furthermore, we will consider only halting computations.

Let us consider in the following the max-sequentiality for SNP systems:

Definition 1 (*Max-sequentiality*). SNP systems defined as above are working in the *max-sequentiality* manner if (by definition) the system is choosing as the spiking neuron at each step only one of the neurons that can fire (thus the system works in a sequential way), and furthermore, the spiking neuron chosen at each time-step has the maximum number of spikes stored among all the other active neurons in that step.

Definition 2 (*Max-pseudo-sequentiality*). Systems can work in *max-pseudo-sequentiality* manner if (by definition) at each time-step fire all the neurons that store the maximum number of spikes among all the active neurons at that step.

Of course *max-sequentiality* is forcing the system to work in a sequential manner as at each step at most one neuron can fire, whereas the *max-pseudo-sequentiality* allows two or more neurons to fire at the same time if all those neurons hold exactly the same number of spikes and that number is the highest value of spikes that is stored among all the active neurons at that moment.

The definitions for min-sequentiality and min-pseudo-sequentiality are obvious dual definitions for the above two definitions in which the maximum is replaced by minimum.

An SNP system can be used as a computing device in various ways. Here, as in previous papers, we will use them as generators of numbers.

(Classification of neurons). We can characterize the neurons from such a system to be of three classes as defined in the following:

(1) A neuron is *bounded* if every rule in the neuron is of the form $a^i|a^j \to a; d$, where $j \leq i$, or of the form $a^k \to \lambda$, provided that for a rule $a^k \to \lambda$ there is no rule of the form $a^k|a^l \to a; d$ in the neuron. Note that there can be several such rules in the neuron. These rules are called *bounded rules*. (For notational convenience, we will write $a^i|a^i \to a; d$ simply as $a^i \to a; d$.)
(2) A neuron is *unbounded* if every rule in the neuron is of the form $E|a^k \to a; d$ where the language associated with E is infinite (i.e. we have at least one $*$ or $+$ in the regular expression E). (Again, there can be several such rules in the neuron.) These rules are called *unbounded rules*. As an example, the neuron having the following three rules is unbounded: $a^{2k+3}|a^5 \to a; 1$ and $a^{2k+3}|a^6 \to a; 2$ and $a^{2k}|a^2 \to a; 1$.
(3) A neuron is *general* if it can have *general rules*, i.e., bounded as well as unbounded rules. As an example, the neuron having the following three rules is general: $a^{2k+3}|a^5 \to a; 1$ and $a^{15}|a^6 \to a; 2$ and $a^{2k}|a^2 \to a; 1$.

(Classification of SNP systems). The classification of neurons into the three classes of bounded, unbounded and general can be extended naturally to systems of neurons in the following way. An SNP system is bounded if all the neurons in the system are bounded. If, in addition, there are unbounded neurons then the SNP system is said to be unbounded. A general SNP system has general neurons.

In this paper, we will study SNP systems operating in sequential and pseudo-sequential mode as described above. Informally, this means that at every step of the computation only active neurons that hold the maximum (or minimum) number of spikes can execute a firing rule. For each neuron that is spiking at one step only one spiking rule (nondeterministically chosen) is to be fired.

As defined before in [4], we can consider the effect of the notion of strong sequentiality (or pseudo-sequentiality) and the weak one on the power of such devices.

(1) *Case* 1: At every step, there is at least one neuron with a fireable rule (the strong case). We show that:
   (a) We obtain universality for unbounded SNP systems with delays.
   (b) We also get universality even for systems without delays, but in this case the type of rules needs to be extended (a neuron can send more than one spike at a time).

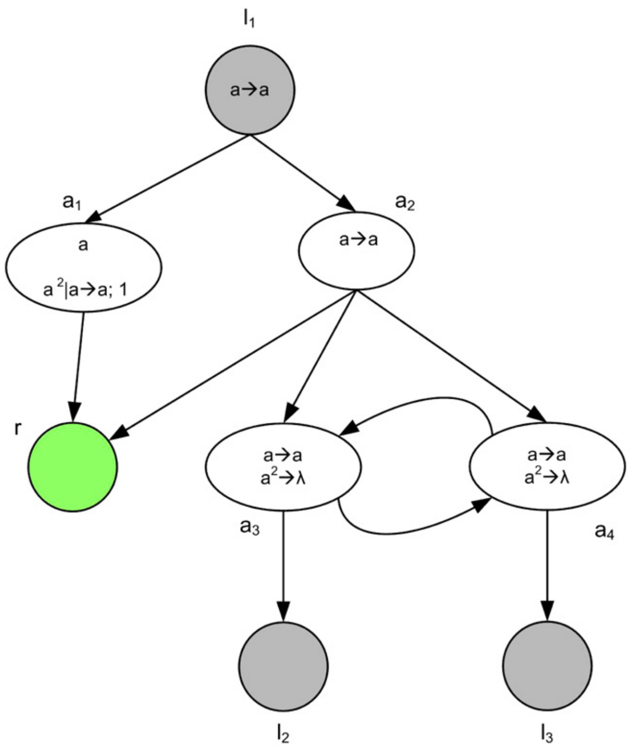

Fig. 1. The addition module for $l_1 : (ADD\, r, l_2, l_3)$.

(2) *Case* 2: Not every step has at least one neuron with a fireable rule (the weak case). (Thus, the system might be dormant until a rule becomes fireable. However, the clock will keep on ticking.) We will consider this second case in the future studies of such systems. One should note that even for the restrictive previous case we obtain universality, thus we need to investigate systems with even lower power than in that case.

For the basic definitions and prerequisites we refer the interested reader to [12,10,13]. We will use in the following universality proofs the fact that register machines are universal, but due to space limitations we will not provide the prerequisite description of the register machines, the reader is referred to [13] and [8] as this is a common proof technique.

We will study in the following the generative power of this restriction in the application strategy of the SNP systems. We will compare the previous results for the unrestricted sequential SNP systems as defined in [4] with results obtained for the devices that use the restricted versions of sequentiality proposed above (the max-sequentiality and min-sequentiality).

3. Universality of systems using the max-sequentiality strategy

We will start the description of results of these systems by giving the first theorem about systems based on strongly max-sequentiality with delays. We will show the universality of such systems as opposed to the result in [4] where the strongly sequential ones were shown to be not universal.

Theorem 3. *Unbounded SNP systems with delays working in a strongly sequential mode induced by the maximum number of spikes (max-sequentiality) are universal.*

Proof. We will show that any register machine can be simulated by a system working in a max-sequentiality manner, with rules using delays. Since it is known that the register machines are universal see e.g. [8], this would also prove that SNP systems as considered are also universal.

As we will see in the following, we will need to use rules with delays only for the neurons simulating the *ADD* rules in the register machine.

Let us give a brief description of the construction. In the system we will have neurons associated with each label in the program code of the register machine, also the registers will be modeled by neurons holding $2n$ spikes for the value n being stored in the register. Thus the *ADD* module will need to increase by 2 the number of spikes stored in the neuron associated with the register r (effectively incrementing the register r in the simulation) and then choose nondeterministically a new instruction to execute out of the two possibilities given by the *ADD* instruction.

In the following, in Fig. 1 we give the graphical description of the neurons that can be used to simulate a rule of the form $l_1 : (ADD\, r, l_2, l_3)$:

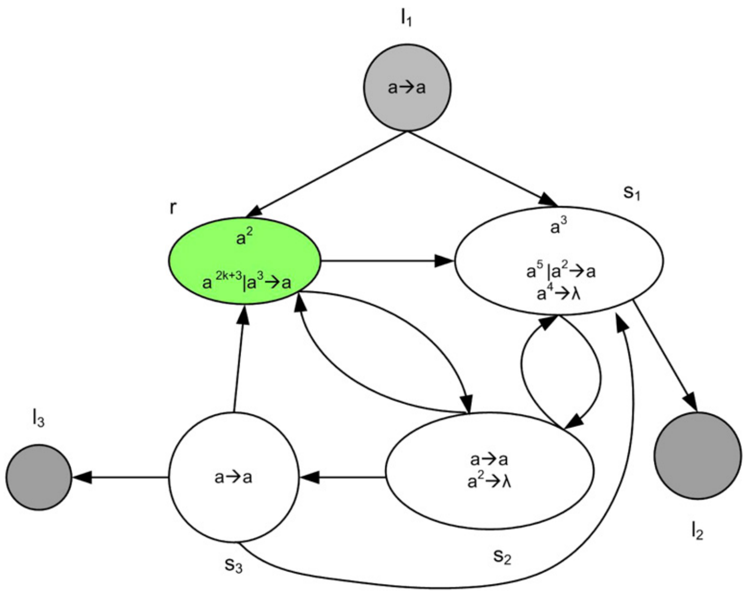

Fig. 2. The subtract module for l_1 : (*SUB* r, l_2, l_3).

The template module described in Fig. 1 works as follows: the neuron l_1 spikes, signalling that the instruction l_1 is being executed, then the neurons a_1 and a_2 are activated, since a_1 has at this moment two spikes and a_2 only one, and due to the max-sequentiality mode of operation for the system, a_1 fires first, but has a delay of size one associated with its rule, at the next step a_2 fires (since at that moment it is the only active neuron), making the spikes from a_1 and a_2 to arrive at the same time in the neuron r. We will see later that the neurons of type r can only fire when they hold an odd value, thus receiving two spikes keeps the neuron r inactive. At the same time a_2 sends spikes towards neurons a_3 and a_4. The job of the neurons a_3 and a_4 is to choose nondeterministically which register rule to activate next: l_2 or l_3. This is achieved by the fact that when receiving the spikes from a_2, both a_3 and a_4 are activated, both have exactly one spike at this moment, so we need to choose nondeterministically between them the one to fire next. Depending on the choice, the corresponding instruction l_2 (for a_3) or l_3 (for a_4) is activated. This is done in two steps: a_3 (or, in the other case, a_4) fires, then it sends to the other neuron a_4 (or, in the other case, a_3) another spike, making the forgetting rule applicable, and another spike to the neuron simulating the label of the next instruction. Since the a_4 (or, in the other case, a_3) neuron holds two spikes versus one spike for the label neuron, we first apply the forgetting rule, and then we receive a spike in neuron l_3 (or, in the other case, in l_2). This means that the spike from l_1 activates the module that increases the number of spikes in neuron r by 2 and then sends one spike in neuron l_2 or l_3 nondeterministically choosing which one. All the other neurons remain unchanged with respect to the initial configuration of the module. It is clear now that the simulation of the *ADD* rule in the register machine is correctly performed by the module described in Fig. 1. Since the module arrives in a similar configuration as its start configuration, it is clear that one can re-use the same module as many times as it is needed, as well as compose through the "label" neurons several such modules to simulate several *ADD* instructions from the register machine.

We will now give in Fig. 2 the module simulating the *SUB* instruction from the register machine. We will show that the simulation of the *SUB* instructions is also keeping the configuration similar with the start configuration of such a module making these modules reusable (composable).

In Fig. 2, when the neuron l_1 fires, it sends two spikes, one to the neuron r (modeling the register that is decremented or checked for zero) and one to the neuron s_1. We note that we will start with two spikes in the neurons modeling the registers (we will take care of the correct counting in the finalizing module). We also start with three spikes in the neuron s_1; thus at the next step the neuron s_1 contains exactly 4 spikes, whereas the neuron r contains exactly $2n + 3$ spikes, where n is the contents of the register r in the counter automaton.

We have now two possible cases:
Case I: If the register r is empty, it means that the neuron r holds exactly 3 spikes. Since these are the only two active neurons at this moment (r and s_1), then s_1 will execute first since it has four spikes (one more than r). This means that all four spikes in neuron s_1 are deleted through the forgetting rule applicable there; then at the next step r spikes sending one spike back to s_1 and activating s_2. At the next step s_2 fires sending one more spike to s_1 and sending another one back to r which was empty. At the next step s_3 fires also replenishing the two initial spikes in r and the third spike in s_1. This means that we reached a configuration similar to the original configuration when l_1 spiked, and now l_3 is activated (since the register was empty).

Let us consider the case when the register r would be non-empty which forms Case II. Since the register r holds the value n which is non-empty, then the neuron r would hold $2n + 3$ spikes, with $n \geq 1$, thus r will hold at least 5 spikes, more

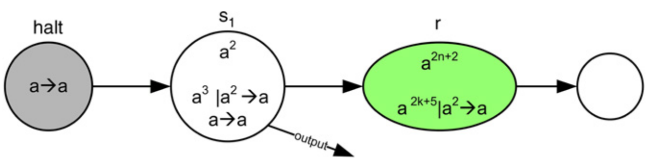

Fig. 3. The halting module.

than the four held by s_1. This means that r spikes next, sending one spike to s_1 and another spike to s_2. At the next step s_1 will have 5 spikes as opposed to s_2 that holds only one, thus s_1 spikes removing two spikes. Then at the next step we will have s_1 holding 3 spikes and being inactive, s_2 holding 2 and l_2 holding one. Thus next s_2 will forget its two spikes, making the configuration as before the module was activated and then the simulation can continue with the instruction labeled l_2 (simulated by the neuron l_2 which is the only active neuron at this time).

It is clear that the rules from the register machine are correctly simulated by the modules presented above. What remains to consider is the finishing stage in which the output register is read and processed in our setting. We need the output neuron to fire twice, and the number of clock cycles between the two fires to be exactly n where $2n + 2$ is the value stored in the neuron r corresponding to the output register r in the register machine.

Without loss of generality we can assume that the output register is never decremented (the register machine can be easily changed by adding another register and a couple of rules that would copy the contents of the output to the new register that would never be decremented).

When we activate the halting label in the register machine we send a spike to the output neuron (the neuron s_1 in Fig. 3). Thus at the next step s_1 spikes (being the only active neuron), then both r and s_1 become active. Thus the one holding the maximum number of spikes will fire. One can notice that in r we are deleting exactly 2 spikes each time, thus s_1 will let r spike as long as the register r (in the register machine) is non-empty, and at each time step two more spikes are removed (thus the register r is decremented by one each clock cycle). Thus the second time that s_1 spikes would have been exactly n clock cycles after the first spike, making the whole system to correctly simulate the work of the starting register machine. This completes the proof. □

If we consider the case of extended systems (where the neurons can send more than one spike through the synapse in one clock cycle), then we can easily remove the delay that appears in the template for the *ADD* instructions that was described in Fig. 1.

Theorem 4. *Extended unbounded systems operating under the max-sequentiality strategy that use only neurons without delays (thus strongly sequential) are universal.*

Proof. We change the *ADD* module depicted in Fig. 1 in the following way: change the rule in a_1 to: $a^2|a \to a^2$ and remove the synapse between a_2 and r. Everything else remains the same, thus one would use the template from Fig. 2 for the *SUB* instructions and use the module described in Fig. 3 for the terminating work. □

In the following section we will consider an even more realistic way of spiking for the neurons in the system: if there are ties for the maximum number of spikes stored in active neurons, then all the active neurons holding the maximum number of spikes will fire. This will be the case of max-pseudo-sequentiality that is discussed next.

4. Results concerning the universality of systems using the max-pseudo-sequentiality strategy

We start by noting that there is no nondeterminism at the level of the system: from each configuration to the next, we know for sure which neuron(s) will fire next (this was not the case with the max-sequentiality discussed in the previous section, for example one can refer to the work of neurons a_3 and a_4 in Fig. 1). Since the *SUB* module from Fig. 2 and *HALT* module from Fig. 3 given for the proof of Theorem 3 have always a single neuron holding the maximum, they actually satisfy also the max-pseudo-sequentiality case. It only remains to describe the *ADD* module for this setting of pseudo-sequentiality. We will see that we can give a system that does not use delays since the previous *ADD* module was the only module using this feature; thus we obtain the universality of the systems without requiring delays.

Theorem 5. *Unbounded SNP systems without delays working in the max-pseudo-sequentiality mode are universal.*

Proof. As we already mentioned above, we only need to describe the *ADD* module without delays that would be used as a template for simulating all the *ADD* instructions from the register machine that we simulate. The module is described in Fig. 4:

We will explain the work of the module depicted in Fig. 4 in the following. We start with the neuron l_1 firing, then the neurons a_1 and a_2 are activated as they both receive a spike from l_1. Because there is a tie for the maximum spikes contained in the active neurons, both a_1 and a_2 fire, sending exactly two spikes to r and another two spikes in a_3. We have simulated the incrementing of register r, so what remains now is the nondeterministic jump to the instruction labeled either l_2 or l_3. This will be achieved by sending a single spike at the end of the processing in the module either to the neuron l_2 or neuron l_3, which is done with the help of neurons a_3 through a_8. Because the contents of r will have an even number of spikes, it

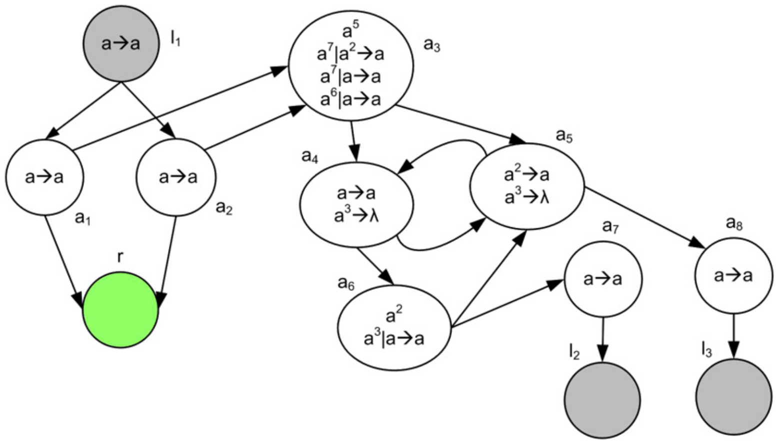

Fig. 4. The addition module for l_1 : $(ADD\ r, l_2, l_3)$.

is inactive at the next step, thus only a_3 can fire. At this moment we will have a nondeterministic choice between firing using the rule $a^7|a^2 \rightarrow a$ or $a^7|a \rightarrow a$. The choice is whether the neuron will fire once (with $a^7|a^2 \rightarrow a$) or twice (through $a^7|a \rightarrow a$, and then $a^6|a \rightarrow a$).

Case I: Let us assume that a_3 uses the rule $a^7|a^2 \rightarrow a$, then at the next step only a_4 is active, and after a_4 fires, both a_5 and a_6 are active, but a_5 holds 2 spikes and a_6 holds 3, thus a_6 fires. This means that at the next step a_5 (with 3 spikes) and a_7 (with one spike) are active, so a_5 fires, erasing all its spikes and then a_7 activates the new instruction to be executed, l_2. Let us consider the other case.

Case II: a_3 fires the rule $a^7|a \rightarrow a$, then at the next step both a_3 (with 6 spikes) and a_4 (with 1 spike) are active, then a_3 spikes once more, making a_5 the only active neuron, at the next step a_4 with 3 spikes is activated, together with a_8. Then a_5 and a_4 apply their forgetting rules, and then a_8 activates the label l_3.

Thus we correctly simulated the increment instruction on register r and the module depicted in Fig. 4 reaches a configuration similar to its start configuration; thus it can be reused as well as linked with other modules. The *SUB* module from 2 and *HALT* module from 3 remain the same. This completes the proof. □

An interesting observation is the fact that if one considers deterministic neurons (neurons in which the regular languages associated with each rule are disjoint), then such a system cannot produce nondeterminism. Thus we have immediately the following result:

Theorem 6. *A system of deterministic neurons working in a max-pseudo-sequential manner (as a generator) is non-universal.*

Proof. We notice that we cannot have nondeterminism at the level of neurons because they are deterministic. Since the determinism at the level of the system is also removed by the max-pseudo-sequentiality, then each such system will generate at most one value for each starting configuration. Obviously such devices that are able to generate only singleton sets are non-universal. □

This previous result contrasts with the fact that if such devices are used in an acceptor mode, then they are universal:

Theorem 7. *A system of deterministic neurons working in a max-pseudo-sequential manner (as an acceptor) is universal.*

Proof. Given an acceptor register machine, one can construct an SNP system with input neuron r simulating the work of the register machine.

We start with the neuron r containing exactly $2n + 2$ spikes (for the value n to be accepted or rejected by the register machine). Then using the *ADD* module from Fig. 4 and the *SUB* module from Fig. 2 given above one can simulate correctly the instructions in the register machine. Thus if we reach the neuron with the label *HALT*, we should accept the value n, whereas if we do not reach the neuron *HALT*, then we should reject the value n. □

We now pass to the dual case of considering the minimum rather than maximum when choosing which neuron will spike next. What this means is that at any step we will be choosing the active neuron that has the minimum number of spikes.

5. Universality of systems using the min-sequentiality strategy

As opposed to Theorem 3, we obtain the universality without delays for the case when considering the minimum number of spikes in neurons rather than the maximum. Since we have the min-sequentiality without delays then the system will also operate in a so-called strongly sequential mode. This result is obtained in the case when the output is realized through accumulation of spikes in the output neuron rather than the time elapsed between the two spikes of the output neuron.

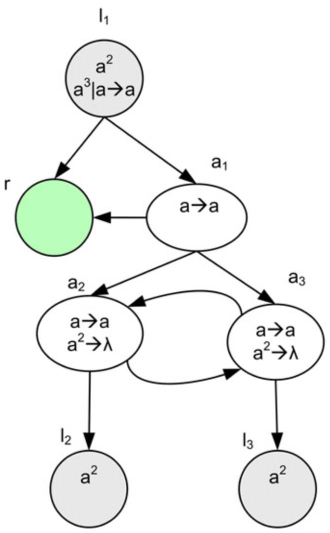

Fig. 5. The addition module for $l_1 : (ADD\, r, l_2, l_3)$, min-sequentiality.

Theorem 8. *Unbounded SNP systems operating under the min-sequentiality strategy without delays are universal. In this case we consider the output to be given as the number of spikes in a given neuron.*

Proof. We will show that any register machine can be simulated by a system working in a min-sequentiality manner, with rules without delays.

Let us give a brief description of the construction: In the system we will have neurons associated with each label in the program code of the register machine, also the registers will be modeled by neurons holding $2n$ spikes for the value n being stored in their respective register. Thus the *ADD* module will increase by 2 the number of spikes stored in the neuron associated with the register r (effectively incrementing the register r in the simulation) and then choose nondeterministically a new instruction to execute out of the two possibilities given by the *ADD* instruction.

In the following Fig. 5 we give the graphical description of the neurons used to simulate a rule of the form $l_1 : (ADD\ (r), l_2, l_3)$:

Let us give the description of the work of the module depicted in Fig. 5. When the neuron l_1 fires it increases the number of spikes stored in neuron r and activates the neuron a_1. At this point the register r has at least two spikes stored (since it starts with 2, and the *SUB* instruction (as we will see in Fig. 6) will never decrement it beneath 2), thus the next neuron firing is a_1 (having only one spike). When a_1 fires a_2 and a_3 receive each one spike. Neuron r receives one more spike at this step effectively incrementing the register r. By increasing with 2 spikes the contents of neuron r we have effectively simulated an increment in the value of the register r (2 spikes mean the addition of value 1 to the contents of the register). At the next step we have a tie between the neurons a_2 and a_3 with respect to the minimum number of spikes stored in each of them. Depending on which neuron wins the tie (a_2 or a_3) then the next instruction in the register machine is chosen (l_2 or l_3). The process proceeds in the following way: a_2 fires, then l_2 will store 3 spikes and a_3 only 2, so a_3 will first use the forgetting rule $a^2 \to \lambda$ and then l_2 will fire, signalling the beginning of the simulation of instruction with label l_2. In a similar way we have also the case for the activation of l_3. It is clear now that the "helper" neurons will have the same number of spikes as they had in the initial configuration of the system, thus this module can be re-used any number of times (depending on how many times the instruction is used in the register machine).

Let us now pass to the case of modeling the *SUB* instructions in the register machine. We will need to consider both the case when the register is non-empty and the case of the empty register.

For an instruction $l_1 : (SUB\, r, l_2, l_3)$ we will have the following module in Fig. 6 that simulates the work of the register machine instruction:

We will now describe in detail the work of the *SUB* module from Fig. 6: for the case when $r \neq 0$ we will have the neuron r containing at least five spikes, since at the same step β_1 contains only four spikes, we have that the next neuron that fires is β_1. It fires and then β_2 and β_3 receive one spike each. β_2 has only one spike, so it will fire at the next step, then β_3 will now store 4 spikes and cannot fire as it does not contain any rule applicable for 4 spikes. Thus, at this step we have that neuron r is firing making its number of spikes even (and in this way brings β_1 to its original number of three spikes); also it pushes β_3 to five spikes and at the same time the neuron α_1 will reach six spikes. Since β_3 stores less spikes than α_1 ($5 < 6$) we then have β_3 firing at the next step. During the next clock-cycle α_1 is receiving its seventh spike and β_4 is activated (it will hold eight spikes). Obviously α_1 has less spikes than β_4, thus α_1 forgets all its spikes and during the next step β_4 fires activating β_5. During the next few steps the neurons β_5, β_6, β_7, β_8 are used to replenish the contents of the neuron α_1 to five spikes that it had originally, and after this sequence of fires, l_2 is activated by sending one spike to that neuron. Obviously we will have the same configuration for the neurons in the system as the initial configuration, and the contents of the neuron r was

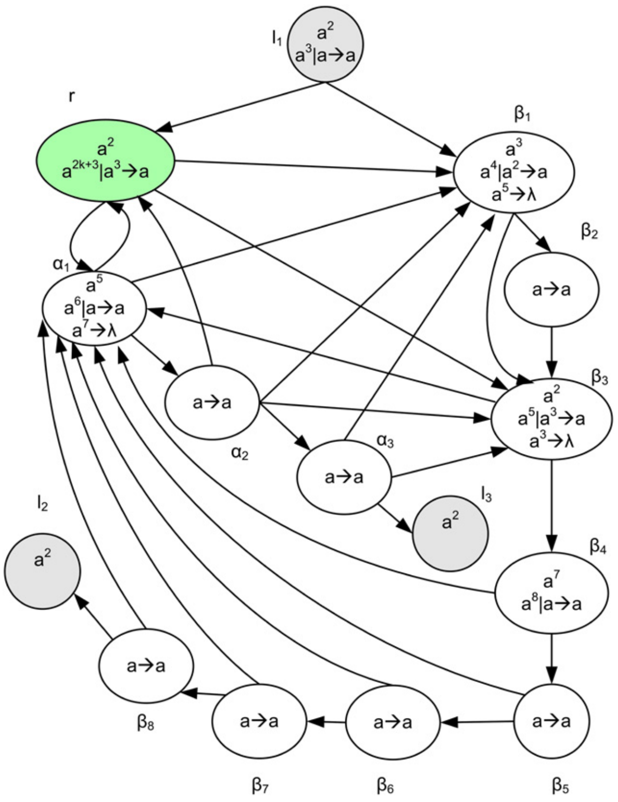

Fig. 6. The subtract module for l_1 : (*SUB* r, l_2, l_3), min-sequentiality.

decremented by 2 and the next instruction label was chosen as l_2. Considering all of the above, it should be clear that the work of the instruction *SUB* in the case when the register is non-empty is correctly simulated by the given module.

We now pass to the second case, when $r = 0$. In this setting the neuron r contains exactly 3 spikes whereas the neuron β_1 contains 4 spikes. Because of this, at the next step we will have the neuron r firing. α_1 receives its sixth spike, β_1 receives its fifth spike and β_3 receives its third. Thus at the next step β_3 forgets all its spikes, then β_1 forgets all its spikes, and finally, α_1 fires. The neurons α_2 and α_3 are then activated sequentially and used to replenish β_1 and β_3 to their original levels (3 and respectively 2 spikes). These neurons are used to replenish as well the contents of register r to the value 2. Finally the instruction l_3 is activated by the spike being sent to the register l_3.

It is clear that the rules from the register machine are correctly simulated by the modules presented above.

Since we consider the case when the output is by definition the contents of a specific register and because we can assume without loss of generality that the output register is never decreasing we can change briefly the *ADD* modules that are involving the output register: we will use as a template the *ADD* module presented as above in Fig. 5, but since there is no decrement instruction associated with this register, there is no need for storing double the number of spikes with respect to the value of the register. Thus only for the output register, if we store the value 8 in the register r_{out}, we will have 8 spikes in the neuron labeled r_{out}. We can remove the synapse between a_1 and r_{out} in the *ADD* module depicted in Fig. 5 and the behavior needed is achieved.

It should be clear now that at the end of the simulation the neuron r_{out} will hold exactly the same number of spikes as the value of the register r_{out} in the register machine, thus we have correctly simulated the register machines and proved the universality of SNP systems that work in a min-sequential way without delays and which have the output considered as the number of the spikes stored in a specific neuron in the final configuration of the system. □

We have showed that systems based on min-sequentiality are universal; the question whether min-pseudo-sequentiality is universal or not is still open. One can note immediately (from the proof of Theorem 8) that the module from Fig. 5 would not work correctly in the pseudo-sequentiality case, thus the proof is not trivial.

6. Final remarks

We plan to continue the investigation of this special type of strategy that induces the sequentiality (or pseudo-sequentiality) of the system. We are working on improving the results obtained; especially in the min-sequentiality case we believe that universality can be obtained also for the more restrictive case of the output being the time that elapsed between two consecutive spikes of the output neuron rather than the number of spikes stored at the moment of the halting computation.

Another direction would be to consider SNP systems that are stochastic with respect to the next spiking neuron in the following sense: each active neuron has a probability of spiking that increases with the number of spikes stored in the neuron. Such a strategy would yield probabilistic results for the computation, which we believe would be very interesting to characterize and study. We will pursue more avenues of research in this direction as we believe that this model could be very relevant to an experimental implementation of such a stochastic system.

Acknowledgements

We gratefully acknowledge the anonymous referees that helped improve significantly the presentation of the paper.

References

[1] W. Gerstner, W. Kistler, Spiking Neuron Models. Single Neurons, Populations, Plasticity, Cambridge Univ. Press, 2002.
[2] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth, Normal forms for spiking neural P systems, Theoretical Computer Science 372 (2–3) (2007) 196–217.
[3] O.H. Ibarra, A. Păun, A. Rodríguez-Patón, Sequentiality induced by spike number in SNP systems, in: Ashish Goel, Friedrich C. Simmel, Petr Sosík (Eds.), DNA14, June 2–6, 2008, Prague, Czech Republic, Preliminary Proceedings, 2008, pp. 36–46.
[4] O.H. Ibarra, S. Woodworth, F. Yu, A. Păun, On spiking neural P systems and partially blind counter machines, in: Unconventional Computing 2006, in: Lecture Notes in Computer Science, vol. 4135, 2006, pp. 113–129.
[5] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundamenta Informaticae 71 (23) (2006) 279–308.
[6] W. Maass, Computing with spikes, Foundations of Information Processing of TELEMATIK 8 (1) (2002) 32–36 (special issue).
[7] W. Maass, C. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, 1999.
[8] M. Minsky, Computation — Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, 1967.
[9] A. Păun, B. Popa, P systems with proteins on membranes, Fundamenta Informaticae 72 (4) (2006) 467–483.
[10] Gh. Păun, Membrane Computing — An Introduction, Springer-Verlag, Berlin, 2002.
[11] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, Spike trains in spiking neural P systems, International Journal of Foundations of Computer Science 17 (4) (2006) 975–1002.
[12] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, 3 volumes, Springer-Verlag, Berlin, 1997.
[13] The P systems web page: http://psystems.disco.unimib.it.