Fundamenta Informaticae 134 (2014) 167–182
DOI 10.3233/FI-2014-1097
IOS Press

Three Universal Homogeneous Spiking Neural P Systems Using Max Spike

Andrei Păun[*][†]
Department of Computer Science, Faculty of Mathematics and Computer Science
University of Bucharest, Str. Academiei nr.14, sector 1, C.P. 010014, Bucuresti, Romania
apaun@fmi.unibuc.ro

Petr Sosík
Research Institute of the IT4Innovations Centre of Excellence, Faculty of Philosophy and Science
Silesian University in Opava 74601 Opava, Czech Republic
petr.sosik@fpf.slu.cz

Abstract. We improve and extend a recent result showing that spiking neural P systems with the same rules in all neurons of the system (homogenous) and working in the max sequential manner are universal. The previous work in this area reported by the group led by Dr. Linqiang Pan did not put any bound on the number of neurons used. We believe this is an important question for any future practical implementation of such systems that deserves investigation, and we provide some results in this direction. Extending the aforementioned construction with the work of Korec on small register machines one could estimate the size of the previous construction at 105 neurons. We are able to improve this result and to show that an SNP system with 83 neurons having homogenous rules and working in the max sequential manner is universal. Several related results with respect to max-pseudo sequentiality mode are also obtained: 83 neurons are necessary for this case, too. When considering the case of systems without weighted synapses, we show that one needs at most 244 homogenous neurons for reaching universality in the max-pseudo sequentiality case.

Keywords: Membrane computing, spiking neural P system, max spike, universal machine

[*]Address for correspondence: Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei nr.14, sector 1, C.P. 010014, Bucuresti, Romania
[†]Also works: Bioinformatics Department, National Institute of Research and Development for Biological Sciences

1. Introduction

This paper deals with Spiking Neural P systems (SNP systems) as defined in [5] and working in a sequential manner induced by the max-spike idea described in [3]. The present paper can be viewed as a continuation of the result reported in [13] in combination with the result recently described in [6]. In [13] it is shown that SNP systems with a fixed number of neurons are universal. A rather surprising result was reported in [6], showing the universality of SNP systems containing homogeneous neurons (all the neurons in the system have the same spiking rules, the difference residing in the number and types of connections between the neurons, the axons [22]). With this construction in mind we proceeded to show that systems with homogenous neurons and a fixed number of such neurons (albeit large at the moment) are universal. This is a significant result in the area as we are moving closer to a wet-lab implementation of these results: recent reports in the area of nanotechnology applied to neurons (the real cells in the human or mouse body) show that we can re-wire a neuronal network and influence the axon and synapse growth and even design networks with different topologies of real neurons, glia and synapses between neurons by using nanoparticles such as gallium phosphide, magnetic fields and other techniques [1], [18], [16]. This, combined with recent simulation techniques [20], [21] and HPC resources [11] might eventually lead to implementations of networks of real neurons.

With this information in mind we point out that neurons in organisms frequently form homogeneous blocks with the same set of rules. This observation makes the results from [6] even more relevant. We now know that systems/networks of neurons working together, each of them having the same rules, are computationally universal with at most 105 neurons.

We now pass to a brief description of the restrictions considered here: if at any step there are more than one neuron that can spike (according to their pre-defined rules) then only the neuron(s) containing the maximum number of spikes (among the currently "active" neurons) will fire. This is contrasting with the maximal parallel application of the rules case, in which case all the "active" neurons will fire at that step. If there is a tie for the maximum number of spikes stored in the active neurons, then there are two possibilities: only one such neuron fires or all the neurons containing the maximum will fire.

The restriction proposed above makes the spiking of the neurons in the system almost sequential (more than one neuron can spike only in the special case when there is a tie in the number of spikes contained, the two or more active neurons that contain the maximum number of spikes over all the active neurons at that step will spike). Because of this, we will call this application strategy *max-pseudo sequential.* 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 from now on *max sequential.*

The dual case of considering the minimum number of spikes contained in neurons will lead to two more strategies: *pseudo-sequential* with respect to minimum, and *sequential* with respect to minimum. These types of firing strategies are not discussed further in this paper but we consider interesting such a setting as well.

2. Basic Description and Definitions

The spiking neural P systems (for short, SNP systems) were introduced in [5], and then intensely investigated, incorporating in membrane computing [14] some ideas from spiking neurons, see, e.g., [2], [8], [9].

A *weighted spiking neural P system* (abbreviated as SNP), 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 $a^s \notin L(E)$;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \times N$ with $(i, i, n) \notin syn$ for $1 \leq i \leq m$ and $n \in N$ and the triplet (i, j, k) specifying that there is a synapse leaving neuron i and arriving in neuron j of weight k (*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 they are applied as follows. 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 is fired, and it 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 at the next step, etc. If the rule is used at the step t of the computation and $d \geq 1$, then we have the following setting: at 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 neuron and tries to send a spike along it, then that particular spike is lost). At 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 $t + d + 1$). At the next time-step after the application of a spiking rule the spikes are sent from the firing neuron(s) to the receiving neurons according to the synapses described in syn. If a neuron i fires and there are synapses with the neurons j_1 and j_2 described as (i, j_1, k_1) and (i, j_2, k_2) in syn then at the next step k_1 spikes are sent to neuron j_1 and k_2 spikes towards neuron j_2. If the neurons j_1 or j_2 are not in the refractory period then their current number of spikes is incremented with the value k_1 or k_2 respectively.

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, thus 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: $a^c \to a; d$.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i *must* be used. 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 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.

Thus, 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 $(aa)^*/a \rightarrow a; 0$, $a^3 \rightarrow a; 0$, $a^2 \rightarrow \lambda$, 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 $a^5/a^2 \rightarrow a; 0$ 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.

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 numbers computed by a computation. Furthermore, we will consider only halting computations.

It is known that SNP systems can be used both as generators or acceptors and that sometimes there is a difference in their power between the two models of computation [3]. Specifically, we observed 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).

In the following we will consider a special methodology for rule application: the neuron that contains the most spikes at one moment is the next neuron firing, which will make the system sequential.

Definition 2.1. 1. 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, 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.
2. 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 since at most one neuron can fire at each step, whereas the *max-pseudo sequentiality* allows two or more neurons to fire at the same time if all those neurons hold the exactly same number of spikes and that number is the highest value of spikes that is stored among all the active neurons at that moment.

Register Machines: In the proofs in the next sections we will use register machines as universal computing devices characterizing NRE, hence the Turing computability.

Informally speaking, a register machine consists of a specified number of registers (counters) which can hold any natural number, and which are handled according to a program consisting of labeled instructions; the registers can be increased or decreased by 1 – the decreasing being possible only if a register holds a number greater than or equal to 1 (we say that it is non-empty) –, and checked whether they are non-empty. It is known (see [10]) that non-deterministic register machines generate exactly the family NRE, of Turing computable sets of numbers. Moreover, without loss of generality, we may assume that in the halting configuration all registers except the second one, where the result of the computation is stored, are empty.

We will use the construction from [7] where the universality of a small register machine with only 23 instructions (including the halt instruction) working on 8 registers was proven: In the registers 1 and 2 there are codified the machine and the input for the simulated machine.

3. Universality of Small Homogenous SNP Systems: the Generator Case

When we talk about the universality (actually strong universality) of a generator, in this setting we will use the same framework from Korec: an SN P system Π_μ is universal if, given a fixed admissible enumeration of the unary partial recursive functions, $(\phi_0, \phi_1, \dots)$, there is a recursive function g such that for each natural number x, if we input the number $g(x)$ in Π_μ, by "reading" the sequence $10^{g(x)-1}1$ from the environment, the set of numbers generated by the system is equal to $\{n \in N \mid \phi_x(n) \text{ is defined}\}$. Otherwise stated, after introducing the "code" $g(x)$ of the partial recursive function ϕ_x in a specified neuron, the system generates (hence halts sometimes after sending two spikes out) all numbers n for which $\phi_x(n)$ is defined.

Considering the construction of a universal SNP system in [6], we will give an estimation of the number of neurons in the system and then we improve it. First, let us count the number of neurons needed in each module: the construction in [6] used one neuron for each register, one neuron for each instruction label, 2 extra neurons for the ADD instructions and 3 extra neurons for the SUB instructions. The input would need 3 neurons (the reading of the $g(x)$) and then the generating phase will require 2 neurons. Finally, the halting module requires two neurons. So in total combining the constructions from [6] with the strong universality result of Korec [7] we obtain the following number of neurons in the system: the construction of Korec's small register machine [7] which has 23 instructions including the halt instruction working on 8 registers. We need to add two more instructions (one ADD and one SUB) and one extra register because the construct from [6] did not allow decrements on the output register. Thus the system using the previous construction will have the following number of neurons: 9 for each register, 25 instruction labels, 10*2 neurons for the ADD instructions, 14*3 for the SUB instructions as well as 2 for the halt module and 4 for input plus 3 more for the generating phase of the number, in total we have: 105.

We will improve this number in several ways: using the specific construction of Korec one can observe that the non-determinism will be needed only in the initialization phase of the system, thus the majority of ADD instructions will be deterministic, thus eliminating 20 neurons from the system. We will also modify some modules of the construction to make them more efficient and such that one could use the framework of Korec. We obtain the following result:

Theorem 3.1. There are universal generative SNP systems with at least 83 homogenous neurons and operating in max sequentiality mode.

Proof:
We will show that Korec's register machine [7] can be simulated by a system working in a max sequentiality manner. The Korec register machine (in the main theorem it is the a2 machine: RiP, RiZM type machine) has 8 registers and 23 instructions including the halt instruction, see bellow.

$$\begin{array}{ll}
l_0 : (SUB(1),\, l_1,\, l_2), & l_1 : (ADD(7),\, l_0), \\
l_2 : (ADD(6),\, l_3), & l_3 : (SUB(5),\, l_2,\, l_4), \\
l_4 : (SUB(6),\, l_5,\, l_3), & l_5 : (ADD(5),\, l_6), \\
l_6 : (SUB(7),\, l_7,\, l_8), & l_7 : (ADD(1),\, l_4), \\
l_8 : (SUB(6),\, l_9,\, l_0), & l_9 : (ADD(6),\, l_{10}), \\
l_{10} : (SUB(4),\, l_0,\, l_{11}), & l_{11} : (SUB(5),\, l_{12},\, l_{13}), \\
l_{12} : (SUB(5),\, l_{14},\, l_{15}), & l_{13} : (SUB(2),\, l_{18},\, l_{19}), \\
l_{14} : (SUB(5),\, l_{16},\, l_{17}), & l_{15} : (SUB(3),\, l_{18},\, l_{20}), \\
l_{16} : (ADD(4),\, l_{11}), & l_{17} : (ADD(2),\, l_{21}), \\
l_{18} : (SUB(4),\, l_0,\, l_h), & l_{19} : (SUB(0),\, l_0,\, l_{18}), \\
l_{20} : (ADD(0),\, l_0), & l_{21} : (ADD(3),\, l_{18}), \\
l_h : HALT &
\end{array}$$

We will add one more register that is non decreasing (and this will be the output register) and to integrate it we will replace the old halt instruction $l_h : HALT$ with the instructions $l_h : (SUB(0), l_{22}, l'_h)$, $l_{22} : (ADD(8), l_h)$, and $l'_h : HALT$. Thus at the end of the program of the initial machine we will copy the contents of the old output register (register 0) into the register 8 which will serve as the new output register. We do this so that no substraction rule will be applicable to the output register and we will be able to use the same proof technique as used before in [3], [6].

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 $5n$ spikes for the value n being stored in the register. Thus the ADD module will increase by 5 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 end we will give a module INIT that will take care of initializations. For now let us assume that the system starts with $5n$ spikes in neuron 1 for the recursive function $g(x)$ and $5m$ spikes in neuron 2 codifying y, where $g(x)$ and y are defining the partial recursive function that will be computed from the list of a fixed admissible enumeration of the partial recursive functions: $\phi_0,\ \phi_1,\ \ldots$. I.e. $\phi_x(y)$ is computed by the register machine M that we simulate. Actually we compute a value that is five times the $\phi_x(y)$ in register 8 and then in the HALT module we show how to output the result.

The most important observation that needs to be made at this moment is that in the aforementioned figures all the neurons have the *same* spiking rules. These rules are given in the following:

$a \to a$
$a^2 \to \lambda$
$a^3/a^2 \to a$
$a^4/a^2 \to a$
$a^6 \to \lambda$
$a^7 \to a$
$a^9/a^4 \to a$
$a^{6(5)^+}/a^{11} \to a$
$a^{7(5)^+}/a^5 \to a$

In the Figure 1 we give the neurons necessary to simulate an $l_1 : (ADD(r), l_2)$ instruction (note that the instructions used in the Korec construction are all deterministic, thus the nondeterminism at the level of the ADD rules is not needed):

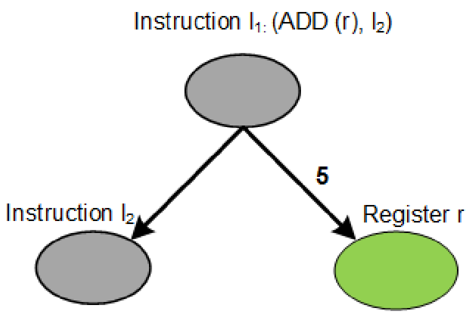

Figure 1. The addition module for $l_1 : (\mathrm{ADD}(r),\ l_2)$

The module works as follows: the neuron l_1 spikes, signalling that the instruction l_1 is being executed, then immediately we send 5 spikes to the neuron r that simulates the register r effectively incrementing that register and at the same time we send one spike to register l_2 activating that instruction. Obviously the instruction is simulated correctly.

We will now give the module simulating the SUB instruction from the register machine in the Figure 2. We show how we are simulating all the ADD and SUB instructions in general.

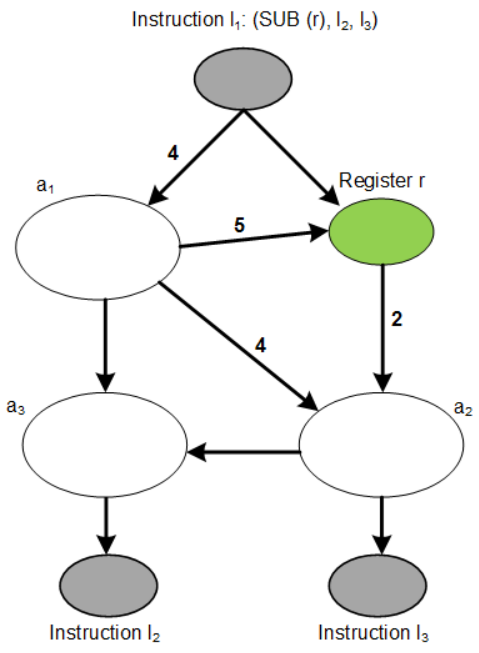

Figure 2. The subtract module for $l_1 : (\mathrm{SUB}(r),\ l_2,\ l_3)$

Initially the neuron l_1 is activated and fires according to the rule $a \to a$. This sends one spike to neuron r and four to the neuron a_1. At the next step the checking of the contents of the register r is performed by the max sequentiality: only the register with the maximum number of spikes will fire. We have two cases: the register r is empty (value 0) meaning that the value in the neuron r is 6 or the register is non-zero, making the number of spikes in the neuron r at least 11. Case 1: if r has 6 spikes it will forget all these spikes using the rule $a^6 \to \lambda$, and at the next step neuron a_1 is the only one firing using the rule $a^4/a^2 \to a$ replenishing the neuron r to the value 5 (which was stored initially in this neuron) and then activating the neurons a_2 (with 4 spikes) and a_3 that receives one spike. In the next step we have the following configuration: a_1 with 2 spikes, a_2 with 4 and a_3 with one. Obviously a_2 will fire because it has the maximum number of spikes and sends one extra spike to a_3 and one spike to l_3. Thus at the next step the neurons a_1 and a_2 and a_3 will all have 2 spikes and l_3 one spike. In the next three steps the neurons $a_i,\ 1 \leq i \leq 3$ will fire forgetting their spikes (and thus coming back to the original configuration of the SUB module) and finally at step 4 the neuron l_3 fires activating the rule associated with the label l_3 in the register machine.

Case 2: the register is non-zero, there are a^{5k+1} spikes in the neuron r with $k \geq 2$. In this case the neuron r will fire next, but in this case by using the rule $a^{6(5)^+}/a^{11} \to a$ which means that it sends two spikes to all the neurons of type a_2 from all the instructions of type SUB that work on the register r. We still have 4 spikes in the neuron a_1 from the current SUB module, this will fire next sending four more spikes to the a_2 current and one to a_3. At the next step we have several a_2 neurons in the whole system activated, all having 2 spikes, only one having 6 and one a_3 with one spike. In several steps the neurons labeled a_2 will forget their spikes by using the forgetting rules: $a^6 \to \lambda$ in the first step and $a^2 \to \lambda$ in the subsequent steps. Thus all the two spikes that were sent to the neurons a_2 that were not from the current module have been forgotten by using the rule $a^2 \to \lambda$. Finally, the neuron a_3 fires activating the instruction l_2 effectively simulating correctly also this case of the SUB instruction. Since the configuration is reversed to the initial state for this module, it can be re-used in the program, making the simulation of the SUB instruction correct.

It is clear that the rules from the register machine are correctly simulated by the modules presented above. What remains to discuss about is the initial module that adds $5 * g(x)$ in neuron 1 and $5 * y$ in neuron 2, starts the l_0 instruction and also the finishing stage in which the output register (register 8) is read and processed in our setting:

The INIT module described in Figure 3 functions in the following way: the input neuron (neuron i_1) receives exactly 2 spikes, the distance in clock cycles between the first two is the value $g(x)$ which needs to be put in the register 1 and then we generate an arbitrary value y that we then process; this value will be stored in register 2. Neuron i_1 is the only active neuron immediately after the first spike received from *outside*; it will spike using the rule $a^{6(5)^+}/a^{11} \to a$ reaching 10 spikes (and being inactive), but activating i_4, and simultaneously increasing the number of spikes in neuron i_2 to 8 and in i_3 to 5. In the next step the neuron i_4 fires, incrementing the value in the register 1 and activating the neuron i_3. This neuron will fire using the rule $a^9/a^4 \to a$ and sending another 5 spikes into neuron r incrementing that register and also 2 spikes to i_4. This process repeats with i_4 firing at one step and i_3 at the next until another spike is received by i_1 and it fires for the second time. This will stop the computation of both these neurons: depending whether the input value is odd or even i_4 will stop with 8 spikes or with 6 (and later forget these 6 stopping at 0) and i_3 will stop with 10 spikes or with 14.

Then the neuron i_2 with 11 spikes will start the processing on register 2: this is the generative part of the machine, it will randomly put a value into the register 2 using the neurons i_5 and i_6. They are used as

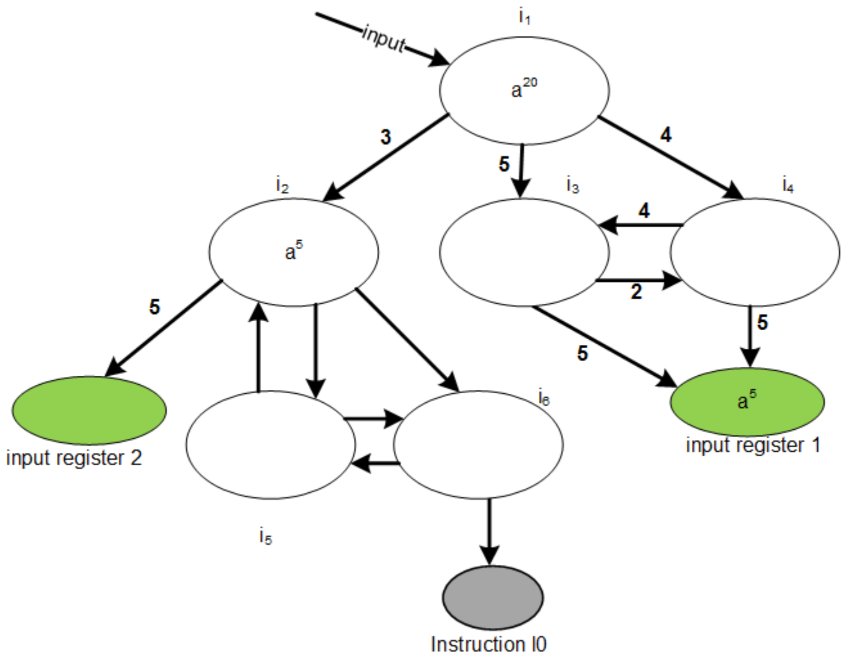

Figure 3. The INIT module.

in a non-deterministic ADD instruction, re-using the neuron i_2 for the nondeterminism: i_2 fires sending 5 spikes into the neuron 2 incrementing the register. At the next step we choose nondeterministically which neuron to fire: i_5 or i_6. If i_5 fires, it re-activates i_2 which will fire again and increment once more the register 2. At the next step we forget the two spikes from i_6 and we can repeat these steps until we choose nondeterministically to fire i_6 in the tie between i_5 and i_6. This activates the neuron l_0 which is the first label in the register machine, while i_5 with two spikes eventually forgets them.

Let us now pass to the HALT module described in Figure 4. From the construction we know that the output register (register 8) is never decrementing.

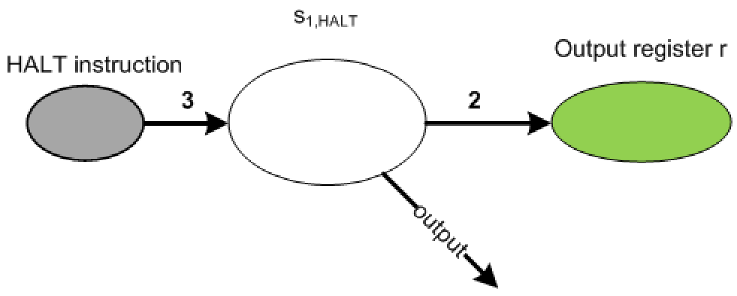

Figure 4. The halting module.

When we activate the halting label in the register machine we send a spike in the output neuron (the neuron $s_{1,HALT}$ in the picture above). Thus at the next step $s_{1,HALT}$ spikes (being the only active neuron), then both r and $s_{1,HALT}$ are active. Thus the one holding the maximum number of spikes will fire. One can notice that in r we are deleting exactly 5 spikes each time by using the rule $a^{7(5)^+}/a^5 \to a$, thus $s_{1,HALT}$ will let r spike as long as the register r (in the register machine) is non-empty, and at each time step five more spikes are removed (thus the register r is decremented by one each clock cycle). The last spiking rule will be $a^7 \to a$ making that neuron inactive, thus the second time that $s_{1,HALT}$ spikes would have been exactly n clock cycles after the first spike for $5n$ the value of register r, making the whole system to correctly simulate the work of the starting register machine.

It remains to count now the neurons: INIT module requires 6 neurons plus the registers (green) and the start instruction label (grey), HALT requires 1 neuron, we will have 9 neurons for the registers (green) and 25 neurons (grey) for the instruction labels (the 23 instructions from the modified Korec construction plus the one ADD and one SUB instructions that deal with our new register 8). So we will have exactly 10 ADD, 14 SUB and 1 halt instructions. Since in the ADD module we do not need any extra neurons (other than the two labels and the register) and in the SUB module we need 3 (a_1, a_2, a_3) the total becomes: 6+9+25+14*3+1=83. This completes the proof. □

4. Universality in the Max-Pseudo Sequentiality Mode

The results given in the previous section were based on the max sequential manner of the execution. In this section we extend the universality results also for the case of the max-pseudo sequential mode. In this mode, all the active neurons must fire that store the maximum number of spikes among all the active neurons at that step. Therefore, the non-deterministic selection of one of these neuron as in the max sequential mode is suppressed.

Theorem 4.1. There are universal generative SNP systems with at least 83 homogenous neurons and operating in max-pseudo sequentiality mode.

Proof:
Inspecting carefully the modules ADD and HALT given at Fig. 1 and 4, we observe that these modules are fully deterministic in the sense that there is never a tie for the maximum number of spikes between two or more active neurons. Therefore, their function is identical in the max-pseudo sequential mode. Inspecting the module SUB at Fig. 2 decrementing a register r, there are two such ties: (i) if the register r was empty, then in a certain step neurons a_1, a_2 and a_3 will all have 2 spikes; (ii) if r was nonempty, then several neurons a_2 in the whole system are activated, all having 2 spikes. In both cases (i) and (ii) all these neurons forget their spikes by using the rule $a^2 \to \lambda$. Therefore, during their sequential application no other spike is sent to the system. Hence, if these rules are applied in parallel, the system would reach in one step the same configuration as in several steps in the sequential case.

Finally, the situation is more complicated in the module INIT at Fig. 3 where the non-deterministic choice in tie between neurons i_5 and i_6 is used to generate a random number y. As this method is unusable in the max-pseudo sequential mode, the module INIT must be modified as indicated at Fig. 5, and two new rules are added to all neurons:

$a^{14}/a \to a$,
$a^{14}/a^9 \to a$.

These rules can act only in neurons i_5 and i_6 in the INIT module at Fig. 5, as no other neuron (in any module) can reach 14 spikes.

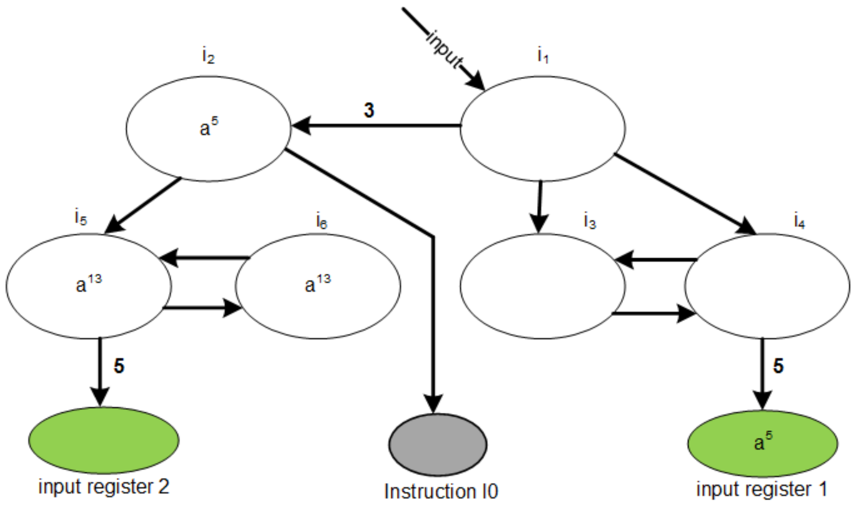

Figure 5. The INIT module working in the max-pseudo sequential mode.

When i_1 spikes first (notice that no initial spikes are necessary in it in the pseudo-sequential mode), the neuron i_2 reaches 8 spikes with no rule applicable, and both i_3 and i_4 are activated with one spike. Both of them fire in the next step, incrementing the value in the register 1 and re-activating both i_3 and i_4 with one spike. This process repeats until another spike is received by i_1 and it fires again. This will increase the number of spikes in both i_3 and i_4 to two which are later forgotten.

The generation of the random value y and its placement into the input register 2 works as follows:

- when i_1 spikes for the second time, i_2 reaches 11 spikes and fires in the next step, leaving i_5 with 14 spikes and l_0 with one spike;
- i_5 fires next, and assume that the rule $a^{14}/a \to a$ is chosen non-deterministically; then i_6 reaches 14 spikes and i_5 reaches 13, while the input register 2 is incremented;
- then the whole situation repeats with neurons i_5 and i_6 having exchanged their roles, until one of these neuron chooses the rule $a^{14}/a^9 \to a$;
- when the rule $a^{14}/a^9 \to a$ is chosen in either i_5 or i_6, then this neuron stays with 5 spikes while the other neuron reaches 14 and fires in the next step. Then it stops with either 13 or 5 spikes, depending on the chosen rule. The spike will replenish the first neuron to 6 spikes which are forgotten in the next step.

Finally, when i_5 and i_6 finish their activity, then the neuron l_0 with one spike can fire and start the simulation of the register machine.

Therefore, following the same argumentation as in the proof of Theorem 3.1, the total number of necessary neurons is again 83, which concludes the proof. □

5. Universality Without Weighted Synapses

The weighted synapses, allowing to "multiply" spikes sent between neurons, represent an extension of the original model of SN P systems. We show that in the max-pseudo sequential mode the weights can be omitted, i.e., all the synapses would have weight 1, but the number of necessary neurons will increase.

The usual way how to avoid weighted synapses in SN P systems is to replace a neuron with a maximum weight n of an outgoing synapse with an n-tuple of identical neurons with identical input and output working in parallel. Thus, instead of one spike weighted by n, the destination neuron would receive n spikes from these n neurons. If the neuron has another outgoing synapse with a weight $k < n$, the corresponding destination neuron will receive spikes only from k of these n parallel neurons. The approach in the case of module ADD is illustrated at Fig. 6.

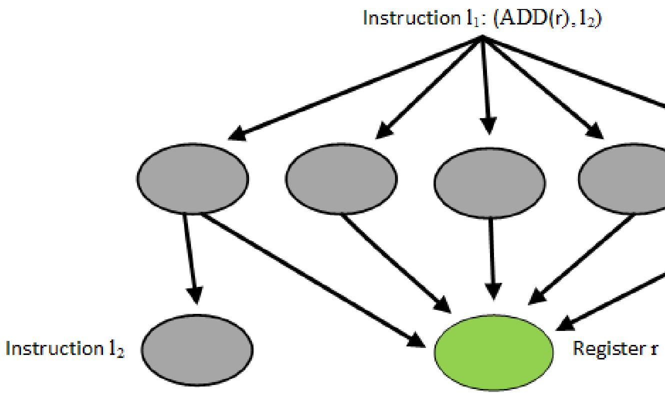

Figure 6. The module ADD without weighted synapses.

The max sequential mode would cause sequential firing of these n parallel neurons, thus "de-synchronizing" the system. Therefore, in this section we consider the max-pseudo sequential mode only.

Theorem 5.1. There are universal generative SNP systems with at least 244 homogenous neurons without weighted synapses and operating in max-pseudo sequentiality mode.

Proof:
We apply the principle of "neuron multiplication" described above to all the modules ADD, SUB, INIT and HALT. The module ADD implementing instruction $l_1 : (ADD(r), l_2)$ has been already described in Fig. 6. Note that each instruction which precedes the one labeled l_1 must connect its corresponding output neuron to all 5 neurons labeled l_1. The total number of neurons in the module will be 5.

In the module SUB at Fig. 2, also each neuron representing register i which is decremented must be duplicated, to be able to send 2 spikes to neuron a_2. We will denote these two neurons register i and register i'. The module will contain 11 neurons in total (without the neurons implementing registers).

Rewriting the module HALT at Fig. 4 without weighted synapses results in 5 neurons.

Finally in the module INIT at Fig 5, the neuron i_5 cannot be replaced with five parallel neurons, since i_5 non-deterministically chooses when it stops sending spikes to the input register 2. If some of the 5 parallel neurons stopped sooner than the others, the number of spikes in the input register 2 would not be a multiple of 5, corrupting the computation.

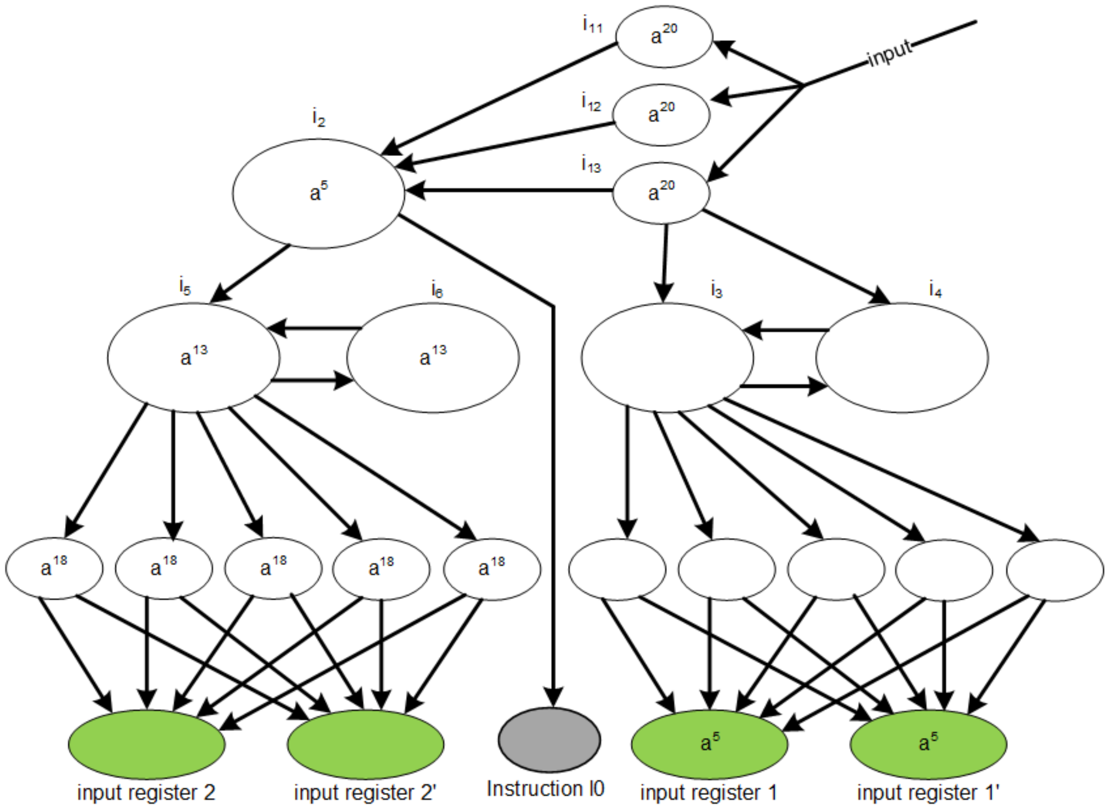

Figure 7. The INIT module working in the max-pseudo sequential mode.

Therefore, the solution depicted at Fig. 7 is adopted. Neuron i_3 is connected to a 5-tuple of intermediate neurons to send 5 spikes to the input register 1 for each step when i_3 spikes (actually, the last 5 spikes are postponed until the neuron corresponding to Instruction I0 spikes, too, due to the maximality principle). Neuron i_5 is connected to another 5-tuple of intermediate neurons. However, since the rule $a \to a$ is not applicable while neurons i_5 and i_6 exchange spikes, a new rule was added to all neurons:

$$a^{19}/a \to a.$$

Each time i_5 spikes, the new rule is activated in these five intermediate neurons, sending 5 spikes to the input register 2. Then neuron i_6 spikes and the process continues as described in the previous section.

Summarizing the number of neurons necessary to implement the Korec's register machine: INIT module requires 18 neurons, not counting the registers (green) and the I0 instruction label (grey). We will have $8 \times 2 + 1 = 17$ neurons for the registers (green), as there are 8 decrementable and 1 non-decrementable register). There are 10 ADD instructions (5 neurons each including the label), 14 SUB instructions (11 neurons each including the label), the HALT instruction with 5 neurons including the label. The total number of neurons becomes: 18+17+10*5+14*11+5=244, which concludes the proof.

$\square$

6. Final Remarks

We plan to continue the investigation of this special type of homogenous neurons working in max sequentiality mode; it would be interesting if in the case of max sequentiality we could obtain a similar universality result for the non-weighted synapses case; we believe this is doable with an increase in rules (and neurons). We think that other small improvements are possible which would lead to 2–3 fewer neurons for each case (max sequentiality or max-pseudo sequentiality). Such optimizations could be achieved by observing some particularities of the machine designed by Korec: there is a sequence of two consecutive ADD instructions:

$$l_17:\ (ADD(2),\ l_{21}),\ l_{21}:\ (ADD(3),\ l_{18}),$$

without any other instruction addressing the label l_{21}. This could be simulated by a module incrementing both registers at the same time and in this way we can save the neuron associated with the label l_{21}. A similar operation may be possible for the following two sequences of ADDSUB instructions, where again we could save the intermediate labels (l_6 and l_10), for each pair of the following instructions:

$$l_5:\ (ADD(5), l_6),\ l_6 : (SUB(7), l_7, l_8),$$

$$l_9:\ (ADD(6), l_{10}),\ l_{10} : (SUB(4), l_0, l_{11}).$$

Instead of modules ADD and SUB as in Figs. 1 and 2 (which are used in both sequential and pseudo-sequential construction), for each couple of instructions as above we could use a module doing all these operations at the same time. In each case, we will save the neurons of the intermediate labels; together with the neuron saved by the module of two consecutive ADD instructions, we get the improvement from 83 to 80 neurons in the max sequential and max-pseudo sequential case.

Another setting to be investigated is the acceptor mode which could be also feasible. We are looking also at the min-spike case which aside from a non-biological motivation seems to be a bit harder and the construction would need a significant increase in the number of neurons. Finally we need to mention the work of Pan and collaborators in [23], [24] or [17] which devised other proof techniques that could prove useful in improving the current results.

Finally, one could be interested in decreasing the number of rules associated with each neuron, thus a trade-off between the number of rules and the size of the system could be envisioned. All these remarks show that this area is a rich research field which has also practical relevance due to the recent advances in the bio-nano-technology related to neurons and manipulating networks of neurons in vitro.

Acknowledgements

We gratefully acknowledge support in part from CNCS grant TE-97, ANCS project PN-II-PT-PCCA-2011-3.2-1710 and FP7 project DANCERS, the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), the Ministerio de Ciencia e Innovación, Spain, under project TIN2012-36992, and the Silesian University in Opava under the Student Funding Scheme, project SGS/6/2014.

References

[1] G. Bugnicourt, J. Brocard, A. Nicolas, C. Villard, Catherine: Nanoscale surface topography reshapes neuronal growth in culture, *Langmuir*, 2014, in press.

[2] W. Gerstner, W Kistler: *Spiking neuron models. Single neurons, populations, plasticity*. Cambridge Univ. Press, 2002.

[3] O. H. Ibarra, A. Păun, A. Rodriguez-Paton, Sequential SNP systems based on min/max spike number, *Theoretical Computer Science*, 410(30-32), (2009), 2982–2991.

[4] O. H. Ibarra, A. Păun, A. Rodriguez-Paton, Sequentiality induced by spike number in SNP systems, *Proceedings of DNA Computing conference 2008*, also *Lecture Notes in Computer Science*, Volume 5347, (2009), 179–190.

[5] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2-3), (2006), 279–308.

[6] K. Jiang, T. Song, W. Chen, L. Pan: Homogeneous spiking neural P systems working in sequential mode induced by maximum spike number, *International Journal of Computer Mathematics*, 90 (4), (2013), 831–844.

[7] I. Korec, Small universal register machines, *Theoretical Computer Science*, 168, (1996), 267–301.

[8] W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1, (2002), 32–36.

[9] W. Maass, C. Bishop, eds.: *Pulsed neural networks*, MIT Press, Cambridge, 1999.

[10] M. Minsky: *Computation – Finite and infinite machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[11] Y. Liu, R. Nassar, C. Leangsuksun, et al.: An optimal checkpoint/restart model for a large scale High Performance Computing system. *IEEE Int. Symposium on Parallel and Distributed Processing 2008*, 1–9.

[12] A. Păun, Gh. Păun, Small universal spiking neural P systems, *BioSystems*, 90(1), (2007), 48–60.

[13] A. Păun, M. Sidoroff: Sequentiality induced by spike number in SNP systems: small universal machines, *Conference on Membrane Computing*, *Lecture Notes in Computer Science*, Volume 7184, (2012), 333–345.

[14] Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.

[15] 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.

[16] G. Piret, M. Perez, C. Prinz: Neurite outgrowth and synaptophysin expression of postnatal CNS neurons on GaP nanowire arrays in long-term retinal cell culture, *Biomaterials*, 34 (4), (2013), 875–887.

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

[18] C. Riggio, M. P. Calatayud, M. Giannaccini, B. Sanz, T. E. Torres, R. Fernndez-Pacheco, A. Ripoli et al.: The orientation of the neuronal growth process can be directed via magnetic nanoparticles under an applied magnetic field, *Nanomedicine: Nanotechnology, Biology and Medicine*, (2014) in press.

[19] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.

[20] B.M. Strimbu, J.L. Innes, V.F. Strimbu, A deterministic harvest scheduler using perfect bin-packing theorem. *European Journal of Forest Research*, 129-5, (2010), 961–974.

[21] B.M. Strimbu, J.L. Innes, An analytical platform for cumulative impact assessment based on multiple futures: the impact of petroleum drilling and forest harvesting on moose (Alces alces) and marten (Martes americana) habitats in northeastern British Columbia, *Journal of Environmental Management*, 92 (7), (2011), 1740–1752.

[22] X. Zeng, X. Zhang and L. Pan: Homogeneous spiking neural P systems, *Fundamenta Informaticae*, 97, (2009), 1–20.

[23] X. Zhang, X. Zeng, and L. Pan: Smaller universal spiking neural P systems, *Fundamenta Informaticae*, 87 (1), (2008), 117–136

[24] X. Zhang, Y. Jiang and L. Pan: Small universal spiking neural P systems with exhaustive use of rules, *Journal of Computational and Theoretical Nanoscience*, 7 (5), (2010), 890–899.

[25] The P Systems Web Page: `http://ppage.psystems.eu`.