Notes About Spiking Neural P Systems

Mihai Ionescu[1], Gheorghe Păun[2,3]

[1] University of Piteşti
Str. Târgu din Vale, nr. 1, 110040 Piteşti
Romania
mihaiarmand.ionescu@gmail.com
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
[3] Research Group on Natural Computing
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Summary. Spiking neural P systems (SN P systems, for short) are much investigated in the last years in membrane computing, but still many open problems and research topics are open in this area. Here, we first recall two such problems (both related to neural biology) from [15]. One of them asks to build an SN P system able to store a number, and to provide it to a reader without losing it, so that the number is available for a further reading. We build here such a memory module and we discuss its extension to model/implement more general operations, specific to (simple) data bases. Then, we formulate another research issue, concerning pattern recognition in terms of SN P systems. In the context, we define a recent version of SN P systems, enlarged with rules able to request spikes from the environment; based on this version, so-called SN dP systems were recently introduced, extending to neural P systems the idea of a distributed dP automaton. Some details about such devices are also given, as a further invitation to the reader to this area of research.

1 Introduction

The present notes are only an invitation to the reader to a recent and vividly investigated branch of membrane computing, inspired from the way the neurons cooperate in large nets, communicating (among others) by means of spikes, electrical impulses of identical shapes. In neural computing, this biological reality has inspired a series of research which are considered "neural computing of the third generation", see, e.g., [5], [12]. In terms of membrane computing, the idea was captured in the form of so-called *spiking neural P systems*, in short, SN P systems, introduced in [10] and then investigated in a large number of papers. We refer to the Handbook [20] and to the membrane computing website [23] for details.

For the reader's convenience, we shortly recall that an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. Such a rule is of the form $E/a^c \to a^p; d$, where E is a regular expression over the alphabet $\{a\}$ (a denotes the spike); such a rule can be used in a neuron if the number of spikes present in the neuron is described by the regular expression E (if there are k spikes in the neuron, then $a^k \in L(E)$, where $L(E)$ is the regular language identified by E), and using it means consuming c spikes (hence $k - c$ remain) and producing p spikes, which will be sent to all neurons to which a synapse exists which leaves the current neuron, after a time delay of d steps (if $d = 0$, then the spikes leave immediately). If a neuron can use a rule, then it has to use one, hence the system is synchronized, in each time unit, all neurons which can spike should do it. One starts from an initial configuration and one proceed by computation steps as suggested above. One neuron is designated as the *output* neuron of the system and its spikes can exit into the environment, thus producing a *spike train*. Two main kinds of outputs can be associated with a computation in an SN P system: a set of numbers, obtained by considering the number of steps elapsed between consecutive spikes which exit the output neuron, and the string corresponding to the sequence of spikes which exit the output neuron. This sequence is a binary one, with 0 associated with a step when no spike is emitted and 1 associated with a step when a spike is emitted.

Several variants were considered in the literature. We recall here the most recent one, SN P systems with request rules, proposed in [9]: also rules of the form $E/\lambda \leftarrow a^r$ are used, with the meaning that r spikes are brought in the neuron, provided that its content is described by the regular expression E.

Such rules are essentially used in defining SN dP systems, a class of distributed computing devices bridging the SN P systems area and the dP systems area – this latter one initiated in [16] and then investigated in [4], [17], [18]. We recall here the definition of SN dP systems from [9], as well an example from that paper.

Also, two research topics mentioned in the last years (especially in the framework of the Brainstorming Week on Membrane Computing, organized at the beginning of each February in Sevilla, Spain – see [23]) are briefly discussed, for the first one also providing a preliminary answer (which, in turn, raises further questions). Namely, the challenge was to define a SN P module which simulate the "memory function" of the brain: stores a number, which can then be read by another "part of the brain" without losing the respective number. Such a module is provided here, but it suggests a series of continuations in terms of (simple) data bases, where several "memory cells" can be considered, loaded and interrogated, removed and added. Continuing our construction remains a task for the reader, and similarly with the second problem: to construct an SN P system able to recognize patterns, in a precise way which will be defined below.

In short, this is only a quick introduction to the study of SN P systems, by giving a few basic definitions and a short list of recent notions and research topics. The interested reader should look for further details in the domain literature.

2 Formal Language and Automata Theory Prerequisites

We need below only a few basic elements of automata and language theory, and of computability theory. It would be useful for the reader to have some familiarity with such notions, e.g., from [21] and [22], but, for the sake of readability we introduce here the notations and notions used later in the paper.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. If $x = a_1 a_2 \dots a_n$, $a_i \in V$, $1 \leq i \leq n$, then $mi(x) = a_n \dots a_2 a_1$.

We denote by REG, RE the families of regular and recursively enumerable languages. The family of Turing computable sets of numbers is denoted by NRE (these sets are length sets of RE languages, hence the notation).

In the rules of spiking neural P systems we use the notion of a *regular expression*; given an alphabet V, (i) λ and each $a \in V$ is a regular expression over V, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. The non-necessary parentheses can be omitted, while $E_1^+ \cup \lambda$ can be written as E_1^*. With each expression E we associate a language $L(E)$ as follows: (i) $L(\lambda) = \{\lambda\}, L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1)(E_2)) = L(E_1)L(E_2)$, $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for any regular expressions E_1, E_2.

The operations used here are the standard union, concatenation, and Kleene +. We also need below the operation of the *right derivative* of a language $L \subseteq V^*$ with respect to a string $x \in V^*$, which is defined by

$$L/x = \{y \in V^* \mid yx \in L\}.$$

In the following sections, when comparing the power of two language generating/accepting devices the empty string λ is ignored.

3 Spiking Neural P Systems with Request Rules

We directly introduce the type of SN P systems we investigate in this paper; the reader can find details about the standard definition in [10], [19], [2], etc.

An *(extended) spiking neural P system* (abbreviated as *SN P system*) with *request rules*, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0),$$

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 forms
   (i) $E/a^c \to a^p$, where E is a regular expression over a and $c \geq p \geq 1$ (spiking rules);
   (ii) $E/\lambda \leftarrow a^r$, where E is a regular expression over a and $r \geq 1$ (request rules);
   (iii) $a^s \to \lambda$, with $s \geq 1$ (forgetting rules) such that there is no rule $E/a^c \to a^p$ of type (i) or $E/\lambda \leftarrow a^r$ of type (ii) with $a^s \in L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn$, $1 \leq i, j \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (σ_{i_0}) of the system.

A rule $E/a^c \to a^p$ is applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can *fire*, and its application means consuming (removing) c spikes (thus only $k - c$ remain in σ_i) and producing p spikes, which will exit immediately the neuron. A rule $E/\lambda \leftarrow a^r$ is used if the neuron contains k spikes and $a^k \in L(E)$; no spike is consumed, but r spikes are added to the spikes in σ_i. In turn, a rule $a^s \to \lambda$ is used if the neuron contains exactly s spikes, which are removed ("forgotten"). A global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.

If a rule $E/a^c \to a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \to a^p$.

The spikes emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$, i.e., if σ_i has used a rule $E/a^c \to a^p$, then each neuron σ_j receives p spikes. The spikes produced by a rule $E/\lambda \leftarrow a^r$ are added to the spikes in the neuron and to those received from other neurons, hence they are counted/used in the next step of the computation.

If several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

During the computation, a configuration of the system is described by the number of spikes present in each neuron; thus, the initial configuration is described by the numbers $n_1, n_2, \ldots, n_m$.

Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule can be used.

There are many possibilities to associate a result with a computation, in the form of a number (the distance between two input spikes or two output spikes) or of a string. Like in [3], we associate a symbol b_i with a step of a computation when i spikes exit the system, thus generating strings over an alphabet $\{b_0, b_1, \ldots, b_m\}$,

for some $m \geq 1$. When one neuron is distinguished as an input neuron, then the sequence of symbols b_i associated as above with the spikes taken from the environment by this input neuron also forms a string. In both cases, we can distinguish two possibilities: to interpret b_0 as a symbol or to simply ignore a step when no spike is read or sent out. The second case provides a considerable freedom, as the computation can proceed inside the system without influencing the result, and this adds power to our devices.

In what follows, we also consider an intermediate case: the system can work inside for at most a given number of steps, k, before reading or sending out a symbol. (Note the important detail that this is a *property* of the system, not a *condition* about the computations: all halting computations observe the restriction to work inside for at most k steps, this is not a way to select some computations as correct and to discard the others. This latter possibility is worth investigating, but we do not examine it here.) The obtained languages, in the accepting and the generating modes, are denoted $L_k^a(\Pi), L_k^g(\Pi)$, respectively, where $k \in \{0, 1, 2, \ldots\} \cup \{\infty\}$. Then, L_0^g corresponds to the restricted case of [3] and L_∞^g to the non-restricted case (denoted L_λ in [3]).

The respective families of languages associated with systems with at most m neurons are denoted by $L_k^\alpha SNP_m$, where $\alpha \in \{g, a\}$ and k is as above; if k is arbitrary, but not ∞, then we replace it with $*$; if m is arbitrary, then we replace it with $*$. (Note that we do not take here into account the descriptional complexity parameters usually considered in this framework: number of rules per neuron, numbers of spikes consumed or forgotten, etc.)

The computing power of the previous devices was preliminarily investigated in [9], with many questions still remaining open. We do not recall them here, but, instead, we mention the notion of a SN dP system introduced in [9].

4 SN dP Systems

We first recall from [9], without proofs, two basic results, because they provide a way to find counterexamples in this area. Actually, they are extensions to SN P systems with request rules of some results already proved in [3].

Lemma 1. *The number of configurations reachable after n steps by an extended SN P system with request rules of degree m is bounded by a polynomial $g(n)$ of degree m.*

Theorem 1. *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π with request rules such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L_*^g(\Pi)$.*

Corollary 1. *The following two languages are not in $L_*^g SNP_*$ (in all cases, $card(V) = k \geq 2$):*

$$L_1 = \{x\, mi(x) \mid x \in V^+\},$$
$$L_2 = \{xx \mid x \in V^+\}.$$

Note that language L_1 above is a non-regular minimal linear one and L_2 is context-sensitive non-context-free.

We introduce now the mentioned distributed version of SN P systems:

An *SN dP system* is a construct

$$\Delta = (O, \Pi_1, \ldots, \Pi_n, esyn),$$

where (1) $O = \{a\}$ (as usual, a represents the spike), (2) $\Pi_i = (O, \sigma_{i,1}, \ldots, \sigma_{i,k_i}, syn, in_i)$ is an SN P system with request rules present only in neuron σ_{in_i} ($\sigma_{i,j} = (n_{i,j}, R_{i,j})$, where $n_{i,j}$ is the number of spikes initially present in the neuron and $R_{i,j}$ is the finite set of rules of the neuron, $1 \leq j \leq k_i$), and (3) $esyn$ is a set of *external synapses*, namely between neurons from different systems Π_i, with the restriction that between two systems Π_i, Π_j there exist at most one link from a neuron of Π_i to a neuron of Π_j and at most one link from a neuron of Π_j to a neuron of Π_i. We stress the fact that we allow request rules only in neurons σ_{in_i} of each system Π_i – although this restriction can be removed; the study of this extension remains as a task for the reader. The systems $\Pi_i, 1 \leq i \leq n$, are called *components* (or *modules*) of the system Δ.

As usual in dP automata, each component can take an input (by using request rules), work on it by using the spiking and forgetting rules in the neurons, and communicate with other components (along the synapses in $esyn$); the communication is done as usual inside the components: when a spiking rule produces a number of spikes, they are sent simultaneously to all neurons, inside the component or outside it, in other components, provided that a synapse (internal or external) exists to the destination.

As above, when r spikes are taken from the environment, a symbol b_r is associated with that step, hence the strings we consider introduced in the system are over an alphabet $V = \{b_0, b_1, \ldots, b_k\}$, with k being the maximum number of spikes introduced in a component by a request rule.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over V if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, working in the synchronous (in each time unit, each neuron which can use a rule should use one) non-deterministic way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

Hence, the SN dP systems are synchronized, a universal clock exists for all components and neurons, marking the time in the same way for the whole system.

In what follows, like in the communication complexity area, see, e.g., [8], we ask the components to take equal parts of the input string, modulo one symbol. (One also says that the string is distributed *in a balanced way*. The study of the unbalanced (free) case remains as a research issue.) Specifically, for an SN dP system Δ of degree n we define the language $L(\Delta)$, of all strings $x \in V^*$ such that we can write $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the computation halts. Moreover, we can distinguish between considering b_0 as a symbol or not, like in the previous sections, thus obtaining the languages $L_\alpha(\Delta)$, with $\alpha \in \{0, 1, 2, \ldots\} \cup \{\infty, *\}$.

Let us denote by $L_\alpha SNdP_n$ the family of languages $L_\alpha(\Delta)$, for Δ of degree at most n and $\alpha \in \{0,1,2,\ldots\} \cup \{\infty, *\}$. An SN dP system of degree 1 is a usual SN P system with request rules working in the accepting mode (with only one input neuron). Thus, the universality of SN dP systems is ensured, for the case of languages $L_\infty(\Delta)$.

In what follows, we prove the usefulness of distribution, in the form of SN dP systems, by proving that one of the languages in Corollary 1, can be recognized by a simple SN dP system (with two components), even working in the L_k mode.

Proposition 1. $\{ww \mid w \in \{b_1, b_2, \ldots, b_k\}^*\} \in L_{k+2}SNdP_2$.

Proof. The SN dP system which recognizes the language in the proposition is the following:

$$
\begin{aligned}
\Delta &= (\{a\}, \Pi_1, \Pi_2, \{((2,1),(1,3)),\ ((1,5),(2,1))\}),\ \text{with the components}\\
\Pi_1 &= (\{a\}, \sigma_{(1,1)}, \ldots, \sigma_{(1,7)}, syn_1, (1,1)),\\
&\quad \sigma_{(1,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \le r \le k\} \cup \{a^4a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}),\\
&\quad \sigma_{(1,2)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,3)} = (0, \{a \rightarrow a,\ a^3 \rightarrow a^3\}),\\
&\quad \sigma_{(1,4)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow \lambda,\ a \rightarrow a,\ a^4 \rightarrow a\}),\\
&\quad \sigma_{(1,5)} = (0, \{a^2 \rightarrow \lambda,\ a^6 \rightarrow a,\ a^6 \rightarrow a^3\}),\\
&\quad \sigma_{(1,6)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad \sigma_{(1,7)} = (0, \{a^+/a \rightarrow a\}),\\
&\quad syn_1 = \{((1,1),(1,2)),\ ((1,5),(1,1)),\ ((1,2),(1,4)),\ ((1,4),(1,6)),\\
&\qquad\quad ((1,4),(1,7)),\ ((1,6),(1,7)),\ ((1,7),(1,6)),\ ((1,2),(1,5)),\\
&\qquad\quad ((1,3),(1,4)),\ ((1,3),(1,5))\},\\
\Pi_2 &= (\{a\}, \sigma_{(2,1)}, \emptyset, (2,1)),\\
&\quad \sigma_{(2,1)} = (3, \{a^3/\lambda \leftarrow a^r \mid 1 \le r \le k\} \cup \{a^4a^+/a \rightarrow a,\ a^4 \rightarrow a^3\}).
\end{aligned}
$$

The proof that this system works properly, recognizing indeed the language in the statement of the proposition, can be found in [9].

Many problems can be formulated for SN P systems with request rules and for SN dP systems. Several were formulated in [9], from where we recall the following two general ones. :esides the synchronized (sequential in each neuron) mode of evolution, there were also introduced other modes, such as the exhaustive one, [11], and the non-synchronized one, [1]. Universality was proved for these types of SN P systems, but only for the extended case. Can universality be proved for non-extended SN P systems also using request rules?

5 Two Research Topics About SN P Systems

Many research topics and open problems about SN P systems can be found in the literature. We mention here only the collection from [14], and we recall two of the problems formulated in [15].

G. Continuing with SN P systems, a problem which was vaguely formulate from time to time, but only orally, refers to a basic feature of the brain, the memory. How can this be captured in terms of SN P systems is an intriguing question. First, what means "memory"? In principle, the possibility to store some information for a certain time (remember that there is a short term and also a long term memory), and to use this information without losing it. For our systems, let us take the case of storing a number; we need a module of an SN P system where this number is "memorized" in such a way that in precise circumstances (e.g., at request, when a signal comes from outside the module), the number is "communicated" without "forgetting" it. In turn, the communication can be to only one destination or to several destinations. There are probably several ways to build such a module. The difficulty comes from the fact that if the number n is stored in the form of n spikes, "reading" these spikes would consume them, hence it is necessary to produce copies which in the end of the process reset the module. This is clearly possible in terms of SN P systems, what remains to do is to explicitly write the system. However, new questions appear related to the efficiency of the construction, in terms of time (after getting the request for the number n, how many steps are necessary in order to provide the information and to reset the module?), and also in terms of descriptional complexity (how many neurons and rules, how many spikes, how complex rules?). It is possible that a sort of "orthogonal" pair of ideas are useful: many spikes in a few neurons (n spikes in one neuron already is a way to store the number, what remains is to read and reset), or a few spikes in many neurons (a cycle of n neurons among which a single spike circulates, completing the cycle in n steps, is another "memory cell" which stores the number n; again, we need to read and reset, if possible, using only a few spikes). Another possible question is to build a reusable module, able to store several numbers: for a while (e.g., until a special signal) a number n_1 is stored, after that another number n_2, and so on.

H. The previous problem can be placed in a more general set-up, that of modeling other neurobiological issues in terms of SN P systems. A contribution in this respect is already included in the present volume, [13], where the sleep-awake passage is considered. Of course, the approach is somewhat metaphorical, as the distance between the physiological functioning of the brain and the formal structure and functioning of an SN P system is obvious, but still this illustrates the versatility and modeling power of SN P systems. Further biological details should be considered in order to have a model with some significance for the brain study (computer simulations will then be necessary, like in the case of other applications of P systems in modeling biological processes). However, also at this formal level there are several problems to consider. For instance, what happens if the sleeping

period is shortened, e.g., because a signal comes from the environment? Can this lead to a "damage" of the system? In general, what about taking the environment into account? For instance, we can consider a larger system, where some modules sleep while other modules not; during the awake period it is natural to assume that the modules interact, but not when one of them is sleeping, excepting the case of an "emergency", when a sleeping module can be awakened at the request of a neighboring module. Several similar scenarios can be imagined, maybe also coupling the sleep-awake issue with the memory issue.

The first of these problems will be answered below.

6 A Memory Module

We start by directly given the memory module which answers the requests of problem **G** from [15]; we present it in a graphical form, in Figure 1, using the standard way of representing SN P systems (neurons as nodes of a graph, linked by arrows which represent synapses, with the rules of each neuron written in the respective nodes, together with the spikes initially present there; input and output neurons have incoming or outgoing arrows, respectively).

The system in Figure 1 works as follows. The number n is introduced in the memory module, in the form of a spike train containing n occurrences of 1, one after the other. The computation starts when the first spike enters the system – at that moment, rule $a^5/a^3 \to a$ is enabled. For the other spikes, the rule $a^3/a \to a$ is used, always two spikes remaining inside neuron σ_1. If no spike enters the system, then the two existing spikes are removed. If, at a subsequent step, any spike enters neuron σ_1, then it is forgotten, too, by means of the rule $a \to \lambda$. Thus, the number to be stored should be introduced as a compact sequence of spikes.

Each input spike is doubled by neurons σ_2, σ_3, and in this way $2n$ spikes are accumulated in neuron σ_4. They can stay here forever, unchanged. If the trigger neuron, σ_9, receives a spike (we assume that this happens after completing the introduction of the n spikes in neuron σ_1), then a further spike is sent to neuron σ_4. With an odd number of spikes inside, neuron σ_4 consumes two by two the spikes, moving in this way n spikes in the "beneficiary" neuron σ_{10}, at the same time moving $2n$ spikes to neuron σ_8. Note that after exhausting the spikes of neuron σ_4, one further spike is produced by σ_7, hence in the end neuron σ_8 gets an odd number of spikes. In the same way as σ_4 has moved its contents to σ_8, now the reverse operation takes place, hence σ_4 will end with $2n$ spikes inside (and σ_8 is empty). Therefore, the "cell memory" is restored, the number n can be read again, when the trigger gets one further spike. Always, the number n is made available outside the memory module, but the module "remembers" the number, for a further usage.

Now, may extensions can be imagined, for instance, towards data bases. A table in a data base can be imagined as a sequence of "memory cells" (modules), each one with its own label and value (number stored) and subject to updating

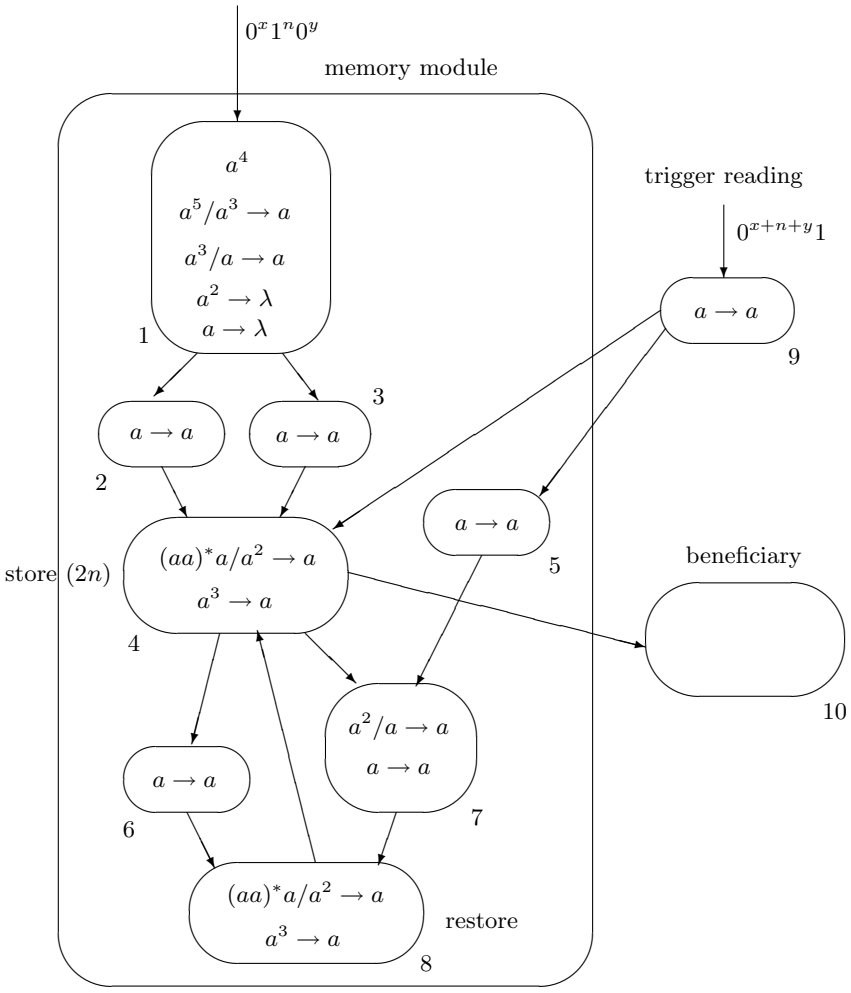

Fig. 1. A memory module

and interrogating operations. Modules can also be inserted and removed. Each of these operations are done by means of a specialized module, one for updating, one for interrogating, and so on, always specifying first the "address" (the label, the ID) of the cell which is operated. The construction becomes complex (only the indicated modules should be modified, although the trigger for the respective operation can do the same with all other modules), but presumably feasible; we leave this task to the reader.

Another direction of research is to consider other "brain modules" or functions and implement them in terms of SN P systems. An example is that from [13], where

the sleeping activity of the brain is modeled, but many others can be addressed: learning, getting tired, taking the task of another part of the brain.

7 Pattern Recognition

One function of the brain which we have not mentioned above is pattern recognition. Neural computing is especially concerned with this task, and learning (training the net) is an essential step of pattern recognition. Although the problem is so natural, only a few efforts were paid in SN P systems area to incorporating the learning feature, see, e.g., [7]. In turn, nothing was done in what concerns the patterns recognition, that it why we call here the attention about this problem.

A simple framework for that concerns handling – generating or recognizing – arrays, pictures realized by marking the positions of a grid with symbols of a given alphabet, in the sense of array grammars and languages. For instance, we can imagine the following task. Consider a language of arrays, for instance, the letters of the Latin alphabet, each one written in a rectangle with black and white lattice positions. We can interpret a marked spot (black) as a spike and a non-marked one (white) as no spike. Let us construct an SN P system having certain input neurons aligned, able to read one by one the rows of the array picture. The reading can be done in consecutive steps or internal computations are allowed between reading two neighboring rows. After reading the whole array, the system can continue working and, if the computation halts, the input array is recognized.

The difficulty of constructing such an SN P system lies in the fact that we have to recognize several different letters. A variant is to construct a system which recognizes only one of the letters, say A, but of different sized. In this latter case, the number of input neurons should be large enough for reading all possible sizes of the letter, and the difficulty lies again in the fact that the same neurons can behave differently for letters of different sizes.

Anyway, we find the construction of such patter recognition SN P systems interesting, non-trivial and rather instructive, a good exercise for understanding the functioning of SN P systems, and, hopefully, a way to find applications for them.

8 Final Remarks

We insist about the fact that this is only an invitation to a dynamical research area of membrane computing, trying to convince the reader that there are interesting problems to address and providing a few bibliographical hints. For comprehensive presentations and references, the reader should consult the domain literature, available at [23] (for instance, all brainstorming volumes can be found there, in a downloadable form).

Acknowledgements

The work of M. Ionescu was possible due to CNCSIS grant RP-4 12/01.07.2009. The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 24-25 (2009), 2352–2364.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162
3. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
4. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University. Mathematics-Informatics Series*, 63 (2009), 5–22.
5. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
6. R. Gutierrez-Escudero et al.: *Proceedings of the Seventh Brainstorming Week on Membrane Computing*. Sevilla, 2009, 2 volume, Fenix Editora, Sevilla, 2009.
7. M.A. Gutierrez-Naranjo, M.J. Pérez-Jiménez: A first model for Hebbian learning with spiking neural P systems. In *Proc. 6th Brainstorming Week on Membrane Computing*, Sevilla, 2008.
8. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing*. Springer, Berlin, 1997.
9. M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez T. Yokomori: Spiking neural dP systems. In *Proc. 9th Brainstorming Week on Membrane Computing*, Sevilla, January 31– February 4, 2011.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
11. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
12. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, 1999.
13. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In [6], vol. 2, 59–72.
14. Gh. Păun: Twenty six research topics about spiking neural P systems. In *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, Fenix Editora, Sevilla, 2007, 263–280
15. Gh. Păun: Some open problems collected during 7th BWMC. In [6], vol. 2, 197–206.
16. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.

17. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Lecture Notes in Computer Science*, 6570, in press.
18. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, in press.
19. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
20. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
21. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
22. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
23. The P Systems Website: `http://ppage.psystems.eu`.