Spiking Neural P Systems with Memory

Ammar Adl†, Amr Badr†, Ibrahim Farag†

†Computer Science Department, Faculty of Computers and Information, Cairo University

Summary
This paper is an attempt to discuss the idea of memory into the area of spiking neural P Systems; we introduce a draft hybrid model of SNP system encapsulating a dynamic arrays System inside. In this device, we use the aspects of dynamic array management to construct a memory sub-system inside the atomic building block of the SNP System, which is the single neuron. And as a computing device, the proposed machine, we will call it the SNP A-Machine. We also try to visit broadly the learning mechanism for the machine, through a learning model, incorporating the time concept using some forget factor *f* and a recall factor *r*.

Key words:
Spiking neural P System, Single neuron, Shot-term memory, long-term memory, SNP A-Machine.

1. Introduction

A Spiking Neural P system is a class of P systems inspired by the functioning of neural networks, and the ways they use to exchange signals through their specialized junctions called chemical synapses. Go to [10] for more details. SNP System is a construction of a networked membranes hosting a multi-set of objects and being in a certain state according to which objects are dealt with. The communication channels among different cells are specified in advance and correspond to axons in neural cells. To consider a spiking neural P system - recalling from [10] - of degree m ≥ 1, in 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. $\sigma_1, \dots, \sigma_m$ 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 by the cell;
b) R_i *is* a finite set of *rules* of the following two forms:
(1) $E/a^r \rightarrow a; t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$;
(2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin E/a^r \rightarrow a; t$ of type (1) from *Ri; L(E)* for any rule.
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (synapses among cells);
4. $i_0 \in \{1, 2, \dots, m\}$ *indicates the output neuron.*

First type of rules are the *firing rules*: provided that the contents of the neuron, is introduced by the regular expression *E*, and there are *r* spikes are consumed, the neuron is fired, and it produces a spike which will be sent to other neurons after *t* time units, considering the usage of a global clock all across the system, identifying the time for the whole system, hence the functioning of the system could be set as a synchronized model.

There are two actions that take place in a single step: firing and spiking. A neuron fires when using a rule $E/a^r \rightarrow a; t$, this is only if the neuron contains *n* spikes and $a^n \notin L(E)$ and $n \geq r$. The regular expression *E* represents the contents of the neuron.

Here, at the level of a single neuron computation is in sequential mode, i.e. a single rule is to be fired at each step. Still, the maximal parallelism is at the level of the whole system, in the sense that in each step all neurons which can evolve (use a rule) have to do it. For spiking, the use of a rule $E/a^r \rightarrow a; t$ in a step q means firing in step q and spiking in step $q + t$. That is, if $t = 0$, then the spike is produced immediately, in the same step when the rule is used. If $t = 1$, then the spike will leave the neuron in the next step, if we consider that t is represented by a time interval $t = \{0, \dots t_n\}$, then moving from $0 \rightarrow t_n$ *in time* will be simulating a recalling factor (*r*) and moving from $t_n \rightarrow 0$ *in time* will be simulating a forgetting process by a forget factor (*f*). In the time between firing a rule and producing a spike, the neuron is assumed to be building up for next firing stage (the refractory period); so it will not be able to accept any more incoming spikes, this much like going into a short hibernation state.

This means that if $t \geq t_n$ and another neuron produces a spike in any moment, then its spike will not pass to the hibernated neuron which has used some rule R in some step q. In the moment when the spike is out, the neuron can be activated once again, and accept new spikes. This means that if $t = 0$, then it can be considered as a passive gateway with no delay; the neuron can receive spikes in the same step when using the rule. Similarly, the neuron can receive spikes in moment t, in the case $t \geq 1$. [10]

Manuscript received October 5, 2011
Manuscript revised October 20, 2011

2. The SNP A-Machine

From a general perspective, SNP systems are devices constructed by number of computational nodes, which are considered atomically built, they are the neurons, we here take a further closer look, to these devices. A neuron in its basic biological structure is considered a living cell, this leads to the idea that it encapsulates the features of a specialized sub-system composed of a group of arrays and a synchronizer or processor, representing some computational model inside, a question comes up here, what if we merge the two models; of SNP systems and the new dynamic arrays sub-system inside? A closer look to the neuron itself, based on the fact that if the same inputs are fed to the neuron it produces the same outputs, so it could be a complete computational machine. Some types of neurons deliver different outputs while time passes, more evidence on the neuron containing some kind of memory, and a learning mechanism. [2]
Some premises could be considered here:

(i) There are already existing rules inside the neuron. (Initially to be considered inside the long term memory).
(ii) Rules are set by biological evolution and genetic structures. (Initial Configuration).
(iii) The system initial set of rules, leads to specialization of the machine, in biology there are different specialized neurons for different processing schemas.[4]
(iv) A neuron is preprogrammed by some initial firing rules in biology.
(v) A sub-system of a predefined structure could be set – just for simplicity we will use three arrays, Rules Store, and a Synchronizer– representing the neuron soma, we will refer to this sub-system by (Λ)**:**

- The first array will be holding the rules indexes, to be consulted at first time a sequence of objects is sent to the neuron, to check if there is any matching results from previous computation.
- The second array will be representing the short-term memory.
- The third array will be representing the long-term memory.
- Rules will be residing into some sort of storage device; we will represent it via a *RulesDB*.
- A Synchronizer (inner neuron processor), that will be responsible for firing the rules and conducts the search and match process inside the memory.

(vi) There is a sub-system for the initial computational steps, the inner processor or the synchronizer.
(vii) A set of presynaptic links are introduced to the machine, as the computational input channels, carrying the spikes incoming the system.
(viii) A set of postsynaptic links are introduced to the machine, as the computational output channels, carrying the outgoing spikes, forming the result.
(ix) The neuron itself is represented via an oval figure. (The computational environment).

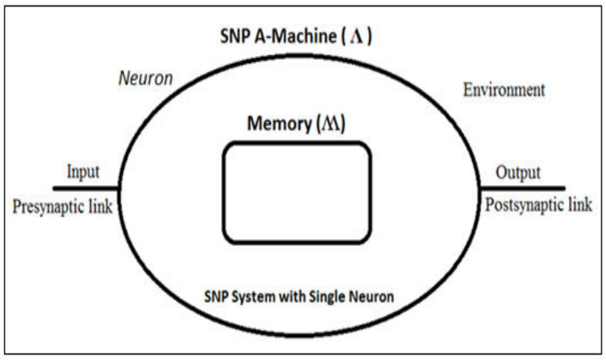

Fig.1. The general SNP A-Machine structure

3. Using Arrays in SNP A-Machine Memory

Using arrays to represent memory encapsulates some aspects that will help us in this model, firstly indexing, which will fasten search procedures, secondly storage of the previously computed results, thirdly sorting the fired rules, fourthly the insertion and deletion processes of the memory. For simplicity, we will use three arrays as mentioned, for storing rules' indexes, short and long term memories, which hold the results of the last successful computations conducted, the design of the arrays will be based on the initial configuration of the systems, to speed-up computations, this will lead to static configurations in some cases, also might lead to some dynamic configuration in others, but here we will be discussing the static type.

Each time a spike is produced by the neuron, we store that action; rules are into the processor, and results in the short-term memory array, with a time counter called the recall factor (r), set to a high recalling potential n, and decreasing by a time unit t, when this decreasing rate crosses some certain time threshold, let's say T_h, then results are moved over to the long-term memory array, emptying the short-term one for the upcoming computations results, meanwhile each time unit is decreased from r is increased into another direction, a forget factor (f), forgetting contrasts recalling, every time unit passes, a single result –spike– of a previous computation is being more forgotten, increase $(f) = \{0,..,n\}$, $\mathrm{tm}(f) = 0 \rightarrow n;\ \ n \geq t \geq 0$, and will be difficult to recall, in our structure, this means decrease(r) $= \{0,..,n\}$, $\mathrm{tm}(r) = \mathrm{n} \rightarrow 0;\ \ n \geq t \geq 0$, tm is a time function. Rules inside the processor of last computation are now attached with the new forget factor value, and when the recall factor r has crossed the time threshold T, it is set once again to the value n, now the processor carries one type of rules, that need to count down their f values to be

fired again, so the incoming spike trains will eventually - along with their firing rules- sort these rules according to the most fired and used, this will give some indications of how the firing schema might be more precise by time, which is a dynamic factor here based upon two equal values $\{r, f\}$, but opposite in polarization, until $r \lessdot Th$, then it is set to its initial value.
In order to formalize the idea of rule sorting in the framework of SN P systems and inspired by the concepts introduced in [5], [11], we use some type of extended rules: the *memory rules*, they are rules of the form:

(i) F rules: $E/a^k \rightarrow (a^m, f); t$

(ii) R rules: $E/a^m \rightarrow (a^k, r); t$

where, E is a regular expression over $\{a\}$, k and m are natural numbers with $m \geq k \geq 0$, $t \geq 0$ and $f = (f_1, f_2, \ldots, f_n)$ is a finite sequence of natural numbers called the forgetting sequence where $f_1 = k$ and $f_n = 0$. If $E = a^k$, The idea behind the forgetting sequence is the following, when the rule $E/a^k \rightarrow (a^m, f); t$ is triggered at t_0 we look in $f = (f_1, \ldots, f_n)$ for the greatest i such that $m \geq f_i$. And also $r = (r_1, r_2, \ldots, r_n)$ is a finite sequence of natural numbers called the recalling sequence, where $r_1 = m$ and $r_n = 0$. If $E = a^k$, When a new input is inserted in the system to start a new computation, first we access the short-term memory array, if there are any spikes found that match the incoming ones, and before the computation ends, then a copy of the solution is sent from that memory array to the system - revise [5] for another implementation schema- search is stopped, now the processor decreases the recall factor *r value* for the stored rules from type (R) above, and increases the forgetting factor *f value* for the type (F) rules.

This means that if a match is found in the short-term memory this means that last computation rules were the correct ones to fire, so, let the system recall them faster next time, and other stored rules are being forgotten by time, and in order, now it is obvious that this style is limiting the non-determinism rules firing mechanism, and adapting the system by time, to some certain problems to compute, which will be very useful in the learning process, recall [11]. On the other hand, if the solution is not found in short-term memory subsystem, then go and check the long-term memory one, if found a result, then copy it to the short-term memory, update this particular result's rule to be a recalling one, empty the long-term array, delete the F rules, except of course the one with the result, for it is now updated to be of the recall type R, now update the R rules with *r values*, this might be described as a long recall action.

4. General Design for the SNP A-Machine Memory

As told above, the main idea is to define a system which allows storing information of previously executed computations, in order to save computation time, and make the system more realistic by putting its rules in some sorted manner, based on their firing frequency and timing. Hence, the final system (we will call it the SNP A-Machine Memory denoted by ʍ) is designed by adding an original computation system (i.e., the processor, designed to work on the rules, synching among rules firing, sorting, and finding or matching input data with the previously stored results), a short-term memory array, and a long-term memory array.

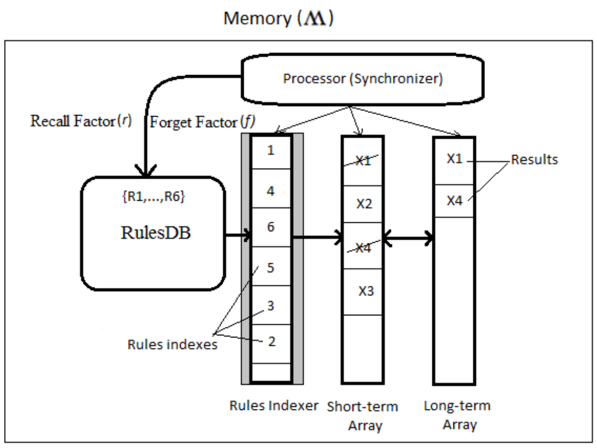

Fig.2 A general structure of the SNP A-Machine Memory (ʍ)

The general structure of the memory system is depicted in Fig.2. There are five main constructs in this view:

The Processor (Synchronizer)
Holds system rules, that perform computations on the input spikes that are injected from outside. It works on the used firing rules, synchronizing action with memory arrays due to time units, this by updating constantly the *r, f values,* or deleting the R rules.

The Indexer
A dynamic array used to store the indexes of the recently fired rules.

The long-term Memory array
A dynamic array used to store old spiking results, whose forget factor *f* is of high value.

The Short-term Memory array
A dynamic array used to store fast and latest spiking operations concerning solutions the system is currently computing, carrying the freshest spikes for the last action of the system. As told above, at the end of the

computation the storage device has to store the new spikes, along with the fired rules indexes.

The Rules Repository (RDB)
A structure used to hold system rules, it is consulted every time a process takes place by the processor, if a stored rule is fired, then raise its index, pass it to the processor, which in return, works on this index passing it to the indexer.

5. How it works

Using the above constructions, and modules, we tried to lay some closure to the idea, of how the processing of the incoming spikes might take place. We will try here to informally describe an algorithm for the process, under some certain constraints in the computational environment; there are three broadly speaking stages:

(i) Configuring the SNP A-Machine, set values for recalling and forgetting schemas, and the processing timeline in time units, and the time threshold mentioned above.
(ii) Processing: managing the incoming spikes train, the match searching, and rule firing.
(iii) Output: used in search, checking for results into the memory's arrays.

Configuration stage:

```
SET $T_h$.value (time threshold) =some initial non-negative integer value.
SET r.value (recall factor value) =some initial non-negative integer value.
SET f.value (forget factor value) = 0,
SET T.value (global machine timer value) = non-negative integer value $> 0$.
SET (word) = predefined number of spikes to be searched with, thus invokes a certain rule Ri
```

Output stage:

```
PROCEUDRE (OUTPUT)
BEGIN
READ incoming search_pattern.
DEFINE Search_Value = Search_Pattern.
// match search with all previously computed results stored into short-term memory array first.
IF Search_Value = any result in short-term memory array.
THEN HALT, RETURN TURE. //halt the system, success.
ELSEIF
// if not in short-term memory then go to the long-term one and repeat the step.
IF match_pattern = result in long-term memory array.
THEN HALT, RETURN TRUE. // halt, success.
// if not in neither short-term nor the long–term memory then go to the Rules store and search.
ELSE RETURN FALSE.  // not in memory.
END
```

Processing stage:

```
PROCEUDRE (PROCESS_INCOMING_SPIKES_TRAIN)
BEGIN

IF SNP A-machine status = not in refactory period.
SET match_pattern = word.
READ incoming spikes train.
WHILE the whole train
//match search with all previously computed results stored into memory arrays first.
CALL PROCEUDRE (output)
IF output = TRUE.
THEN HALT, RETURN match_pattern. // success
// if not in neither short-term nor the long–term memory then go to the Rules store and search.
ELSEIF output = FALSE.
match_pattern = match search with all active rules in RDB in parallel.
IF R.E (E regular expression of rule R) = match_pattern.
THEN CALL PROCEDURE (Processer).
ELSE HALT // nothing matched, wait.

END
```

```
PROCEUDRE (PROCESS)

START global machine timer
WHILE time counter is true

BEGIN PROCESSING
CALL PROCEDURE (PROCESS_INCOMING_SPIKES_TRAIN).
FOREVERY fired rule $R_i$ ;

BEGINLOOP on j;
IF (r $>$ $T_h$ ) SET indexer.indexes(j) = i; //store index of the fired rule.
ASSIGN a time_stamp to $R_i$ = r.value;         // current value of the recall factor.
ASSIGN $R_i$ = R type; //now it is a rule of type (recall rule).
INSERT result xi into the short-term memory array.
DECREASE r.value by one;           // now we need one time unit to recall this rule.
INCREASE f.value by one;
```

```
// system is going to forget this rule for one time unit
in next round.
ENDLOOP

IF (r ≤ T_h)

BEGINIF
SET time_stamp = f.value.
ASSIGNALL {Ri,…,Rn} = F type; //now all
rules fired inside the processor are of type
forget rules.
COPYALL previously computed results
{Xi,…,Xn} from short-term to long-term memory
array.
IF(r = 0) // recall factor reaches 0 time unit.
THEN
SET r.value = r.initial value. // reinitializing the
recall factor.
SET f.value = 0 // reinitializing the forget factor.
DELETE long-term memory array content. //
obsolete values.
RETURN TO BEING PROCESS.
ENDIF
END PROCESSING
```

In (Table.1) we notice the flipping between the R rules to F rules, and how the Rules inside memory are moving, towards some ranking schema that will contribute into the rule firing selection and results mapping, from short-term memory to the long-term one. This will speed up the computation process and trend to a linear processing style, deducing the non-determinism into the original system.

6. Proposing the SNP A-Machine logical Model (Λ)

Theorem.1
An SNP A-Machine (Λ) is the atomic component of an SNP system, the neuron, including an array based memory sub-system.

Definition.1

Λ = A Single Neuron SNP System.
+
A dynamic array based memory sub-system (ʍ).

The SNP System is with a single neuron, representing the computation container, carrying some firing and forgetting rules. And the memory sub-system representing the memory, might be carrying a set of rules. The system adapts a fixed topology, memory, recall and decay [11] (Forget) factors. It has the following features:

– There could be an initial number of the spikes inside the (Λ) or not, i.e. Si $\in S$ (Si); i $\geq$ 0.

– There is one or more presynaptic links, carries the input into the machine.
– There is one or more postsynaptic links, carry the output to the environment.
– There exists a memory structure inside the machine we will refer to it by (ʍ).
– The memory is constructed in its simplest form of a short term memory "buffer" structure, and a long term one, depicted by two separated arrays.
– There are preset rules inside the scope of the machine – initial configuration.
– The memory is represented a dynamic array based sub-system.
– Rules are two types forgetting and firing.
– The decay (Forget) factor *f* is instantiated and attached to each computation taking place.
– The recall factor *r* is also instantiated and attached to each computation taking place.
– Computation succeeds when it halts, and output spikes are sent to the system over the postsynaptic link, to the computation environment.

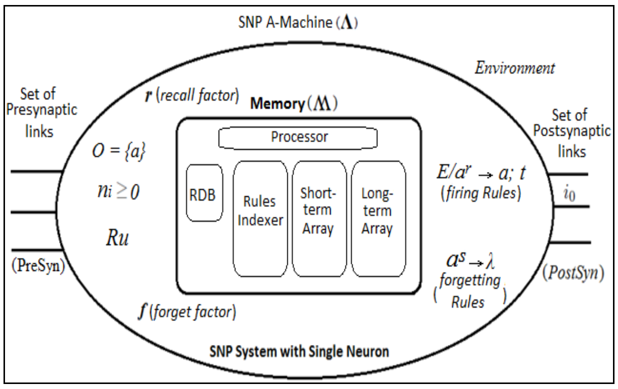

Fig.3. A closer look on The SNP A-Machine Structure

Bearing in mind these features, we describe an SNP A-Machine System in the following way:

Definition.2. *An SNP A-Machine unit of degree m, m ≥ 1, is a construct:*

$$\Lambda = (O, PreSyn, f_1, \ldots, f_r, r_1, \ldots, r_c, N, Ru, ʍ, PostSyn),$$

Where,
– *O = {a} is the alphabet (the object a is called spike);*
– *PreSyn* $\subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ *with* $(i, i) \notin$

PreSyn for $1 \leq i \leq m$ *(input synapses); where:*

– *$\mathcal{M}$ is a dynamic array based sub-system.*

– *N = {n}, ni ≥ 0; is the initial number of spikes contained by the machine;*

– *Ru is a finite set of rules {Rs, R, F} of the following forms:*

1. *Rs where:*
1. *$E/a^r \rightarrow a; t$. E is a regular expression over O, $r \geq 1$, and $t \geq 0$; (firing Rule).*
2. *$a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \rightarrow a; t$ of type (1) from Rsi; (forgetting Rule).*

2. *$R = \{Rr_{i1}, \ldots, Rr_{il_1}\}$, for each $i \in \{1,\ldots, m\}$ and $j \in \{1,\ldots, l_i\}$, where:*

*$E/a^k \rightarrow (a^m, r)$; t is a rule with **recalling** factor, $k \geq m_{ij} \geq 0$ and $r_{ij} \geq 0$. The sequence $f = (r_1, r_2, \ldots, r_c)$ is a finite sequence of natural numbers called the recalling sequence where $r_1 = k$ and $r_c \geq 0$. Inspired by rules in* [11].

3. *$F = \{Rf_{i1}, \ldots, Rf_{il_1}\}$, for each $i \in \{1,\ldots, m\}$ and $j \in \{1,\ldots, l_i\}$, where:*

*$E/a^k \rightarrow (a^m, f)$; t is a rule with **forget** factor, $k \geq m_{ij} \geq 0$ and $f_{ij} \geq 0$. The sequence $f = (f_1, f_2, \ldots, f_r)$ is a finite sequence of natural numbers called the decaying (forgetting) sequence where $f_1 = k$ and $f_r \geq 0$. Inspired by rules in* [11].

– *PostSyn $\subseteq \{1, 2,\ldots, m\} \times \{1, 2,\ldots, m\}$ with $(i, i) \notin$ PostSyn for $1 \leq i \leq m$ (indicates the output neuron postsynaptic link.);*

Definition.3

An input for a SNP A-Machine of degree m is a vector $x = (x_1, \ldots, x_m)$ of m spikes objects x_i.

An SNP A-Machine with input is a pair (Λ, x) where Λ is SNP A-Machine and x is the spikes train as an input for it.

7. The Learning Model

From the previous arguments, we come up with the idea of rule selection, which rules are better to be chosen than the others, and when to invoke a certain rule to be fired, given that an input to the system x, success can be reached or not, in other words, was there a result sent to the outer system or not? If the choice of some certain firing path leads to the success with a higher probability than the choice of another path, then the device computation process is getting better by every time unit passes. Formally, a learning problem - extending the one shown in [11]- could be constructed as a tuple of an SNP A-Machine, an input vector for the machine, a learning schema or function, a learning time interval, a threshold for recalling and forgetting a certain set of rules, a learning curve slope (*Lr*):

Definition.4

A Learning SNP A-Machine is:

$$AL = (\Lambda, X_{input}, L, T, Th, Lr), \text{ where:}$$

– Λ is an SNP A-Machine.

– $X_{input} = \{x1,\ldots, xn\}$ is a finite set of inputs of Λ.

– L: $Z \rightarrow Z$ is a function from the set of integer numbers onto the set of integer numbers. The learning function.

– T = {*t1,..., tn*} is a finite set of time stamps attached to the inputs for Λ.

– ***Th*** = {*rth, fth*} is a set of two values for recall and forget values thresholds.

– *Lr* is a positive value called the learning rate.

We have developed a software simulator that employs the latter problem structure, and an SNP A-Machine, with multiple simulation experiments, initial outputs tend to be more closer to the realistic situation where a computational device with memory and a learning schema, over time, is delivering much faster results than another device without a memory involved, also the curve of leaning is tending to be linear rather than an exponential one, by time, the learning function, is contributing into the computational process positively, lessening the non-determinism in rules firing, and shrinking the search space for a certain halt state for the system.

8. Conclusions and Future work

From previous observations we found that including a memory structure into the SNP system model would lead to some more efficiency in the computation process, focusing on the atomic building block of such systems, the single neuron, and merging the two models of SNP systems and the dynamic arrays systems, affected the whole process of the device, and its speed, we also found that the learning process is highly affected by some factors like, the structure of the short and long term memories (ʍ), the recall and forget factors {*r, f*}, the time threshold for them, the mapping in the learning schema, the learning rate, all affect the device performance, next steps will be trying to study the structure of the memory subsystem in more details, from the design view, also we will visit the area of networks of SNP A-Machines in order to study the effect of memory modules in larger systems. The learning schema needs to be studied on a larger scale, in order to see how far a network of such machines can maintain information, and speed up the computation process; also what are the results when trying to solve an NP-Complete problem. The concept above leads consequently to the idea of specialized neurons – ones that are dedicated to solve certain problems– with learning and memory, a neuron can evolve by time to be treated as standing alone computation device of some purpose, even with the same initial configurations for two similar neurons, a trained neuron will be faster in processing than a non-trained one, this is much like getting old, and building experience in real life biological situations.

Table.1. Rules with their results movement into the system's memory.

T=10 (time units)	T_h=1 Fired Rule	Rules Indexes	Time stamp	Processor	Type R/F	Short-term array	Long-Term array	r=4	f=0	Notes
1	R_1	1	4	$\{R_1\}$	R	$\{X_{i1}\}$	-	4	0	First Rule to fire R_1
2	R_2	2	3	$\{R_2,R_1\}$	R	$\{X_{i2},X_{i1}\}$	-	3	1	
3	R_5	5	2	$\{R_5,R_2,R_1\}$	R	$\{X_{i5},X_{i2},X_{i1}\}$	-	2	2	
4	-	-		$\{R_5,R_2,R_1\}$	R	$\{X_{i5},X_{i2},X_{i1}\}$	-	1	3	
5	-	-	4	$\{R_5,R_2,R_1\}$	F	-	$\{X_{i5},X_{i2},X_{i1}\}$	0	4	Copy short-term to long –term memory
6	R_4	4	4	$\{R_4\}$ $\{R_5,R_2,R_1\}$	R,F	$\{X_{i4}\}$	$\{X_{i5},X_{i2},X_{i1}\}$	4	0	Delete long-term memory content with their corresponding rules
7	R_2	2	3	$\{R_4,R_2\}$	R	$\{X_{i4},X_{i2}\}$	-	3	1	
8	-	-	-	-	-	-	-	2	2	
9	R_3	3	1	$\{R_4,R_2,R_3\}$	R	$\{X_{i4},X_{i2},X_{i3}\}$	-	1	3	
10	-	-	-	-	-	-	-	-	-	Other incoming trains

References

[1] Estimation of single-neuron model parameters from spike train data Randall D. Hayes, John H. Byrnea, Steven J. Cox,Douglas A. Baxter.

[2] Single-Neuron Responses in Humans during Execution and Observation of Actions. Current Biology 20, 750–756, April 27, 2010 Elsevier.

[3] Gh. P˘aun: Membrane Computing–An Introduction. Springer-Verlag, Berlin, 2002.

[4] W. Gerstner, W Kistler: Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge Univ. Press, 2002.

[5] P Systems with Memory Paolo Cazzaniga, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Lecture Notes in Computer Science 3850.

[6] Gh.P˘aun: Computing with membranes – A variant: P systems with polarized membranes. Intern. J. of Foundations of Computer Science, (2000), 167–182.

[7] Gh.P˘aun, P systems with active membranes, Attacking NP–complete problems. Journal of Automata,Languages and Combinatorics, 6, 1 (2001), 75-90.

[8] P systems web page http://ppage.psystems.eu/.

[9] Computational and modelling power of P systems Tesidi Dottorato di Daniela Besozzi. Anno Accademico 2002-2003.

[10] M. Ionescu, Gh. P˘aun and T. Yokomori: Spiking neural P systems. Fundamenta Informaticae, 71, 2-3, 279-308, 2006.

[11] A First Model for Hebbian Learning with Spiking Neural P Systems. Miguel A. Guti´errez-Naranjo, Mario J. P´erez-Jim´enez Research Group on Natural Computing.

[12] G. Paun. Computing with membranes. Journal of Computer and System Sciences, 61, 1 (2000), and Turku Center for Computer Science-TUCS Report No 208.