A software tool for verification of Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo · Mario J. Pérez-Jiménez · Daniel Ramírez-Martínez

Abstract The formal verification of a Spiking Neural P System (SN P Systems, for short) designed for solving a given problem is usually a hard task. Basically, the verification process consists of the search of invariant formulae such that, once proved their validity, show the right answer to the problem. Even though there does not exist a general methodology for verifying SN P Systems, in (Păun et al., Int J Found Comput Sci 17(4):975–1002, 2006) a new tool based on the transition diagram of the P system has been developed for helping the researcher in the search of invariant formulae. In this paper we show a software tool which allows to generate the transition diagram of an SN P System in an automatic way, so it can be considered as an assistant for the formal verification of such computational devices.

Keywords Membrane computing · Spiking Neural P systems · Transition diagram

1 Introduction

Natural Computing studies new computational paradigms inspired from various well known natural phenomena in physics, chemistry and biology. It abstracts the way in which nature computes, conceiving new computing models. There are several fields in Natural Computing that are now well established. Among them, *Genetic algorithms* introduced by Holland (1975) is inspired by natural evolution and selection in order to find a good solution in a large set of feasible candidate solutions; *Neural Networks* introduced by

M. A. Gutiérrez-Naranjo (✉) · M. J. Pérez-Jiménez · D. Ramírez-Martínez
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Sevilla, Spain
e-mail: magutier@us.es

M. J. Pérez-Jiménez
e-mail: marper@us.es

D. Ramírez-Martínez
e-mail: danrammar@us.es

McCulloch and Pitts (1943) which is based on the interconnections of neurons in the brain; *DNA-based* molecular computing, that was born when Adleman (1994) published a solution to an instance of the Hamiltonian path problem by manipulating DNA strands in a lab.

In the framework of Natural Computing, Păun introduced *Membrane Computing* in Păun (2000) under the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations.[1] Since then, a large number of variants have been considered, concerning both the syntax and the semantics of the model. The devices of this model are generically called *P systems*.

The basic idea is to consider a distributed and parallel computing device, structured like a cell, by means of a hierarchical arrangement of membranes which delimit compartments where various chemicals (we call them *objects*) evolve according to local reaction rules. The objects can be described by symbols or by strings of symbols from a given alphabet. These objects can also pass through membranes, under the control of specific rules. Because the chemicals from the compartments of a cell are swimming in an aqueous solution, the data structure we consider is that of a *multiset*—a set with multiplicities associated with its elements. Also, in close analogy with what happens in a cell, the reaction rules are applied in a parallel manner. This means that in each computational step a maximal (multi)set of non-deterministically chosen rules is applied.

The notions of membrane investigated in this new paradigm of computation are abstract entities which try to mimic some features of the functioning of membranes in living cells. The basic function of biological membranes is to *define compartments* and to *relate compartments to their environment*, including neighbouring compartments. The currently accepted model of the membrane structure is the so-called *fluid-mosaic model*, proposed in 1972 by S. Singer and G. Nicholson. According to this model, a membrane is a phospholipid bilayer in which protein molecules (as well as other molecules) are totally or partially embedded.

Spiking Neural P Systems (SN P Systems, for short) were introduced in Ionescu et al. (2006) with the aim of incorporating in membrane computing ideas specific to spike-based neuron models. The intuitive goal was to have a directed graph where the nodes represent the neurons and the edges represent the synaptic connections among the neurons. The flow of information is carried on the action potentials, which are encoded by objects of the same type, the *spikes*, which are placed inside the neurons and can be sent from presynaptic to postsynaptic neurons according to specific rules and making use of the time as a support of information.

The biological motivation for these new SN P Systems is based on recent discoveries on neural coding. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes which matter. Traditionally, it has been thought that most, if not all, of the relevant information was contained in the *mean* firing rate of the neuron. The concept of mean firing rates has been successfully applied during the last 80 years (see, e.g., Mountcastle 1957 or Hubel and Wiesel 1959) from the pioneering work of Adrian (1926a, b). Nonetheless, more and more experimental evidence has been accumulated during recent years which suggests that a straightforward firing rate concept based on temporal averaging may be too simplistic to describe brain activity. One of the main arguments is that reaction times in behavioural experiment are often too short to allow long temporal averages. For instance, recognition

[1] A comprehensive presentation can be found in Păun (2002) and further updated bibliography in http://ppage.psystems.eu/. A presentation of applications can be found in Ciobanu et al. (2006).

and reaction involve several processing steps from the retinal input to the finger movement at the output. If at each processing steps, neurons had to wait and perform a temporal average in order to read the message of the presynaptic neurons, the reaction time would be much longer. Many other studies show the evidence of precise temporal correlations between pulses of different neurons and stimulus-dependent synchronisation of the activity in populations of neurons (see, for example, Eckhorn et al. 1988 or Gray and Singer 1989). Most of these data are inconsistent with a concept of coding by mean firing rates where the exact timing of spikes should play no role.

Instead of considering mean firing rates, spiking-based models consider the realistic situation in which a neuron abruptly receives an *input* at a given instant and for each neuron the timing of the first spike after the reference signal contains all the information about the new stimulus.

The paper is organised as follows: in Sect. 2 we recall some definitions related to SN P Systems (further information can be found in the literature, see http://ppage.psystems.eu/). Section 3 is devoted to general aspects of simulation software in the framework of P systems, presenting the most relevant works until now. Next the simulator is presented and some features of its implementation are given. Section 5 provides an example of how the simulator works and finally, in the last section some remarks and future work lines are presented.

2 Spiking Neural P Systems

An SN P System consists of a set of neurons placed in the nodes of a directed graph and sending signals (called *spikes*) along the arcs of the graph (providing the *synapses* between neurons). The objects evolve according to a set of rules (called *spiking rules*). The idea is that a neuron containing a certain amount of spikes can consume some of them and produce other ones. These produced spikes are sent (maybe with a delay of some steps) to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*. By application of these rules no spikes are sent, simply some spikes are removed from the neuron where the rule was applied. A global clock is assumed and in each time unit each neuron which can use a rule should do it, but only (at most) one rule is used in each neuron. One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment.

Definition 1 An SN P System of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out)$$

where

- $O = \{a\}$ is the singleton alphabet (the object a is called spike);
- $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i)$ with $i \in \{1, \ldots, m\}$ where:
  - $n_i \geq 0$ is the initial number of spikes contained by the neuron σ_i;
  - R_i is a finite set of rules of the form

$$E/a^c \rightarrow a^p; d$$

where E is a regular expression over a, $c \geq 1$ and p, $d \geq 0$, with $c \geq p$; *if* $p = 0$, then $d = 0$ too.

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $i \in \{1,\ldots,m\}$ (synapses);
- $out \in \{1, 2,\ldots,m\}$ is the output neuron.

For the sake of simplicity, if $p = d = 0$, the rule is written as $E/a^c \rightarrow \lambda$ instead of $E/a^c \rightarrow a^0; 0$. This type of rules is called *forgetting rules*. If $L(E) = \{a^c\}$ then the rules are written in the simplified form $a^c \rightarrow a^p; d$ and $a^c \rightarrow \lambda$.

With respect to the application of the rules, if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \rightarrow a^p$; $d \in R_i$ (with $p \geq 1$) can be applied; applying it means that c spikes are consumed, thus only $k-c$ remain in the neuron σ_i; the neuron σ_i is fired and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately, otherwise the spikes are emitted after d steps. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t + 1, t + 2,\ldots, t + d-1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In the step $t + d$, the neuron spikes and becomes again open, hence it can receive spikes. The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one then no spike is emitted and the neuron cannot be closed.

In each time unit, in each neuron which can use a rule, a rule must be used, nondeterministically chosen if several rules are applicable. Note that each neuron processes sequentially its spikes, using only one rule in each time unit. During the computation, a configuration is described by both the number of spikes present in each neuron and by the number of steps to count down until it becomes open (this number is zero if the neuron is open, for example, in the initial configuration). Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where all neurons are open and no rule can be used.

Let Π be an SN P System and let $\mathcal{C} = C_0 \Rightarrow C_1 \Rightarrow C_2 \Rightarrow \cdots$ be a computation in Π. The *spike train* of computation $\mathcal{C}$, denoted by $st(\mathcal{C})$, is the sequence of emitting steps, and we write it in the form $st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle$, with $1 \leq t_1 < t_2 < \cdots$. The set of all spike trains (over the set $COM(\Pi)$ of all computations) of Π is denoted by $ST(\Pi)$. We also denote by $N(\mathcal{C}) = \{n \mid n = t_i - t_{i-1},\ \text{for } 2 \leq i \leq k, st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle\}$.

One can associate a set of numbers with $ST(\Pi)$ in several ways. Then, like in Ionescu et al. (2006), we can consider the intervals between consecutive spikes as numbers computed by a computation, with several alternatives:

- Taking into account only the first $k \geq 2$ spikes:

$$N_k(\Pi) = \{n \mid n = t_i - t_{i-1},\ \text{for } 2 \leq i \leq k, \mathcal{C} \in COM(\Pi),\ st(\mathcal{C}) = \langle t_1, t_2, \ldots \rangle,\ \text{and } \mathcal{C} \text{ has at least } k \text{ spikes}\}.$$

- Taking into account all spikes of computations with infinite spike trains:

$$N_\omega(\Pi) = \{n \mid n = t_i - t_{i-1},\ \text{for } i \geq 2, \mathcal{C} \in COM(\Pi)\ \text{with } st(\gamma) = \langle t_1, t_2, \ldots \rangle \text{ infinite}\}.$$

- Taking into account all intervals of all computations:

$$N_{all}(\Pi) = \bigcup_{k \geq 2} N_k(\Pi) \cup N_\omega(\Pi).$$

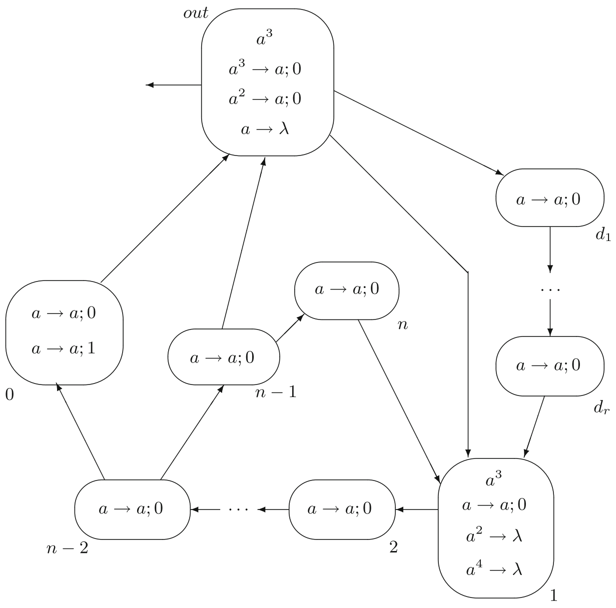

Fig. 1 An SN P system computing $\{r + ni \mid i \geq 1\}$, for $n \geq 3$

Definition 2 A system Π is said to be weakly ω-coherent if there is a computation $\mathcal{C}$ in Π such that $N(\mathcal{C}) = N_{all}(\Pi)$ (that is, there is a computation which provides all numbers which all other computations can provide).

The graphical representation of an SN P System is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses. Figure 1 shows an example of the graphical representation of an SN P System taken from Păun et al. (2006).

The transition diagram of an SN P System is also a directed graph where the nodes are configurations and an arc is drawn between two nodes/configurations if a direct transition is possible between them. This transition diagram is an interesting tool that allows obtaining invariant formulae to make easier the formal verification of a designed SN P System.

3 Software

Since Păun initiated Membrane Computing (Păun 2000) as a new branch of Natural Computing, a large number of variants have been considered, both concerning the syntax and the semantics of the model.

Let us recall here three of the main variants of P systems: Cell-like P systems, Tissue-like P Systems and Spiking Neural P Systems.

In the *cell-like* model of P systems, membranes are hierarchically arranged in a tree-like structure (see Păun 2000). The biological inspiration for it is the morphology of cell, where small vesicles are surrounded by larger ones. This biological structure can be abstracted into a tree-like graph, where the root of the graph represents the skin of the cell (the outermost membrane), the leaves represent membranes that do not contain other ones (elementary membranes) and two nodes in the graph are connected if they represent two membranes such that one of them contains the other one. In the *tissue P systems* (Martín

Vide et al. 2002, 2003) we consider a general graph instead of a tree-like membrane structure, where the nodes can be considered as processors and the edges as connections among them in order to share information. This model has two biological inspirations: intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a network of processors dealing with symbols and communicating these symbols along channels specified in advance.

Irrespectively of the selected approach, it is usually a complex task to predict or to guess how a P system will behave when we are designing a cellular solution to a problem. Moreover, as there do not exist, up to now, implementations in laboratories (neither in vitro nor in vivo nor in any electronical medium), it seems natural to look for software tools that can be used as assistants that are able to simulate computations of P systems.

3.1 Design

Generically speaking, the design and development processes for a P system simulator can be structured as follows (see Gutiérrez-Naranjo et al. 2006 for details):

– *Formal definition of the model*. First of all, one has to choose which variant of membrane systems is going to be simulated, stating precisely the syntax and semantics of the model to avoid ambiguous interpretations.
– *Choice of a programming language*. Each programming language has its own advantages and disadvantages and, up to now, there is no objective criterion to decide which is the more suitable one for simulating the evolution of a membrane system. Indeed, a large number of different languages as Haskell, Prolog, Java, C, Lisp, Scheme, Visual C++, CLIPS, etc. have been chosen by authors in the literature.
– *A good way to represent the knowledge*. The choice of a suitable data structure is a key problem in all fields of Computer Science (in particular, when dealing with the simulation of P systems). This decision is of course related with the programming language that is used, as specific techniques related with it have to be applied. A good representation allows a quick transition between configurations and therefore speeds up the simulation.
– *Design of an inference engine to carry out the computation*. There exists two basic difficulties intrinsic to the simulation of a P system in a conventional computer: their massive parallelism and the non determinism. Both features must be beared in mind for designing simulators because sequential conventional computers have only one processor.

3.2 Existing Software

The first simulators appeared in 2000, only 2 years after Păun's foundational paper (Păun 2000) was presented. It was written by Maliţa (2000) in LPA-Prolog and, since then, many other software simulators have been presented (see Gutiérrez-Naranjo et al. 2006 and references therein for a presentation of the first generation of simulators). The common purpose of this first generation was the better understanding of the computational process of P systems, for pedagogical purposes as well as assistant for researchers. Among these simulators, we can find simulators for P systems where the membrane structure does no change along the computation (as Maliţa's one) or simulators able to deal with P Systems where the membranes can divide (as Ciobanu and Paraschiv's simulator (2002)). Some of them explore new architectures looking for increasing the speed of the computation, as Ciobanu and Wenyuan's simulator (2004) based on parallel architecture. Others put the stress on a simulation closer to biological laws, as the simulator by the *Group for Models of Natural Computing* (http://www.di.univr.it) in Verona, based on the implementation of the metabolic algorithm introduced in Bianco et al. (2006).

After the first wave, a new generation of software simulators has arisen. They have left the pedagogical purposes and focus their goals on realistic simulations of physical processes via membrane computing techniques. Among them, we can find Nishida's simulator for the Membrane Approximation Algorithm, the two simulators from D. Pescini and P. Cazzaniga related to Vibrio Fischeri and Dynamical Probabilistic P Systems, *Cyto-Sim* (Sedwards and Mazza 2007) a processes simulator designed at the Microsoft Research—University of Trento Centre for Computational and Systems Biology, the stochastic simulator based on a generalisation of the Gillespie algorithm designed by Romero-Campero and Pérez-Jiménez (2008) and the simulators for biological processes from the University of Sheffield group and Seville team (all of them available from http://ppage.psystems.eu/).

4 A simulator for SN P Systems

In Ramírez-Martínez and Gutiérrez-Naranjo (2007), a simulator for SN P Systems was presented. It receives the description of an SN P System and outputs its transition diagram with an user friendly interface. Until now, no other software tool has been presented for dealing with SN P Systems. It shares the common purpose of the first simulators of other P system models: the understanding of the computational process of the systems and becoming an assistant for researchers.

As pointed out above, the graphical representation of an SN P System is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses. In the graphical representation of this software tool, the environment is also represented by a membrane and an arrow exits from the output neuron, pointing to this membrane; in each neuron we specify the rules and the spikes present in the initial configuration. Figure 2 shows a screen snapshot of the designer module of the software by showing the graphical representation of an SN P System.

4.1 Technical features

The software tool[2] for simulating the evolution of an SN P System is composed by the integration of three modules:

- A Graphical User Interface (GUI) which allows an easy interface to users, decoupling experimentation and simulation from programming. This module is written in XBase++ (http://www.alaska-software.com), which is an object oriented language to program database oriented graphic applications.
- A second module, written in SWI-Prolog (http://www.swi-prolog.org) which is the inference engine. It takes the initial configuration, the synapses and the rules and produces the transition diagram in text mode.
- A graphical designer tool, which allows the user to draw neurons, synapses and to associate spikes and rules with them by using a friendly interface.

[2] The simulator is available at (http://ppage.psystems.eu/).

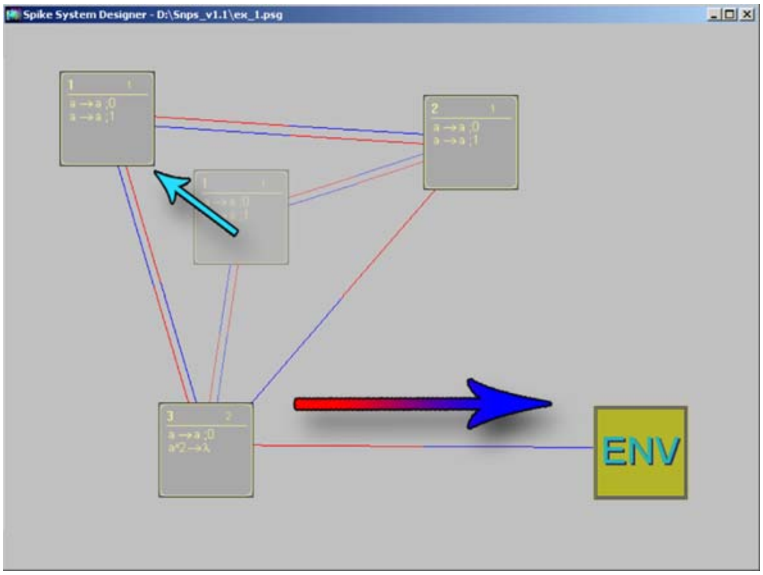

Fig. 2 Screen snapshot of the designer interface

Fig. 3 Input file

```
rule(1-1,a \to a;0).          rule(7-1,a \to a;0).
rule(1-2,a^2 \to \lambda).  rule(8-1,a \to a;0).
rule(1-3,a^4 \to \lambda).  rule(9-1,a^3 \to a;0).
rule(2-1,a \to a;0).          rule(9-2,a^2 \to a;0).
rule(3-1,a \to a;0).          rule(9-3,a \to \lambda).
rule(4-1,a \to a;0).          rule(10-1,a \to \lambda).
rule(5-1,a \to a;0).          rule(11-1,a \to a;0).
rule(6-1,a \to a;0).          rule(11-2,a \to a;1).

initial([3/0,0/0,0/0,0/0,0/0,0/0,0/0,0/0,3/0,0/0,0/0]).

synapses([1-2,2-3,3-4,3-11,4-5,4-9,5-1,
          6-7,7-8,8-1,9-1,9-6,9-10,11-9]).
```

The basic way of providing the data to the simulator is by a plain text file with the information stored in an appropriate way. Figure 3 shows the input file of the example represented[3] in Fig. 1.

The syntax is quite intuitive. The file consists on a set of literals with the predicate symbols `rule`, `synapses` and `initial`.

– Rules are of the form `rule(N − X, Latex)`, where `N` is the label of the neuron, `X` is the ordinal of the rule and `Latex` is the rule written in Latex mode. In the current version, we accept seven syntactically different types of rules:

(1) `a^s/a^c\to a^p; d` meaning $a^s/a^c \to a^p; d$
(2) `a^s/a^c\to a; d` meaning $a^s/a^c \to a; d$
(3) `a^c\to a^p; d` meaning $a^c \to a^p; d$
(4) `a^c\to \lamda` meaning $a^c \to \lambda$

[3] for the values $r = 3$, $n = 5$ and the labels $d_1 = 6$, $d_2 = 7$, $d_3 = 8$, $out = 7$, $env = 10$ and $0 = 11$.

(5) `a\to a; d` meaning $a \to a; d$
(6) `a^c\to a; d` meaning $a^c \to a; d$
(7) `a\to \lamda` meaning $a \to \lambda$

– `synapses` has as argument a list of pairs of neurons `Start – End`.
– `initial` stores the initial configuration. A configuration is a list of pairs `A/B` where the i-th pair `A/B` denotes the number of spikes (`A`) and the number of steps to became open (`B`) the neuron with label i.

The simulator also provides a graphical interface which allows the user to design an SN P System, to store it and build its transition diagram. The SN P System can also be exported from the graphical representation to a file with a text mode representation as described above.

5 An example

Next, we illustrate how important it is to know the transition diagram associated with an SN P System by means of an example.

Let us consider the SN P System, Π , described by Fig. 1. For each configuration C we denote by C^δ (with $\delta = 0{,}1$) the configuration obtained from C applying in neuron 0 the rule $a \to a$; δ. Then, the transition diagram of Π associated with the SN P System Π can be depicted by Fig. 4.

Firstly, we note that there are no halting computations in Π. We denote by σ the path $C_2 \to \cdots \to C_{r+2}$ and by τ the path $C_{r+2} \to \cdots C_{r+n-1} \to C^0_{r+n} \to C^0_{r+n+1}$. Then, Length($\sigma$) = r and Length (τ) = $n-1$.

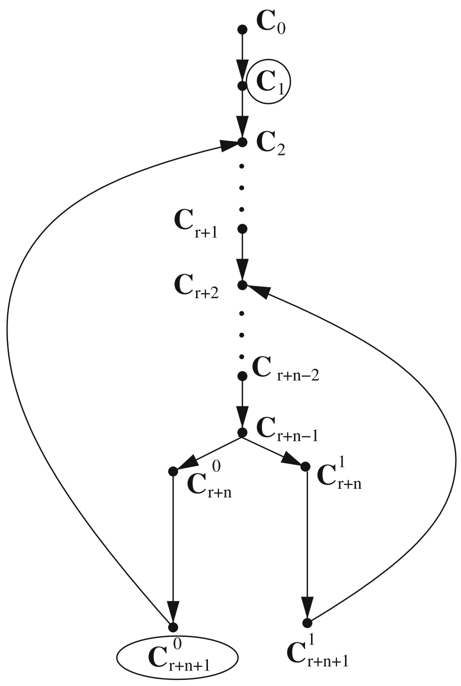

Fig. 4 Transition diagram of the SN P system in Fig. 1

For each $i \geq 0$, we denote by $\gamma(i)$ the path $C_{r+2} \xrightarrow{(i)} \cdots \rightarrow C_{r+n-1} \rightarrow C^1_{r+n} \rightarrow C^1_{r+n+1} \rightarrow C^1_{r+2}$, meaning that the configuration C_{r+2} is passed exactly $(i + 1)$ times. Let $\delta(i)$ be the path $C_2 \overset{\sigma}{\rightsquigarrow} C_{r+2} \overset{\gamma(i)}{\rightsquigarrow} C_{r+2}$. Then, Length $(\gamma(i)) = ni$ and Length$(\delta(i)) = r + ni$.

Hence, for every computation, $\mathcal{C}$, of Π there exists an infinite sequence of natural numbers $\{i_k\colon k \geq 1\}$ such that $\mathcal{C}$ can be described through the following path in the transition diagram associated with Π:

$$C_0 \rightarrow C_1 \rightarrow C_2 \overset{\delta(i_1)}{\rightsquigarrow} C_{r+2} \overset{\tau}{\rightsquigarrow} C^0_{r+n+1} \rightarrow C_2 \overset{\delta(i_2)}{\rightsquigarrow} C_{r+2} \overset{\tau}{\rightsquigarrow} C^0_{r+n+1} \rightarrow \cdots$$

We will denote that computation by $\mathcal{C}(\{i_k : k \geq 1\})$. Then,

$N_{all}(\mathcal{C}(\{i_k : k \geq 1\}) = \{t_{k+1} - t_k : k \geq 1\} = \{r + n(i_k + 1) : k \geq 1\}$, and we have the following result:

For every $n \geq 2$,

(a) For each computation, $\mathcal{C}$ of Π we have $N(\mathcal{C}) \subseteq \{r + ni : i \geq 1\}$.

(b) For each $i \geq 1$ there exists a computation, $\mathcal{C}$, of Π such that $r + ni \in N(\mathcal{C})$. Moreover, the SN P System Π is weakly ω-coherent.

This result illustrates the usefulness of the transition diagram in order to obtain the formal verification of an SN P System.

The simulator described in Sect. 4 automatically generates the transition diagram associated with a given SN P System.

Next, we briefly present how the simulator works on this example. After loading the corresponding file, which can be generated by using a text editor or the designer tool, the simulation can start. At time zero, only the initial configuration is shown.

After that, the user can develop the transition diagram step by step. The new elements (nodes or arcs) in each step are depicted in red color. Figure 5 shows an intermediate stage of the development of the transition diagram. When no new element can be added to the diagram, it is finished. In the example, this happens at step 11. Figure 6 shows the same transition diagram of the SN P System of Figure 1 for the values $r = 3$ and $n = 5$ with the representation of our simulator.

6 Final remarks and future work

The success of first generation of simulators in Membrane Computing is beyond any doubt. On the one hand, one of the main benefits of these simulators is their use for a better understanding of membrane computing, so they are pedagogical tools of first line. On the other hand, it has proved to be a useful assistant tool for the design and verification of complex P systems which solve hard problems, saving the researchers heavy hand-made calculations.

As pointed out in Gutiérrez-Naranjo et al. (2006), one of the common features of the first generation of simulators for cell-like P systems was the lack of efficiency in favor of the expressivity. The simulator reported here follows the same line: a high grade of expressivity is kept, looking for a friendly interface with the user.

The formal verification of an SN P System designed for solving a given problem is usually a hard task. The transition diagram associated with an SN P System provides information about some invariant formulae on the evolution of such system. In this context, it would be interesting to have a tool that automatically generates the transition diagram from the SN P System. The simulator presented in this paper satisfies this requirement.

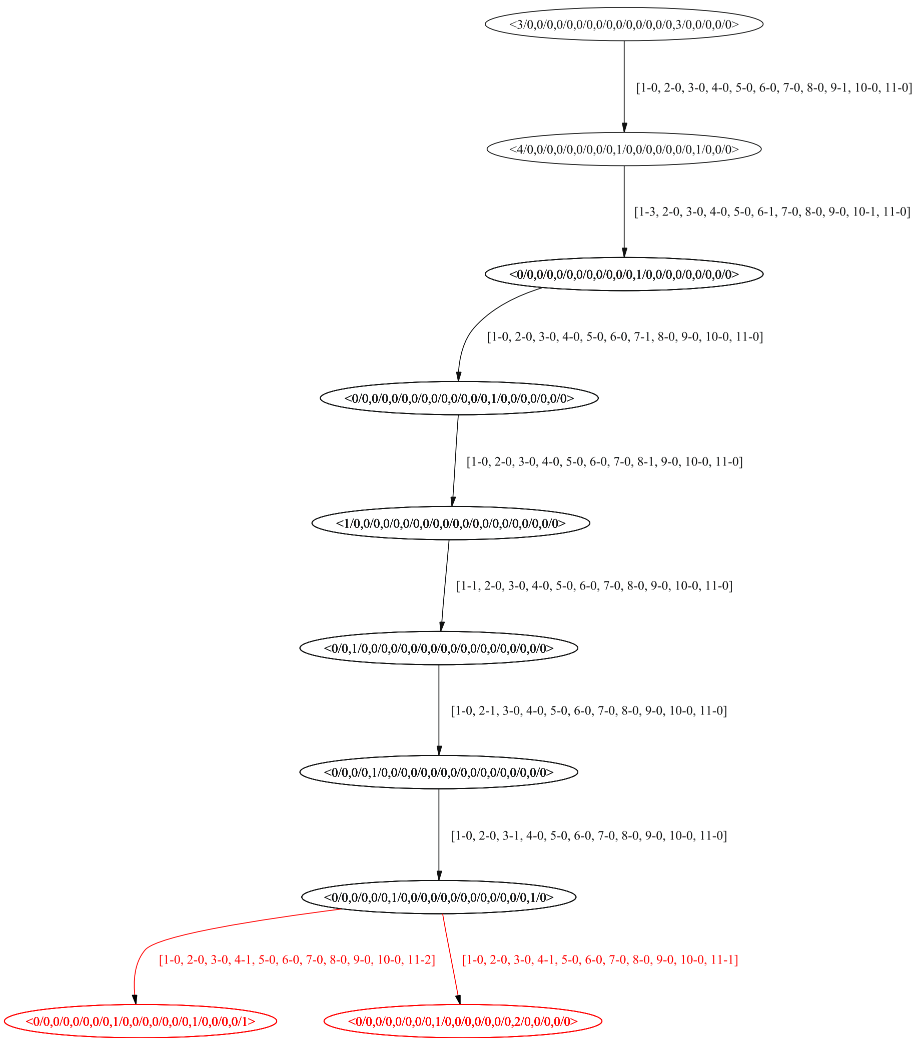

Fig. 5 Intermediate stage (Time 8)

Some technical details could be improved. As pointed out in Ramírez-Martínez and Gutiérrez-Naranjo (2007), one of them is the possibility of fixing a bound on the number of steps from the initial configuration in order to avoid that the software collapses. Another point is to extend the definition of the rules, since in the current version only a restricted definition is used.

It is desirable that the simulator is able to interact with the user by providing detailed information about the computation as, for example, the number of used rules in each step, intermediate configurations or objects sent to the environment (if any), in order to make statistical studies of the computations (see e.g. Cavaliere and Ardelean 2006; Gutiérrez-Naranjo et al. 2005; Suzuki et al. 2001). Indeed, biologically inspired variants of membrane systems are not interested in looking for halting configurations, but on the evolution process itself.

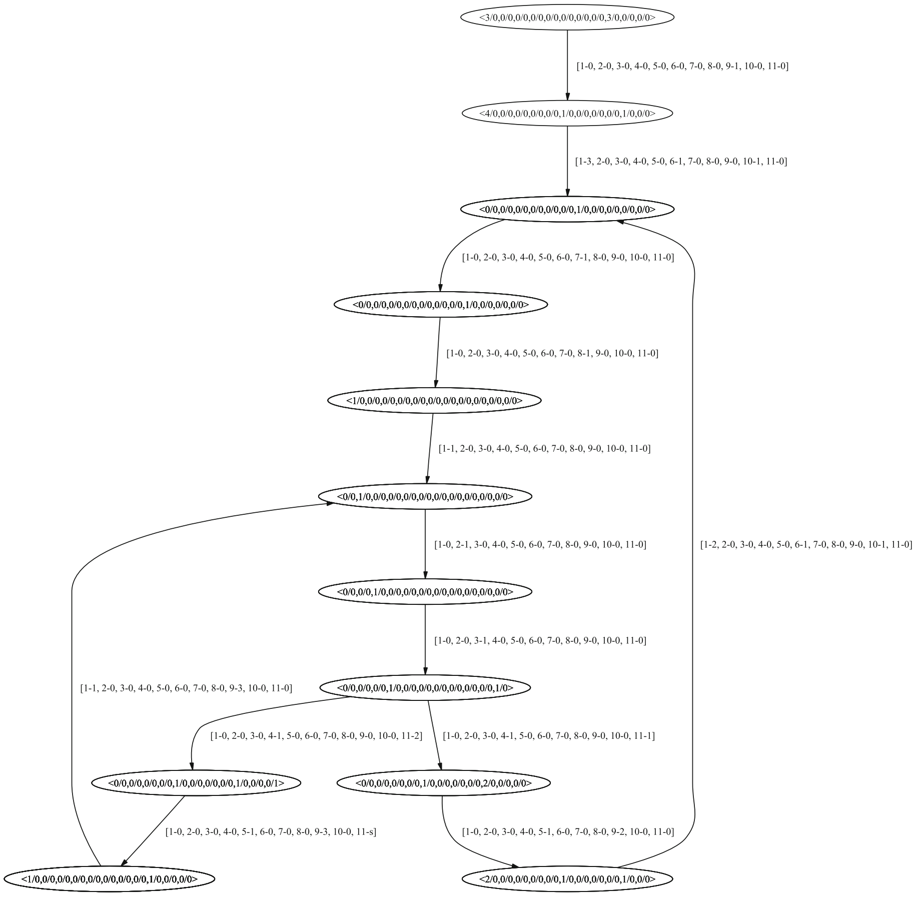

Fig. 6 The transition diagram generated by the simulator

Acknowledgements The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

Adleman LM (1994) Molecular computations of solutions to combinatorial problems. Science 226: 1021–1024

Adrian ED (1926a) The impulses produced by sensory nerve endings. J Physiol 61:49–72

Adrian ED (1926b) The basis of Sensation. WW Norton, New York

Bianco L, Fontana F, Franco G, Manca V (2006) P systems for biological dynamics. In Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) Applications of membrane computing. Natural Computing Series, Springer, pp 83–128

Cavaliere M, Ardelean II (2006) Modelling respiration in bacteria and respiration/photosynthesis interaction in cyanobacteria by using a P system simulator. In Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) Applications of membrane computing. Natural Computing Series, Springer, pp 129–158

Ciobanu G, Paraschiv D (2002) P system software simulator. Fundam Inform 49(1–3):61–66

Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) (2006) Applications of membrane computing. Natural Computing Series, Springer

Ciobanu G, Wenyuan G (2004) P systems running on a cluster of computers. In: Martín-Vide C, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing WMC 2003. Lect Notes Comput Sci 2933:123–139

Eckhorn R, Bauer R, Jordan W, Brosch M, Kruse W, Munk M, Reitboeck HJ (1988) Coherent oscillations: a mechanism of feature linking in the visual cortex? Biol Cybern 60:121–130

Gray CM, Singer W (1989) Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. Proc Natl Acad Sci USA 86:1698–1702

Group for Models of Natural Computing in Verona http://www.di.univr.it

Gutiérrez-Naranjo MA, Pérez-Jiménez MJ, Riscos-Núñez A (2005). On descriptive complexity of P systems. In: Mauri G, Paun Gh, Pérez-Jiménez MJ, Rozenberg G, Salomaa A (eds) Membrane computing. Fifth International Workshop, WMC5 2004. Lect Notes Comput Sci 3365:320–330

Gutiérrez-Naranjo MA, Pérez-Jiménez MJ, Riscos-Núñez A (2006) Available membrane computing software. In Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) Applications of membrane computing. Natural Computing Series, Springer, pp 411–439

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI

Hubel DH, Wiesel TN (1959) Receptive fields of single neurons in the cat's striate cortex. J Phisiol 148:574–591

Ionescu M, Păun Gh, Yokomori T (2006) Spiking Neural P Systems. Fundam Inform 71(2–3):279–308

Maliţa M (2000) Membrane Computing in Prolog, In: Calude CS, Dinneen MJ, Păun Gh (eds) Pre-proceedings of the Workshop on Multiset Processing. Curtea de Arges, Romania, CDMTCS TR 140, Univ of Auckland, pp 159–175

Martín Vide C, Pazos J, Păun Gh, Rodríguez Patón A (2002) A new class of symbolic abstract neural nets: tissue P systems. Lect Notes Comput Sci 2387:290–299

Martín Vide C, Pazos J, Păun Gh, Rodríguez Patón A (2003) Tissue P systems. Theor Comput Sci 296:295–326

McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5:115–133

Mountcastle VB (1957) Modality and topographic properties of single neurons of cat's somatosensory cortex. J Neurophysiol 20:408–434

P systems web page http://ppage.psystems.eu/

Păun Gh (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143

Păun Gh (2002) Membrane computing. An introduction. Springer-Verlag, Berlin

Păun Gh, Pérez-Jiménez MJ, Rozenberg Gr (2008) Spike trains in Spiking Neural P Systems. Int J Found Comput Sci 17(4):975–1002

Ramírez-Martínez D, Gutiérrez-Naranjo MA (2007) A software tool for dealing with Spiking Neural P Systems. In: Gutiérrez-Naranjo MA, Paun Gh, Romero-Jiménez A, Riscos–Núñez A (eds) Fifth brainstorming week on membrane computing. Fénix Editora, Sevilla, Spain, pp 299–313

Romero-Campero FJ, Pérez-Jiménez MJ (2008) A model of the quorum sensing system in vibrio fischeri using P systems. Artif Life 14(1):95–109

Sedwards S, Mazza T (2007) Cyto-sim: a formal language model and stochastic simulator of membrane-enclosed biochemical processes. Bioinformatics 23(20):2800–2802

Suzuki Y, Fujiwara Y, Tanaka H, Takabayashi J (2001) Artificial life applications of a class of P systems: abstract rewriting systems on multisets. In Calude CS, Paun Gh, Rozenberg G, Salomaa A (eds) Multiset processing. Mathematical, computer science, and molecular computing points of view. Lect Notes Comput Sci 2235:299–346

The Alaska Software web page http://www.alaska-software.com

The SWI-Prolog web page http://www.swi-prolog.org