A Software Tool for Dealing with Spiking Neural P Systems

Daniel Ramírez-Martínez, Miguel A. Gutiérrez-Naranjo

Research Group on Natural Computing
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
thebluebishop@gmail.com, magutier@us.es

Summary. Software simulators for P system are nowadays the main tool to carry out experiments in the field of Membrane Computing. Although the simulation of a P system is a quite complex task, current simulators have been successfully used for pedagogical purposes and also as assistant tools for researchers. In this paper we present a first software tool for dealing with Spiking Neural P Systems. This tool outputs the transition diagram of a given system in a step-by-step mode. The code is modular and flexible enough to be adapted for further research tasks.

1 Introduction

Membrane Computing was introduced by Gh. Păun in [11], starting from 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 and subvariants have been considered, concerning both the syntax and the semantics of the model. The devices of this model are called generically *P systems*.

Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules. The evolution of the rules is usually performed in a synchronous non-deterministic maximally parallel manner, but this can vary depending on 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 [11]). The biological inspiration 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

[1] The foundations of Membrane Computing can be found at [12] and updated bibliography at [16].

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* ([9, 10]) 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 (see [10]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules: Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite direction.

Spiking neural P systems (SN P systems, for short) were introduced in [6]. The aim of this new model was to incorporate in membrane computing ideas specific to spiking neurons; the intuitive goal was to have (1) a tissue-like P system with (2) only one (type of) object(s) in the cells, the spike, with (3) specific rules for evolving populations of spikes, and (4) making use of the time as a support of information[2].

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

The first software simulator for P systems appeared in 2000. It was written by Mihaela Maliţa [8] in LPA-Prolog and, since then, more than ten software simulators have been presented (see [7] and references therein). Their common purpose is 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 transition P system (as Maliţa's one) or simulators able to deal with P Systems with active membranes (as Ciobanu and Paraschiv's simulator [3]). Some of them explore new architectures looking for increasing the speed of the computation like Ciobanu and Wenyuan's simulator [5] 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* [17] in Verona, based on the implementation of the metabolic algorithm introduced in [1].

In this paper we present a new tool which is born with the hope of becoming a tool for the creativity in Membrane Computing. It deals with SN P systems, and to the best of our knowledge, so far no other software tool has been presented for dealing with such systems. Our simulator shares the common purpose of the

[2] In the next section we will give a brief introduction of some concepts related to SN P Systems, a detailed description can be found at [13] and the references therein.

others simulators: the understanding of the computational process of P systems and becoming an assistant for researchers. It receives the description of an SN P System and outputs its transition diagram with an user friendly interface.

The paper is organized as follows. First we recall some definitions related to SN P systems. Then, the simulator is presented and some features of its implementation are given. Section 4 provides an example of how the simulator works and finally, in the last section some remarks and future work lines are presented.

2 The Model

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 (called *synapses*). 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. The 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 the 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.

Formally, 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 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;
  - R_i is a finite set of rules of the form:

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

    where E is a regular expression with a the only symbol used, $c \geq 1$ and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$ too.
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, ..., m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots, m\}$ (synapses);
- $out \in \{1, 2, \ldots, m\}$ is the output neuron. Notice that only one neuron can send spikes to the environment.

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 $a^c \rightarrow \lambda$.

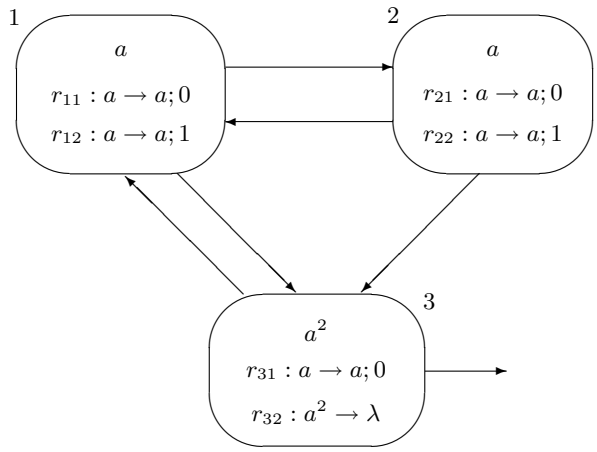

Fig. 1. An example of SN P system

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 $E/a^c \to a^p; d \in R_i$ (with $p \geq 1$) can be applied; applying it means that c spikes are consumed, only $k - c$ remain in the neuron 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 step $t + d$, the neuron spikes and becomes again open, hence 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 non-deterministically 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.

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 [2].

In the graphical representation of our software tool, the environment is also represented by a membrane and an arrow exits from the output neuron, pointing

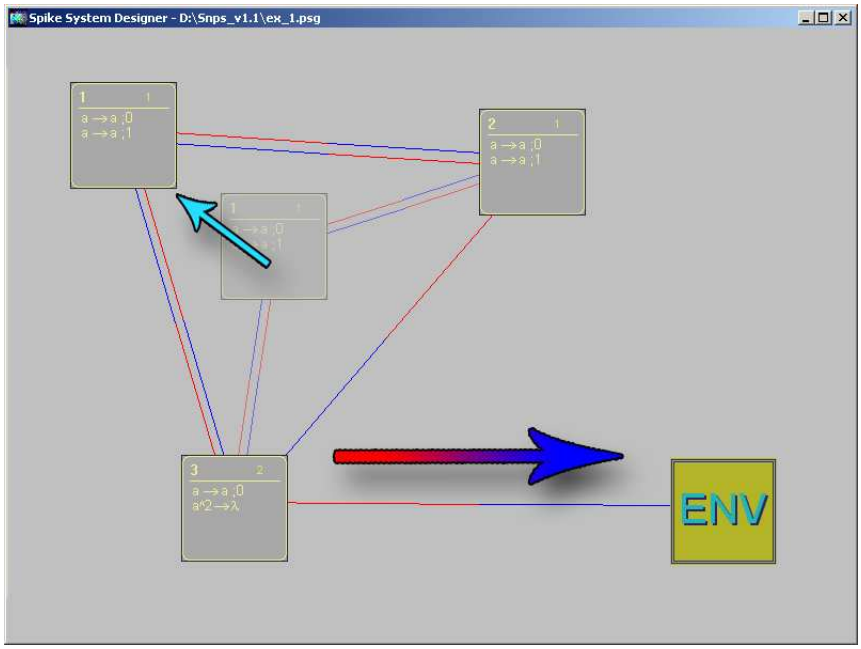

Fig. 2. Screen capture of the designer interface

to this membrane; in each neuron we specify the rules and the spikes present in the initial configuration. Figure 2 shows a screen capture of the designer module of our software by showing our graphical representation of the example.

The transition diagram of an SN P System is also a graph where the nodes are configurations and an arrow is drawn between two nodes/configurations. We draw an arrow if a direct transition is possible between them. Our software also provides labels to the arcs with information about the rules used in the transition.

3 The Simulator

The software tool[3] 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++ [15], which is an object oriented language to program database oriented graphic applications.

[3] The simulator is available at [16].

```
rule(1-1,a \to a;0).          synapses([1-2,2-1,1-3,3-1,2-3,3-env]).
rule(1-2,a \to a;1).
rule(2-1,a \to a;0).          initial([1/0,1/0,2/0]).
rule(2-2,a \to a;1).
rule(3-1,a \to a;0).
rule(3-2,a^2 \to \lambda).
```

Fig. 3. Input file

- A second module, written in SWI-Prolog [14] 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.

Next we briefly describes some features of the software.

3.1 The input data

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 depicted in Figure 1.

The syntax is quite intuitive. The file consists of 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 just the rule written in Latex. The idea is that the researcher preparing a report on SN P systems can move the set of rules from the report to the simulator and backwards with the minimal changes. In the current version, we accept seven syntactically different types of rules:
  (1) `a^s/a^c \to a^p;d`
  (2) `a^s/a^c \to a;d`
  (3) `a^c \to a^p;d`
  (4) `a^c \to \lambda`
  (5) `a \to a;d`
  (6) `a^c \to a;d`
  (7) `a \to \lambda`
- `synapses` has as argument a list of pairs of neurons `Start-End`. Notice that the environment is considered as a special case of neuron without rules and labeled by `env`.
- `initial` stores the initial configuration. We always consider the neurons labeled with numbers $1, 2, 3, \ldots$ and we do not make explicit the label of the neuron in the configuration. In this way, 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)` of 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 be also be exported form the graphical representation to a file with a text mode representation as described above.

3.2 The inference engine

After the input data have been provided (via the graphical interface or a plain text file), the generation of the transition diagram starts. The basic data structure consists of three lists which will be called `ExpandedNodes`, `NonExpandedNodes` and `Arcs`. We consider that a `Node` of the transition graph, i.e., a *configuration* of the SN P system has been *expanded* when all the configurations that can be reached from the `Node` in one step have been computed.

At the beginning, `NonExpandedNodes` contains the initial configuration and `ExpandedNodes` and `Arcs` are empty lists. The main procedure is quite natural. We take a `Node` from `NonExpandedNodes` and compute all the configurations that can be reached from the `Node` in one step. We will denote by `NewNodes` the list of these configurations. The `Node` is added to `ExpandedNodes` and all the elements of `NewNodes` that do not belong to `ExpandedNodes` are added to `NonExpandedNodes`. Analogously, the labelled arcs with the information of the rules that produce the reachable configurations are also stored in `Arcs`. The process ends when the list `NonExpandedNodes` is empty.

In the current version, the unique test for finishing the process is to check if `NonExpandedNodes` is empty. This means that the simulator is not able to deal with large transition diagrams, since the memory of the computer can be filled before the process ends.

The list `NewNodes` consists of all the configurations that can be reached from the `Node` in one step. In order to find all these configurations, for each neuron we check the set of *applicable* rules of the neuron according to the set of spikes and if the neuron is closed or open. To check if a general rule $E/a^c \rightarrow a^p; d \in R_i$ is applicable involves to know if an expression of type a^k belongs to $L(E)$. In the current version we only consider a weak version of applicability that can be extended in future versions of the simulator.

When the set of applicable rules have been found for each neuron, we consider the Cartesian product of these sets in order to get all the possibilities of reaching a new configuration. Each possibility gives us an arc of the transition diagram with a label composed with the following items:

- If the `X`-th rule of the neuron `N` is applied, then `N-X` is added to the label.
- If the neuron `N` is closed and therefore, no rule is applied, then `N-s` is added to the label.
- Finally, if the neuron `N` is open, but no rule can be applied then we add `N-0` to the label.

Fig. 4. Main window of SNPS-GUI

After the set of arcs with its corresponding label is fixed, the application of the rules starts. We have a set of rules (one rule at most for each neuron) associated with each arc that produces a new configuration. In order to apply these rules and build the new configuration we have to consider:

- The spikes sent out from all the neurons.
- For each neuron, all the incoming spikes. Some of them can have been sent some steps before with delay and arrive in the current step.

We need to keep in memory the spikes sent with delay in the previous steps, so the internal representation in the software of a neuron is more complicated than the list of pairs *Spikes/Delay* of the standard representation of the configuration.

The set of all the configurations reachable are determined by the set of arcs. When all these configurations have been generated, the list `NewNodes` is finished and the main procedure goes on.

3.3 The GUI for the inference engine

The transition diagram of an SN P system is usually a large graph and it can be useful to have a tool that shows the evolution in time of the diagram. The GUI module provides a set of images (PS and JPEG files) representing the steps on the

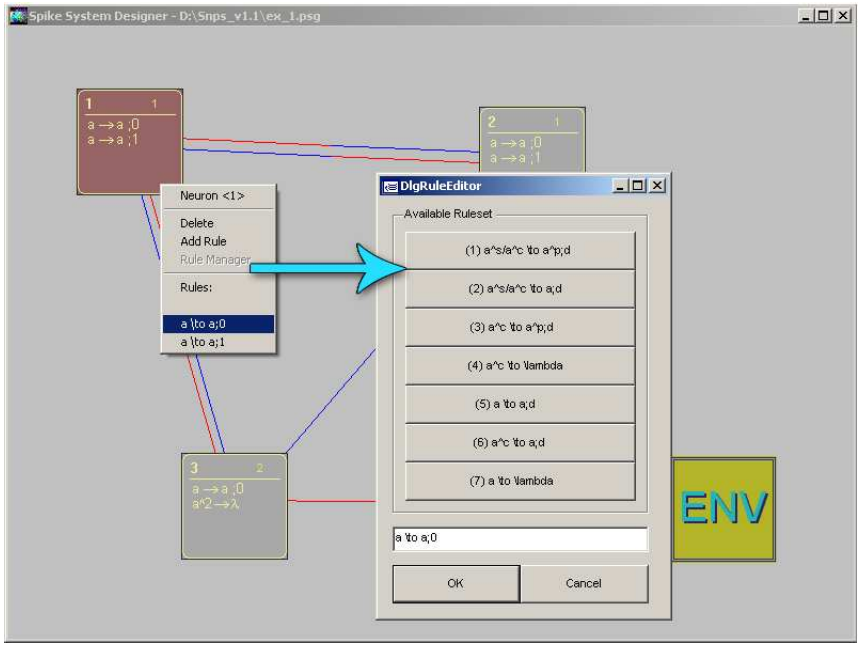

Fig. 5. Rule editor window

evolution of the system. At each step, the picture shows the set of reachable configurations till the moment together with the labeled arcs. This GUI also includes a system designer for the SN P systems.

Basics of SNPS-GUI

This software runs on Windows SO machines, so it has a typical windows 800x600 application look. To complete a simulation the user can follow the following steps[4]:

- Load an ASCII description of the SN P System which must be compatible with inference input description (see section 3.1.)
- Click the `RUN` button.
- Wait
- Navigate through preview images using the ARROW buttons.
- Catch output files, generated in the application directory, showing the evolution in a graphical way.

[4] Along this paper we provide a brief description of the use of the simulator. For a comprehensive list of features, see the file `Readme.txt` distributed with the software for complete information and updated features.

Controls and displays

- *Preview:* After running a simulation, the output PostScript files are converted into two JPEG series with different resolutions. The software shows the lowest resolution picture here to preview the evolution.
- *Navigation buttons:* Two arrows are provided in the preview window, so the user can see the step-by-step evolution of the graph.
- *Input content:* A simple text container where the user can check the input for the engine after having loaded it.
- *Command buttons:* With these buttons the user can browse for an input file in the O.S., run the simulation once the file has been loaded or call the *Graphical Designer* to paint neurons, synapses and to write rules.

The SN P System Graphical Designer

This internal module inside the GUI provides a graphic interface to generate SN P system description files. The user simply draw neurons, synapses and describe rules using a friendly interface. After that, the depicted graph can be exported into a file that can be loaded and modified into the GUI in future sessions.

As one can see in Figure 2, a neuron is just a colored square with some text inside. We have tried to keep notation as standard as possible according with the bibliography without adding unnecessary complexity to the code. As an effective and intuitive way of represent the information in the system we have inside of each neuron the following text:

- *Label (top-left corner of the neuron):* A labeling system based in the creation order is used. The current version does not allow the user to change the label of the neuron. The first created neuron is labeled as 1, second as 2 and so on. If during the creation process a neuron is deleted, the remaining ones are relabeled to avoid holes in the label list.
- *Initial configuration (top-right of the neuron):* Number of spikes of the neuron in the initial configuration.
- *Set of rules:* The `X`-th rule of the of the neuron `N` is denoted by `N-X`. Because of the space limitation, the application just show the first four rules inside each neuron this way. To see all the rules, the Neuron Menu must be activated.

The environment is depicted as a special neuron with a synapse from the output neuron. Obviously, no rules or synapses for this neuron can be defined. With this interface the user can create, destroy (delete) and connect neurons just by clicking neurons, environment and background surface.

Rule editor

A simple rule editor is also provided. It shows to the user the patterns of *compatible-engine* rules. By using it, the user can create, modify or delete rules related to the

selected neuron. Since there is no syntax corrector implemented in the rule editor at the current version, the user must check the rules before the system is exported. The rule editor gives patterns for the seven rules that the engine can recognize. Every a is intended to be a *spike* and the rest of constant (s, c, p and d) of the pattern (e.g. `a^s/a^c \to a^p;d`) *must be replaced by integers* in order to have an engine-compatible file.

4 An Example

In this section we present how the simulator works on a well known example. We have taken the SN P system of Figure 1 and show the use of our simulator in order to build its transition diagram. Figure 6, taken from [13], shows the transition diagram of that SN P system.

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 7 shows the different stages of the development of the transition diagram. When no new element can be added to the diagram, it is finished. Figure 8 shows the same transition diagram from Figure 6 with the representation of our simulator.

5 Final Remarks and Future Work

The success of the simulators in membrane computing is beyond any doubt. On the one hand, one of the main usefulness of these simulators lies in their use for a better understanding of membrane computing, so it is a pedagogical tool of first hand. On the other hand, the simulators have proved to be a useful assistant tool for the design and verification of complex P systems which solve problems, saving the researchers heavy hand-made calculations.

To the best of our knowledge, the simulators presented in the literature corresponds to cell-like models of P systems. In this paper we present a first software tool for dealing with SN P systems with the hope that it will become a useful tool in the same way like the cell-like ones, both for pedagogical purposes as well as an assistant for researchers.

Since membrane computing is a vivid area and new variants of P systems based on neurons and spikes can appear, we have designed a software tool with a code flexible enough to adapt the simulator to forthcoming ideas. Indeed, one of its main features is its modularity and the ability of embedding new P system models in future releases with a minimal modification of the code.

As pointed in [7], 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.

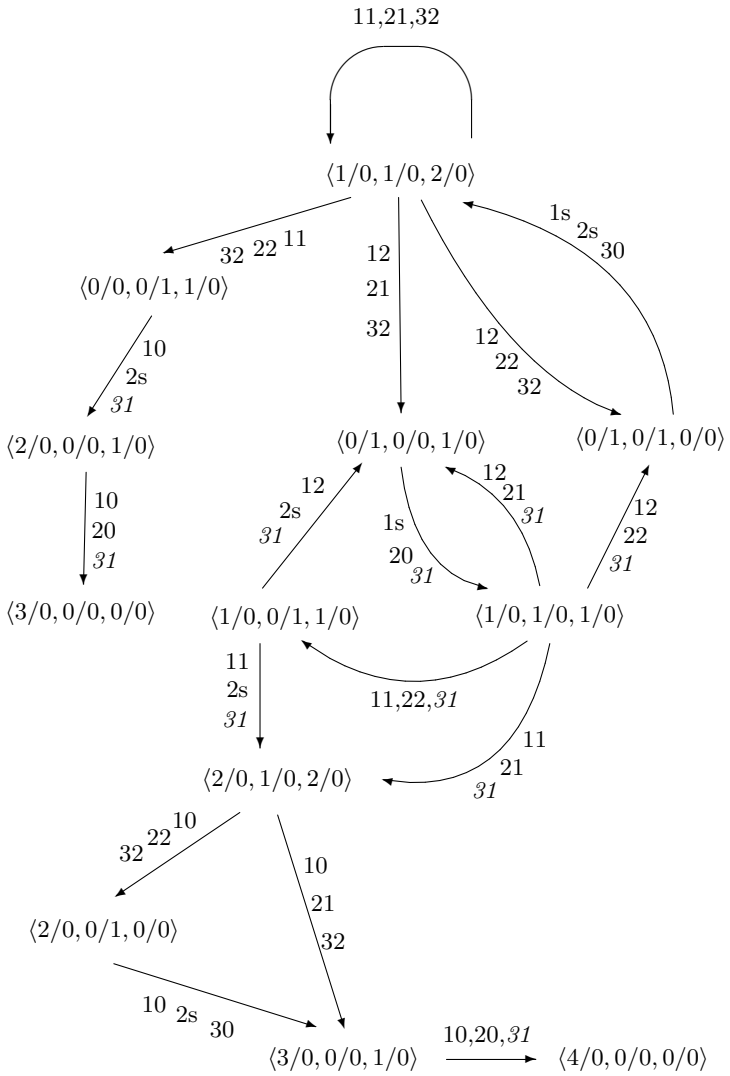

Fig. 6. The transition diagram from [13]

The current version of the simulator follows the same line: we keep a high degree of expressivity, looking for a friendly interface with the user, but some technical details can be improved. As pointed out above, one of them is the possibility of fixing a bound of the number of steps performed from the initial configuration in order to avoid that the software collapses. Anther point is to extend the definition of the applicability of rules, since in the current version only a restricted definition is used.

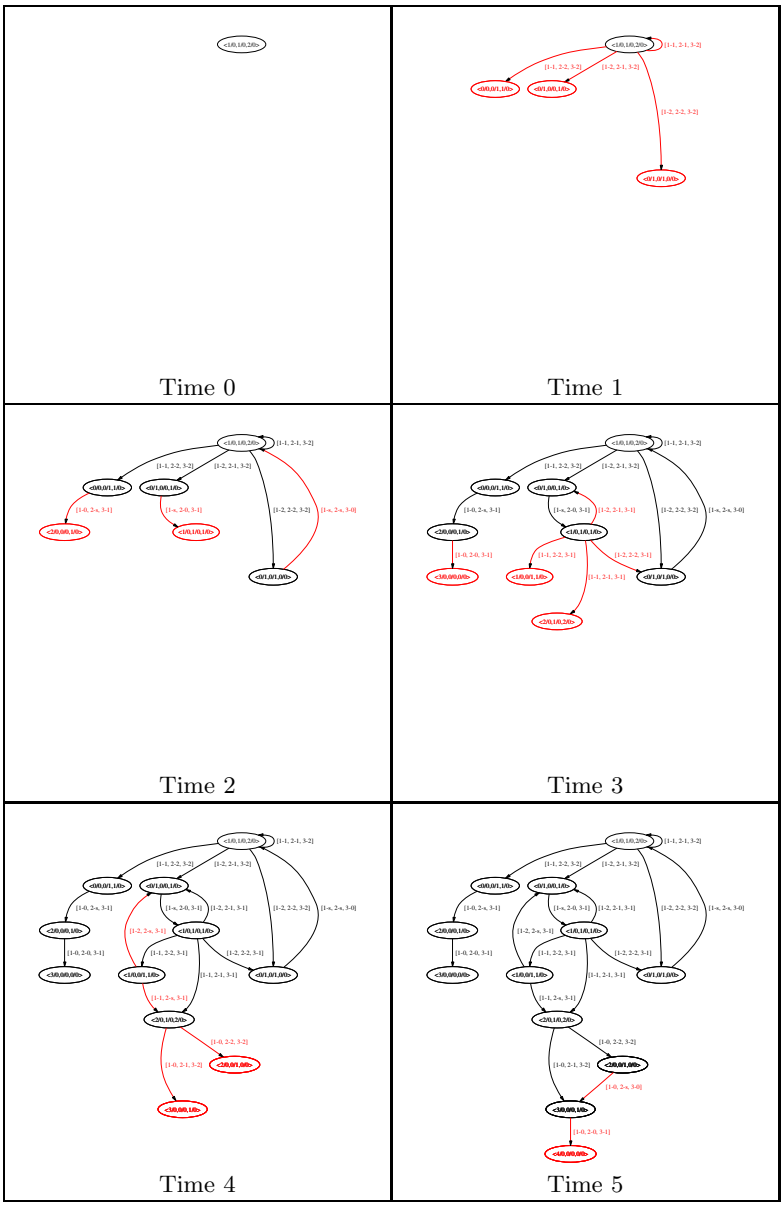

Fig. 7. Evolution of the example

We hope that this simulator will become a useful tool for the P systems community and our aim is to adapt it according to the new developments in the field. All suggestions are absolutely welcome.

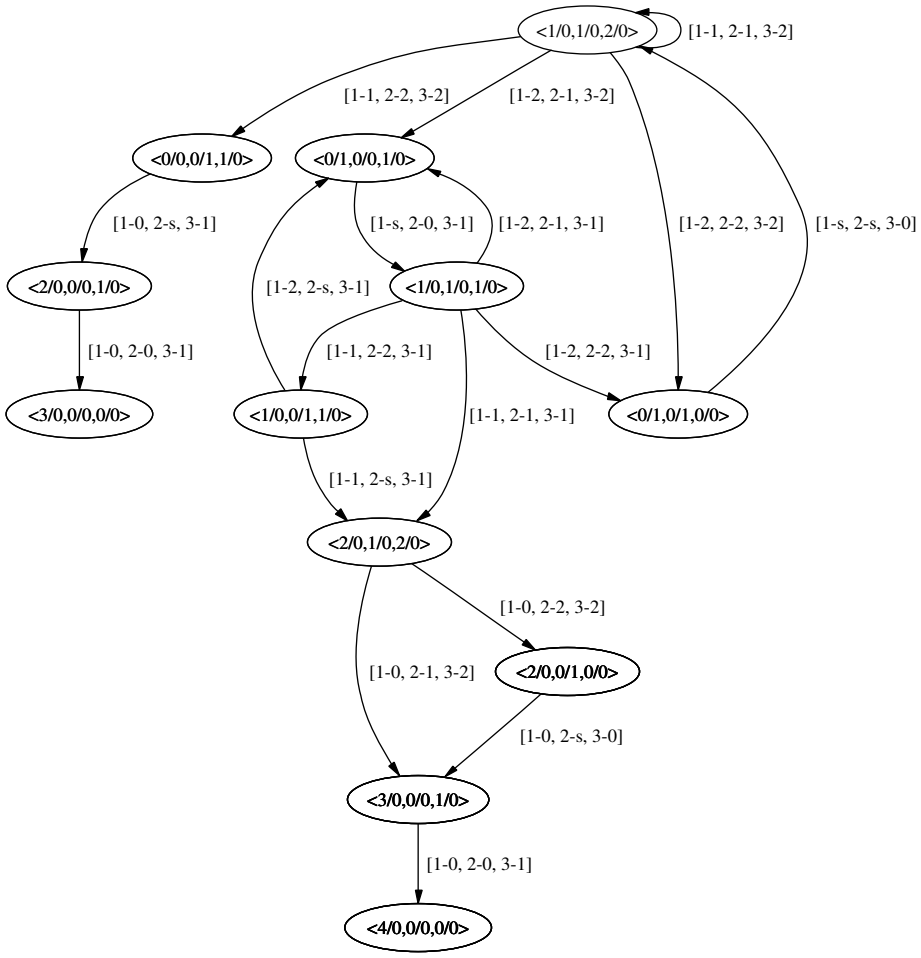

Fig. 8. The transition diagram from generated by the simulator

Acknowledgement

The second author acknowledges the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excelence TIC-581 of the Junta de Andalucía.

References

1. L. Bianco, F. Fontana, G. Franco, and V. Manca: P Systems for Biological Dynamics. In [4], 83–128.
2. H. Cheng, R. Freund, M. Ionescu, Gh. Păun, and M.J. Pérez-Jiménez: On String Languages Generated by Spiking Neural P Systems. In *Fourth Brainstorming Week on Membrane Computing*, Vol. I (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.) Fénix Editora, Sevilla, 2006, 169–193.
3. G. Ciobanu and D. Paraschiv: P System Software Simulator. *Fundamenta Informaticae*, 49, 1-3 (2002), 61–66.
4. G. Ciobanu, Gh. Păun, and M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*. Springer, 2006.
5. G. Ciobanu and G. Wenyuan: P Systems Running on a Cluster of Computers. In *Membrane Computing WMC 2003*. (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.) Lecture Notes in Computer Science, 2933, (2004), 123–139.
6. M. Ionescu, Gh. Păun, and T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
7. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, and A. Riscos-Núñez: Available Membrane Computing Software. In [4], 411–439.
8. M. Maliţa: Membrane Computing in Prolog, *Pre-proceedings of the Workshop on Multiset Processing* (C.S. Calude, M.J. Dinneen, Gh. Păun, eds.) Curtea de Argeş, Romania, CDMTCS TR 140, Univ. of Auckland, 2000, 159–175.
9. C. Martín Vide, J. Pazos, Gh. Păun, and A. Rodríguez Patón: A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* 2387, (2002), 290–299.
10. C. Martín Vide, J. Pazos, Gh. Păun, and A. Rodríguez Patón: Tissue P Systems. *Theoretical Computer Science*, 296, (2003), 295–326.
11. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1, (2000), 108–143.
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
13. Gh. Păun: Twenty Six Research Topics About Spiking Neural P Systems. In this volume.
14. The SWI-Prolog web page `http://www.swi-prolog.org`
15. The Alaska Software web page `http://www.alaska-software.com`
16. P systems web page `http://psystems.disco.unimib.it/`
17. Group for Models of Natural Computing in Verona `http://www.di.univr.it`