ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **20**, Number 1, 2016, 71–84

A Simulation Software Tool for Cell-like Spiking Neural P Systems

Luis Valencia-Cabrera[1], Tingfang Wu[2], Zhiqiang Zhang[2], Linqiang Pan[2,3*], Mario J. Pérez-Jiménez[1]

1. Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Spain
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
2. Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
3. School of Electric and Information Engineering
Zhengzhou University of Light Industry
Zhengzhou 450002, Henan, China
E-mail: lqpan@mail.hust.edu.cn

Abstract. Spiking neural P systems (SN P systems, for short) constitute a class of computing models in the research field of membrane computing. Inspired by the interactions among neurons in the brain, they have attracted much attention since their appearance in 2006. Many variants have emerged, presenting a graph-based structure, and several software simulators were developed for them. Recently, a different approach was proposed by introducing cell-like spiking neural P systems. Unlike previous SN P systems, this new model includes a tree-based structure, taking elements from traditional rewriting rules in the original P systems. In this work, a software tool within the framework of P-Lingua and MeCoSim is presented. This software may play an important role assisting in tasks of design, simulation and experimental validation.

Key-words: Bio-inspired computing, Membrane computing, Spiking neural P system, Cell-like P system, P-Lingua

1. Introduction

Membrane computing, a branch of natural computing taking inspiration from the structure and functioning of living cells, proved from the very beginning to be a very

*Corresponding author

active discipline in the search for proper computing models, with a bunch of variants of devices, P systems, proved to be Turing universal in most cases [20, 23, 27], as well as capable to solve hard problems in an efficient way [18, 22, 28].

Since the first P systems were introduced in [21], many types have emerged, falling into three main categories that could be placed in two groups: cell-like systems, presenting a hierarchical membrane structure inspired in eukaryotic cells internal structure; and tissue P systems and spiking neural P systems, both with a graph structure, the former ones inspired by the way in which cells organize and communicate in tissues [9, 19, 26, 30], and the latter ones inspired by the way in which neurons in the brain exchange information by means of the propagation of spikes [12, 13, 29].

A significant number of works have been published along the last decade concerning SN P systems [1, 14, 24, 33, 34], receiving much attention in the last few years. Variants of SN P systems have the common feature of the existence of the graph structure mentioned, unlike cell-like P systems. However, this clear separation between cell-like P systems with a hierarchical membrane structure, on the one hand, and SN P systems with a graph structure, on the other hand, has suffered a change with the proposal of a new computing model, the so-called cell-like spiking neural P systems (cSN P systems, for short), presented in [31,32]. Thus, this novel model takes elements from both cell-like P systems, given its hierarchical structure, and spiking neural P systems, preserving the existence of a singleton alphabet for spikes and the presence of spiking and forgetting rules, among other features.

As with several other kinds of P systems, when defining new computing models with their own syntactic features and semantic constraints, a number of studies could be conducted. These would range from their computational power and computational efficiency to the possible properties emerging from restricted variants of the proposed models or the solutions to well-known problems based on them. In this context, the possibility of having proper software tools to help in some of the related tasks, is more than interesting and definitely valuable. The variety of functionalities offered may vary depending on the needs detected and the availability of a software team to work in the development. Some of the main tasks where the software tools can aid in the definition of new models are the following: (1) design of solutions based on P system families to efficiently solve computationally hard problems, (2) formal verification of such solutions, (3) simulation of the computational devices, exploring the implications of the addition or removal of certain elements on the evolution of the corresponding systems.

In membrane computing, the idea of automating the evolution of P systems is an important research line. The way to get computational devices for a mechanical application of the rules in P systems is to develop software tools, which are able to run on computers and capable of simulating computations of P systems. A wide range of software simulators have been developed, such as Membrane Simulator for catalytic hierarchical cell systems and active membrane systems [7], SNUPS Simulator for numerical P systems [2], P-Lingua framework for offering a general syntactic framework that defines a unified standard for P systems [8,10], MeCoSim software for simulating biological phenomena by means of P systems [25, 36], SN P systems simulator for generating the transition diagram of an SN P system in an automatic way [11], and some hardware simulators based on highly parallel platforms GPU and CUDA [3, 4].

As P-Lingua framework became popular in the context of P systems, its use in

the community became more extended [16,17]. Moreover, some additional needs were detected, in terms of usability, and delivery of end-user applications, thus leading to the development of MeCoSim. The availability of this kind of pre-existing software tools seems to make advisable to avoid re-inventing the wheel when trying to address the development of similar assistants for new computing models. Hence, the development presented in this work within the framework of P-Lingua and MeCoSim was considered as the most appropriate approach. The work presented here have conducted to provide a software tool to allow computer-aided design and simulation of cell-like spiking neural P systems.

2. Cell-like Spiking Neural P Systems

This section presents some definitions providing the necessary background for the context where the developed tools emerge.

As introduced in [31], cell-like spiking neural P systems are mainly influenced by two kinds of systems: the cell-like P systems with rewriting rules and the classical SN P systems, introduced in [12].

2.1 P systems with multisets rewriting rules

A (cell-like) P system (with multiset rewriting rules) of degree $q \geq 1$ is a tuple of the form $\Pi = (O, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \ldots, \mathcal{R}_q, i_o)$, where O is a finite alphabet whose elements are called *objects*; μ is a rooted tree (the *membrane structure*) whose q nodes (called *membranes*) are injectively labelled by $1, \ldots, q$, respectively; $\mathcal{M}_i$, $1 \leq i \leq q$, is a multiset over O associated with membrane i at the beginning of a computation; $\mathcal{R}_i$, $1 \leq i \leq q$, is a finite set of evolution rules associated with membrane i, and $i_o \in \{0, 1, \ldots, q\}$ indicates the output region (a membrane in μ, or the environment labelled by 0). The rules of the system are of the form $u \to v$, where u is a multiset over O and v is a multiset over $O \times \{here, in, out\}$. Thus, each element of v is of the form (a, tar), where $a \in O$ and $tar \in \{here, out, in\}$. As usual, the target indication *here* can be omitted.

An extensive description of the semantics associated with this kind of systems can be found in [24], but we will briefly recall some relevant aspects. First of all, let us take a careful look at the target indication in. When a rule r associated with a membrane i is applied and pair (a, in) belongs to its right hand side, then object a will be sent to one of their children, non-deterministically chosen. Let us note that the non-deterministic choice is done for each object individually, even if many of them present that same target indicator. For example, if $(a, in)(a, in)$ belongs to the right-hand side of a rule r then two non-deterministic choices of children of membrane i should be considered in order to send "the first" object a and "the second" object a, respectively. As stated in [31], "one may also use the stronger indication in_j", so that the object is sent to the specific child membrane with label j, provided that it exists (otherwise the rule is not applicable).

In what follows, the semantics of these P systems is extended, in order to consider multisets of objects as units associated with the target indicators specified. In the new context, a rule (associated with membrane i) is of the form $u \to v$, where $u \in M(O)$ and v is a multiset over $M(O) \times \{here, in, out\}$, where $M(O)$ is the set of all multisets

over O. If such a rule is applied and pair (w, in), where w is a multiset over O, belongs to the right-hand side of the rule, then multiset w will be sent to one of their children, non-deterministically chosen.

Both in the original and the extended version described previously, the rules of the whole systems are applied in the maximally parallel manner: a maximal multiset of applicable rules is non-deterministically chosen and applied. Results are associated only with halting computations, encoded by the contents of the output region in the halting configuration.

2.2 Spiking neural P systems

Next, we briefly introduce spiking neural P systems in the extended form [6], but without delay, as recalled in [31]. For a detailed description of these systems please see [12].

An SN P system of degree $q \geq 1$ is a tuple $\Pi = (O, \sigma_1, \ldots, \sigma_q, syn, out)$, where $O = \{a\}$ is a singleton alphabet (a is called *spike*); $\sigma_1, \ldots, \sigma_q$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq q$, such that $n_i \geq 0$ is the *initial number of spikes* contained in the neuron, and R_i is a finite set of *rules* of the following two forms: (1) $E/a^c \to a^p$, where E is a regular expression over a and $c \geq p \geq 1$; and (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \to a^p$ of type (1) from R_i; $syn \subseteq \{1, 2, \ldots, q\} \times \{1, 2, \ldots, q\}$ with $(i, i) \notin syn$, for $1 \leq i \leq q$ (*synapses*); and $out \in \{1, 2, \ldots, q\}$ indicates the *output neuron*.

The *spiking rule* $r \equiv E/a^c \to a^p \in R_i$ is *enabled at a moment of time* t if neuron σ_i contains k spikes at that moment, $a^k \in L(E)$ and $k \geq c$. If rule r is applied then c spikes are consumed from neuron σ_i, and p spikes are sent to its outgoing neurons (each neuron σ_j such that $(i, j) \in syn$).

The *forgetting rule* $r \equiv a^s \to \lambda \in R_i$ is *enabled at a moment of time* t if neuron σ_i contains *exactly* s spikes at that moment. By applying rule r, all s spikes are removed from the neuron σ_i.

A *configuration* C_t at an instant t is a tuple $(p_1, \ldots, p_q)$, where $p_i \geq 0$ is the number of spikes contained in neuron σ_i at instant t. The initial configuration is $(n_1, n_2, \ldots, n_q)$. We say that configuration C_1 yields configuration C_2 in *one transition step* if we can pass from C_1 to C_2 by applying the rules from $R_1 \cup \cdots \cup R_q$ in such manner that inside each neuron at most one rule must be applied (in each neuron, if some rule is enabled then exactly one rule should be applied). A global clock is assumed, marking the time for the whole system.

In this way, starting from the initial configuration, a sequence of transitions among configurations take place. Such a (maximal) sequence of configurations is called a *computation*. A computation halts if it reaches a configuration where no rule is enabled. With any computation (halting or not), a *spike train* can be associated, consisting of a sequence of digits (0 and 1) indicating for each instant, if the output neuron sends a spike out of the system (1) or not (0).

The result of a computation can belong to one of the following types:

- The spike train itself, that is, a string over the alphabet $\{0, 1\}$.
- A natural number, representing the amount of steps between the first two spikes emitted to the output neuron by the system.

The set of such numbers is denoted by $N_2(\Pi)$. By convention: (a) number 0 is generated by a computation of Π which sends spikes out only once; and (b) if no computation of Π sends out any spikes then $N_2(\Pi) = \emptyset$.

- The number of spikes present in the output neuron in the halting configuration. The set of numbers generated in this way by Π is denoted by $N_{in}(\Pi)$.

Details about the universality of these systems considering the different types of possible results can be found in [5, 6, 12].

Next, cell-like spiking neural P systems (cSN P systems, for short) are presented, as appeared in [31].

2.3 Cell-like SN P Systems

A cSN P system of degree $q \geq 1$, is a tuple $\Pi = (O, \mu, n_1, \ldots, n_q, R_1, \ldots, R_q, i_o)$, where:

- $O = \{a\}$ is a singleton alphabet (a is called *spike*).
- μ is a hierarchical membrane structure with q membranes.
- $n_i, 1 \leq i \leq q$, is the number of spikes present in compartment i of μ at the beginning of the computation.
- $R_i, 1 \leq i \leq q$, is a finite set of rules from membrane i of the following form:
  (1) $E/a^c \rightarrow u$, where E is a regular expression over O, $c \geq 1$, and u is a multiset of pairs (a^p, tar), where $p \geq 1$ and $tar \in \{here, out, in, in_j, in_{all}\}$ (*spiking rules*).
  (2) $a^s \rightarrow \lambda$, where $s \geq 1$ (*forgetting rules*).
- $i_o \in \{1, \ldots, q\} \cup \{env\}$ indicates the output region (this is the environment if $i_o = env$).

In spiking rules, *tar* is a target indication specifying the destination of the associated spikes. The meaning of targets in, in_j and in_{all} is the following: by applying a rule associated with membrane i whose right-hand side contains the pair (a^p, in), (a^p, in_j) or (a^p, in_{all}), respectively, p spikes are sent to: (a) a children of membrane i, selected in a non-deterministic way; (b) the children j of membrane i; and (c) all children of membrane i. It is worth highlighting that the pairs (a^p, tar) are a particular case of the definition in previous section when the extension of cell-like P systems with rewriting rules was presented.

Computations in cSN P systems are defined as usual in SN P systems: in each membrane, at most one rule is applied, but the membranes work in parallel, synchronously.

The result of a computation can be of the following types:

- The number of spikes present in the output region in the halting configuration. If i_o is a membrane then we denote the set of numbers computed (generated) by the system Π as $N_{in}(\Pi)$ (the set computed in the *internal mode*).

- The time distance between the first two steps when the system sends spikes out. This can be done by rules associated with the skin membrane whose right-hand side contains pairs (a^p, out). The set of numbers computed by Π is denoted by $N_2(\Pi)$, with the same convention previously described.

As mentioned in [31], no restriction is imposed on the number of spikes produced, so that it can be greater than the number of consumed spikes. More details about this kind of systems and their universality are provided in [31].

3. A Unified Software Framework for P Systems

The appearance of P systems brought the emergence of new ideas, along with solid theoretical foundations. Soon after that, a few software tools started exploring the simulation of these systems, as *ad-hoc solutions* to specific problems.

As membrane computing grew, many types of P systems were defined, and the availability of general tools to work with these novel solutions became advisable.

P-Lingua meant a significant milestone, providing a uniform framework to specify, debug and simulate these computing devices. Its broad use in real applications led to the need of additional tools for P systems designers. Besides, the delivery of end-user applications based on this framework would widen the scope and visibility of the underlying systems. This kind of apps should abstract internal details to the end-users of the applications designed, as ecologists, economists, etc. The apps act as black boxes for them, once the proper files are provided by the P systems designers. These goals were achieved by MeCoSim [25].

This section outlines the main elements included in the framework provided by P-Lingua and MeCoSim.

3.1 P-Lingua Framework

P–Lingua framework was introduced in [8], including a general language to define P systems, *P–Lingua*, and a software library, *pLinguaCore*, supporting the specification and simulation of a variety of computing models within Membrane Computing. P-Lingua language *text files* can be easily processed by pLinguaCore, directly or through some client, both in console format or with the visual interface provided by MeCoSim, as described in Section 3.2. Not only specific P systems can be specified, but also families of them, with parameters accepting different values depending on the instances to generate.

3.2 MeCoSim (Membrane Computing Simulator)

MeCoSim offers P systems designers and end users visual tools to handle their solutions, either as white boxes to deepen in the study of P systems or as black boxes to focus on the problems, abstracting from internal details. It supplies model designers a graphic tool to design, simulate, analyse and verify their models. In addition, end users are provided with applications, with interfaces adapted for each problem, to enter the input data and check the results. MeCoSim is built on top of P-Lingua as specification language and simulation engine.

A number of features are available for debugging, visualization and customization. Besides, it provides a plugins architecture to extend its functionalities, and options related with invariants detection and connection with model checking software for the formal verification of the models (see [36]).

4. A Language to Define Cell-like SN P Systems

The previous section outlined the main elements of our framework to define, debug and simulate P systems. In the present work, a new type of P systems has been defined, with genuine features, so the specific language for them must be set.

We will introduce the syntax for cSN P systems in the following subsections..

4.1 Reserved Words and Special Elements

Despite the variety of systems available in P-Lingua, some well-known ingredients of P systems had not been incorporated so far. In particular, the general format for rewriting rules described in [24] including tagert indicators ($here$, in, out, in_{all}, in_j) was not covered. However, cSN P systems (see [31]) use these indicators. Consequently, new elements were needed in our framework.

In fact, before defining the language for cSN P systems, a more general model has been included in P–Lingua, for cell-like P systems including target indicators. They are a generalized version of the systems in [24], considering the extension described in Section 2. This model is now available within the framework, so that any P–Lingua file using these systems starts with the sentence:

```
@model<rewriting_systems>
```

We will omit further details about this new development in the rest of the paper, so that only aspects related with cSN P systems will be explained. Concerning the new target indicators, they are written as `here`, `out`, `in`, `inall` and `in{j}`.

4.2 Model Specification

Any P-Lingua file defining a **cSN P system** must set `cell_like_snp` as its model, thus beginning the file with the sentence:

```
@model<cell_like_snp>
```

The rest of the file will then define the main elements describing the cSN P system, typically consisting of $\Pi = (O, \mu, n_1, \ldots, n_q, R_1, \ldots, R_q, i_o)$. The singleton alphabet $O = \{a\}$ is fixed, so it does not need to be made explicit. The rest of the elements depend on the specific P system, so μ, $n_1, \ldots, n_q$, $R_1, \ldots, R_q$ and i_o will be set as explained in Subsections 4.3 to 4.7.

4.3 Membrane Structure

cSN P systems are a variant of spiking neural P systems based on a tree-structure. Therefore, to specify the initial membrane structure

`@mu` is used:

```
@mu = [ ... ]'1;
```

where ... stands for the definition of the membrane structure as usual.

4.4 Definition of Initial Multisets

When defining cSN P systems, we need to specify the multisets of objects initially placed in every membrane. Given a membrane i containing n_i spikes, the initial number of spikes it contains can be specified as follows:

```
@ms(i) = a*
```

For example, `@ms(2) = a*2` (if only one spike is present, `*1` is omitted). Alternatively, the initial objects can be included directly with `@mu`:

```
@mu = [ a*2 [ a ]'2 [ ]'3 [ a*5 ]'4 ]'1;
```

4.5 Definition of Rules

Two types of rules can be defined: forgetting rules and firing rules. The former ones can be defined in P-Lingua in the traditional way:

```
[a*c]'h --> [#]'h "e";
[a*c    -->  #]'h "e";
```

with h a label, c an integer expression and e a regular expression over $\{a\}$.

Regarding *spiking rules*, they must support the extended form $E/a^c \rightarrow u$, with E a regular expression over O, $c \geq 1$, as described in Section 2. They can be defined in the following ways:

```
[a*c]'h --> LIST_OF_PAIRS "e";
[a*c    --> LIST_OF_PAIRS]'h "e";
```

For any rule, e is optional. When e is not present in the rule, it defaults to the left hand side of the rule. The `LIST_OF_PAIRS`, as defined in Section 2, can present pairs of the form (a^p, tar). In P-Lingua, its syntax would be:

```
(a*p ; TAR)
```

with `TAR` expressed as described in Section 4.1 Concerning the interpretation of the rules, it is important to take into account the extension of P systems with rewriting rules presented in Section 2. Let us consider a rule in a region 1 such as: $a^2bc^3ba^2c \rightarrow (da, out)(ca, in)$.

This rule could be expressed in P-Lingua in the following ways:

```
[a*2,b,c*3]'1 --> (b,a*2,c;here) (d,a;out) (c,a;in);
[a*2,b,c*3]'1 --> b,a*2,c (d,a;out) (c,a;in);
[a*2,b,c*3 --> b,a*2,c (d,a;out) (c,a;in)]'1;
```

Thus, ca will be sent to the same region, non-deterministically chosen.

4.6 Regular Expressions

Regarding the syntax for regular expressions in P–Lingua, for *backwards compatibility* with other systems as the classical SN P systems, we adopted the same policies

implemented yet. As extensively explained in [15, 16], the mechanism to define and evaluate regular expressions is based on the `regex` Java package (details at [35]). As detailed in [15], the following subset of symbols can be used:

`'a', '(', ')', '[', ']', '{', '}', ',', '^', '*', '+','?', '|'`

4.7 Output Region

The last syntactic element present in the definition of a cSN P system is the output region. It can be specified in P-Lingua in a natural way as:

```
@mout = REGION;
```

This REGION should match the **label of a region** in μ, **or** refer to the environment (by setting the region to `environment`, or simply `env` or `e`).

4.8 General Features Derived from the Integration

The tools created for cSN P systems have been integrated in P–Lingua framework. Concerning the specification language, it will imply a clear advantage: the traditional mechanisms present in the general description of the language will be available. This includes features as modules/functions definition as in structured programming, use of variables, parameters, blocks or iterators, among others (see [8]).

In addition, through its integration in MeCoSim, these files may also make use of parameters generated from the input given by the user.

4.9 A New Simulator for Cell-like SN P Systems

A new simulator has been developed within P-Lingua framework to capture the dynamics of cSN P systems. The general idea is clear: once a P system is generated from a P–Lingua file, the simulator performs a possible computation from the initial configuration, producing the corresponding "non-deterministic" transitions until a halting configuration is reached. The simulation algorithm follows the general schema found in most of the simulators in the platform:

1. Initialization
2. For each computation step, while some rules are applicable:
   (a) Selection of rules
   (b) Execution of rules

The **initialization stage** will set the initial structures needed by the algorithm. Then, the main loop will run until a halting configuration is reached; that is, until no rule is applicable at a given computation step.

The **selection phase** will check which rules can be applied. Unlike other cell-like P systems simulators, cSN P systems cannot execute more than one rule at a given compartment in the same computation step, so the selection stage will choose, for each membrane in the system, at most one applicable rule, "non-deterministically" chosen.

The applicability of a rule is determined by the presence of at least the number of spikes in the left hand side of the rule. In addition, the contents of the membrane

where the rule is present must match the regular expression of the rule. Finally, if some target indicators of types in, in_{all} or in_j appear in the right hand side of a spiking rule, then the membrane must have child membranes, and in the latter case a child membrane labelled by j must exist.

As a result of this selection phase, a set of rules will have been selected, verifying that every membrane with applicable rules has selected exactly one.

Then, the **execution phase** applies the change in the configuration, passing from C_t to C_{t+1}, removing the objects consumed by the selected rules, and adding the objects produced by the rules to the corresponding target indicators, with the semantics specified in Section 2; that is, choosing non-deterministically the child membrane receiving the objects in the case of in, and with the corresponding deterministic result for the other target indicators.

4.10 Availability of the Software Tools Developed

The tools described in the previous sections, concerning the language, parsing and simulation have made publicly available in the current version of MeCoSim, that can be downloaded from [36]. Once downloaded, whenever the software runs, if an Internet connection is active, it checks the presence of new versions of any of the files involved, thus guaranteeing it always provides the user with the last version of MeCoSim (that includes in its installation pLinguaCore).

5. Case Studies

This section recalls some examples from [31], illustrating the behaviour of cSN P systems and showing the corresponding P–Lingua files.
The first system considered is given formally in [31] as:

$$\Pi = (\{a\}, [\ \ ]_1, 2, R_1, 1), \text{ where}$$

$$R_1 = \{(a^2)^+/a^2 \to (a^4, here),\ (a^2)^+/a^2 \to (a, here)\}.$$

The initial multiset in membrane 1, as specified in Π, is a^2. The behaviour of the system is as follows: the first rule adds two spikes, repeatedly, but when the second rule is used (this may also in any step) the number of spikes becomes odd, so no further rule can be used. Therefore, $N_{in}(\Pi) = \{2n+1 \mid n \geq 0\}$.

This system is introduced in the simulator developed as follows:

```
@model<cell_like_snp>
def main()
{
    @mu = [ a*2 ]'1;
    [a*2]'1  --> a*4 "(a{2})+";
    [a*2]'1  --> a "(a{2})+";
    @mout = 1;
}
```

As it can be seen, the initial spikes can be introduced in a compact way in the definition of `@mu`, and indicator `here` can be omitted.

Another example, also proposed in [31], involves the use of the target indication in_{all}, as indicated in Figure 1.

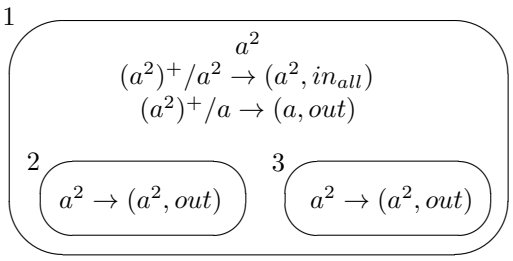

Figure 1: A cSN P system avoiding $c < p$.

In this example, the same set is obtained, but this time one spike is finally sent out to the environment. However, the output membrane is still membrane 1, as specified in the corresponding P-Lingua file.

```
@model<cell_like_snp>
def main()
{
    @mu = [a*2 []'2 []'3 ]'1;
    [a*2]'1  --> (a*2;inall) "(a{2})+";
    [a]'1  --> (a;out) "(a{2})+";
    [a*2]'2  --> (a*2;out);
    [a*2]'3  --> (a*2;out);
    @mout = 1;
}
```

Figure 2 shows this last example loaded in MeCoSim, including *step by step* information, P-Lingua file editor and multisets viewer.

6. Conclusions and Future Work

Over the last decades, membrane computing has kept very active in its search for models showing interesting properties from their computational power and complexity. Particularly, SN P systems have attracted much attention, with a significant amount of variants emerged, all of them with a graph-based structure.

However, recently conceived and published [31], cell-like spiking neural P systems have set a bridge between cell-like P systems, based on a hierarchical structure, and spiking neural P systems, with spiking and forgetting rules. They proved to be computationally universal when no constraints are set over the generation of more spikes than those consumed in spiking rules. It is worth adding efforts in this research, and software tools can play a relevant role as assistants for new computing models and solutions based on such models. The framework given by P-Lingua and MeCoSim provides a common infrastructure with facilities for design, analysis and simulation, among other tasks.

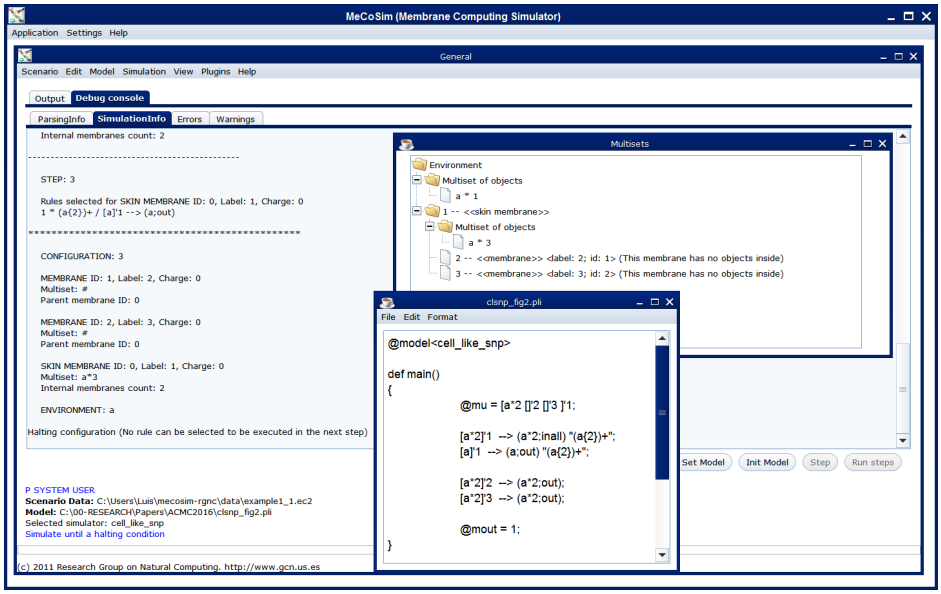

Figure 2: cSN P systems simulation in MeCoSim

All in all, a set of tools has been developed to incorporate cSN P systems within this framework. They include a language based on P–Lingua to define its syntactic ingredients, including *target indicators* for rewriting rules, combined with elements from classical SN P systems. In addition, parsing and debugging facilities have been provided, to check correctness and alert about possible errors. A simulation algorithm has been designed and developed to perform the corresponding computations. Besides, the integration in the framework allows the use of the existing visualization, analysis, customization and validation features. The tools developed have proved their ability to validate the solutions for cSN P systems, allowing the parsing, debugging and "non-deterministic" simulation of the different examples, and helping detecting some subtle details not considered in previous works.

It would be worth deepening into the study of the computational properties of cSN P systems, and complementary tools to aid in the design and validation tasks can definitely provide a value, specially when studying solutions to complex problems by P systems with a significant amount of elements interacting, whose evolution is not easy to analyse without the help of these software assistants.

Acknowledgements

The work of Luis Valencia-Cabrera, Tingfang Wu, Zhiqiang Zhang and Linqiang Pan was supported by National Natural Science Foundation of China (Grants no. 61320106005 and 61033003). In addition, we would like to thank Dr Luis Felipe Macías-Ramos, for his valuable contributions within the field of the simulation of "classical" spiking neural P systems and supportive explanations about the previous developments.

References

[1] Adorna, H.A., Cabarle, F.G.C., Macías-Ramos, L.F., Pan, L., Pérez-Jiménez, M.J., Song, B., Song, T., Valencia-Cabrera, L.: Taking the pulse of SN P systems: A Quick Survey. Multidisciplinary Creativity, Spandugino, Bucharest, 1–16 (2015).

[2] Buiu, C., Arsene, O., Cipu, C., Patrascu, M.: A software tool for modeling and simulation of numerical P systems. BioSystems, 103(3), 442–447 (2011).

[3] Cabarle, F.G.C., Adorna, H., Martínez-del-Amor, M. A.: A spiking neural P system simulator based on CUDA. In: Gheorghe, M., Păun, Gh., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) LNCS, vol. 7184, pp. 87–103. Springer, Heidelberg (2012).

[4] Cabarle, F.G.C., Adorna, H., Martínez-del-Amor, M. A., Pérez-Jiménez, M.J.: Spiking neural P system simulations on a high performance GPU platform. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) LNCS, vol. 7017, pp. 99–108. Springer, Heidelberg (2011).

[5] Cavaliere, M., Ibarra, O.H., Păun, Gh., Egecioglu, O., Ionescu, M., Woodworth S.: Asynchronous spiking neural P systems. Theor. Comput. Sci., 410(24-25), 2352–2364 (2009).

[6] Chen, H., Ishdorj, T.O., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with extended rules. In: Proceedings of the Fourth Brainstorming Week on Membrane Computing, pp. 241–265. Fénix Editora, Sevilla (2006).

[7] Ciobanu, G., Paraschiv, D.: P system software simulator. Fund. Inform., 49, 61–66 (2002).

[8] Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A.(eds.) LNCS, vol. 5391, pp. 187–203. Springer, Heidelberg (2009).

[9] Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Tissue P systems with channel states. Theor. Comput. Sci., 330, 101–116 (2005).

[10] García-Quismondo, M., Gutiérrez-Escudero, R., Martínez-del-Amor, M.A., Orejuela-Pinedo, E., Pérez-Hurtado, I.: P-Lingua 2.0: A software framework for cell-like P systems. Int. J. Comput. Commun., 4(3), 234–243 (2009).

[11] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Ramírez-Martínez, D.: A software tool for verification of spiking neural P systems. Natural Computing, 7(4), 485–497 (2008).

[12] Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fund. Inform., 71(2), 279–308 (2006).

[13] Jiang, K., Chen, W., Zhang, Y., Pan, L.: Spiking neural P systems with homogeneous neurons and synapses. Neurocomputing, 171, 1548–1555 (2016).

[14] Jiang, K., Pan, L.: Spiking neural P systems with anti-spikes working in sequential mode induced by maximum spike number. Neurocomputing, 171, 1674–1683 (2016).

[15] Macías-Ramos, L.F.: Developing Efficient Simulators for Cell Machines. PhD thesis. University of Seville (2016).

[16] Macías-Ramos, L.F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua based simulator for spiking neural P systems. In: Gheorghe, M., Păun, Gh., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) LNCS, vol. 7184, pp. 257–281. Springer, Heidelberg (2012).

[17] Macías-Ramos, L.F., Pérez-Jiménez, M.J., Song, T., Pan, L.: Extending simulation of asynchronous spiking neural P systems in P-Lingua. Fundam. Inform. 136(3), 253–267 (2015).

[18] Macías-Ramos, L.F., Song, B., Valencia-Cabrera, L., Pan, L., Pérez-Jiménez, M.J.: Membrane fission: A computational complexity perspective. Complexity, 21(6), 321–334 (2016).

[19] Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P Systems, Theor. Comput. Sci. 296(2), 295–326 (2003).

[20] Pan, L., Păun, Gh., Song, B.: Flat maximal parallelism in P systems with promoters. Theor. Comput. Sci., 623, 83–91 (2016).

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

[22] Păun, Gh.: P systems with active membranes: attacking NP-complete problems. J. Automata Lang. Combinatorics, 6(1), 75–90 (2001).

[23] Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. New Generat. Comput., 20(3), 295–305 (2002).

[24] Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010).

[25] Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Colomer, M.A., Riscos-Núñez, A.: Mecosim: A general purpose software tool for simulating biological phenomena by means of P systems. In: The Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010), pp. 637–643. Changsha (2010).

[26] Song, B., Pan, L.: The computational power of tissue-like P systems with promoters. Theor. Comput. Sci., 641, 43–52 (2016).

[27] Song, B., Pan, L., Pérez-Jiménez, M.J.: Cell-like P systems with channel states and symport/antiport rules. IEEE T. NanoBiosci., 15(6), 555–566 (2016).

[28] Song, B., Pérez-Jiménez, M.J., Pan, L.: An efficient time-free solution to SAT problem by P systems with proteins on membranes. J. Comput. Syst. Sci., 82(6), 1090–1099 (2016).

[29] Song, T., Xu, J., Pan, L.: On the universality and non-universality of spiking neural P systems with rules on synapses. IEEE T. on NanoBiosci., 14(8), 960–966 (2016).

[30] Song, B., Zhang, C., Pan, L.: Tissue-like P systems with evolutional symport/antiport rules. Inf. Sci., 378, 177–193 (2017).

[31] Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like spiking neural P systems. Theor. Comput. Sci. 623, 180–189 (2016).

[32] Wu, T., Zhang, Z., Pan, L.: On languages generated by cell-like spiking neural P systems. IEEE T. on NanoBiosci., 15(5), 455–467 (2016).

[33] Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An optimization spiking neural P system for approximately solving combinatorial optimization problems. Int. J. Neural Syst., 24(5), Article No. 1440006 (2014).

[34] Zhang, X., Pan, L., Păun, A.: On the universality of axon P systems, IEEE T. on Neur. Net. Lear., 26(11), 2816–2829 (2015).

[35] Java regular expressions lesson by Oracle.
`https://docs.oracle.com/javase/tutorial/essential/regex/`

[36] MeCoSim Website. `http://www.p-lingua.org/mecosim/`

[37] P-Lingua Website. `http://www.p-lingua.org/`