Rediscovering Spiking Neural P Systems
Simulating hierarchically-structured neurons

Luis Valencia Cabrera, Tingfang Wu, Zhiqiang Zhang,
Linqiang Pan, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville

Key Laboratory of Image Information Processing and Intelligent Control
Education Ministry of China, School of Automation
Huazhong University of Science and Technology

02-02-2017, BWMC 2017 - Sevilla

Contents

Contents

Contents

"Classical" P systems with multisets rewriting rules

Syntax definition

A (cell-like) P system (with multiset rewriting rules) of degree $q \geq 1$ is a tuple of the form

$$\Pi = (O, \mu, \mathscr{M}_1, \ldots, \mathscr{M}_q, \mathscr{R}_1, \ldots, \mathscr{R}_q, i_o)$$

- 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;
- $\mathscr{M}_i$, $1 \leq i \leq q$, is a multiset over O associated with membrane i at the beginning of a computation;
- $\mathscr{R}_i$, $1 \leq i \leq q$, 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).

“Classical” P systems with multisets rewriting rules

Types of rules of the system

The rules of the system are of the form:

$$u \longrightarrow v$$

where:

- u is a multiset over O
- v is a multiset over $O \times \{here, in, out\}$.

Note: each element of v is of the form (a, tar), where $a \in O$ and $tar \in \{here, in, out\}$.

"Classical" P systems with multisets rewriting rules

Semantic remarks

An extensive description of the semantics associated with this kind of systems can be found in *Handbook*[1], but we will briefly recall some relevant aspects.

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 a child of i, non-deterministically chosen.
- The **non-deterministic** choice is done **for each object individually**, even if many of them present that same target indicator.
- For instance: 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.

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

P systems with multisets rewriting rules

Rules definition extension

In what follows, the semantics of these P systems is extended to **consider multisets of objects** as *units associated with* the *target indicators* specified.

- A rule is of the form $u \longrightarrow v$, where:
  - $u \in M(O)$
  - v is a multiset over $M(O) \times \{here, in, out\}$
  - $M(O)$ being 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 (as a unit) to one of their children**, non-deterministically chosen.

P systems with multisets rewriting rules

Semantics

Semantics

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

Results are **associated** only **with halting computations**, encoded by the **contents of the output region** in the halting configuration.

Contents

Spiking neural P systems

Syntax of SN P systems with extended rules and no delay[2]

An SN P system of degree $q \geq 1$ is a tuple

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

- $O = \{a\}$ 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
  - R_i is a finite set of *rules* of the following two forms:
    1. $E/a^c \rightarrow a^p$, where E is a regular expression over a and $c \geq p \geq 1$
    2. $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \rightarrow a^p$ 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*)
- $out \in \{1, 2, \ldots, q\}$ indicates the *output neuron*.

[2]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*, 241–265. Fénix Editora, Sevilla (2006)

Spiking neural P systems

Types of rules

Spiking rule

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 $(i,j) \in syn$).

Forgetting rule

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.

Spiking neural P systems

Configurations and computations

- A *configuration* C_t at an instant t is a tuple $\left(p_1, \ldots, p_q\right)$ ($p_i \geq 0$ is the number of spikes in neuron σ_i at instant t).
- Initial configuration $\left(n_1, n_2, \ldots, n_q\right)$
- Configuration C_t yields configuration C_{t+1} in *one transition step* if we can pass from C_t to C_{t+1} 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
- From the initial config., a seq. of transition takes place, which 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 (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-).

Spiking neural P systems

Interpreting the output

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 $N_{in}(\Pi)$.

Contents

A new approach: Cell-like SN P Systems

cSN P systems - Syntax

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.
- $n_i, 1 \leq i \leq q$, number of spikes in region 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. $E/a^c \to u$, 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. $a^s \to \lambda$, where $s \geq 1$ (*forgetting rules*).
- $i_o \in \{1, \ldots, q\} \cup \{env\}$ indicates the output region (this is the environment if $i_o = env$).

As stated in CSNP[3], "one may also use the stronger indication in_j" (the object is sent to the specific child membrane with label j).

[3] Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like Spiking Neural P Systems. *Theoretical Computer Science*. 623, 180–189 (2016)

A new approach: Cell-like SN P Systems

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*;

(c) all children of membrane *i*.

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.

A new approach: Cell-like SN P Systems

Interpreting the output

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 [4], 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 [4].

[4] Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like Spiking Neural P Systems. *Theoretical Computer Science*. 623, 180–189 (2016)

Contents

A Unified Software Framework for P Systems

- P systems brought the emergence of new ideas, along with solid theoretical foundations
- Software tools started exploring the simulation of these systems, as *ad-hoc solutions* to specific problems
- Many types of P systems were defined $\longrightarrow$ availability of **general tools** to work with these novel solutions **became advisable**.
- **P-Lingua** meant a **significant milestone**: uniform framework to specify, debug and simulate these computing devices
- Use in real applications led to:
  - the need of additional tools for P systems designers
  - the delivery of end-user applications based on this framework would widen the scope and visibility of the underlying systems, abstracting internal details to the end-users of the applications designed, as ecologists, economists, etc

These goals were achieved by **MeCoSim**.

P-Lingua framework

P–Lingua framework includes:

- a general language to define P systems, *P–Lingua*
- 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.

Not only specific P systems can be specified, but also families of them, with parameters accepting different values depending on the instances to generate.

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
- 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.

Contents

A Language to Define Cell-like SN P Systems

Reserved Words and Special Elements

A new type of P systems has been defined, with genuine features, so the specific language for them must be set.

Despite the variety of systems available in P-Lingua, some well-known ingredients of P systems had not been incorporated so far.

In particular, rewriting rules including target indicators (*here*, *in*, *out*, in_{all}, in_j) were not covered. However, cSN P systems use these indicators. Consequently, new elements were needed.

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 *Handbook*. This model is now available within the framework, so that any P–Lingua file using these systems starts with the sentence:

```
@model<rewriting_systems>
```

A Language to Define Cell-like SN P Systems I

Model Specification

New target indicators are written as `here`, `out`, `in`, `inall` and `in{j}`.

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 will be explained.

A Language to Define Cell-like SN P Systems I

Membrane Structure and initial multisets

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.
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$\star n_i$;

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;
```

A Language to Define Cell-like SN P Systems

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}*.

Spiking rules, must support the extended form $E/a^c \rightarrow u$, with E a regular expression over O, $c \geq 1$. 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` can present pairs of the form (a^p, tar). In P-Lingua, its syntax would be:

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

with `TAR` expressed as described before.

A Language to Define Cell-like SN P Systems I

Interpreting the rules

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

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.

Regarding the syntax for regular expressions in P–Lingua, for *backwards compatibility* with other classical SN P systems, we adopted the same policies implemented yet. The mechanism to define and evaluate regular expressions is based on `regex` Java package.

The following subset of symbols can be used:

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

A Language to Define Cell-like SN P Systems

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`).

A Language to Define Cell-like SN P Systems

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

A New Simulator for Cell-like SN P Systems

A **new simulator** within P-Lingua framework captures 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:
    1. Selection of rules
    2. 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 (no rule is applicable).

The **selection phase** will check applicable rules. At most one rule can be executed at a given region in a computation step, so the selection stage will choose at most 1 rule, "non-deterministically" chosen, per region. The applicability of a rule is determined by the presence *in the region* of at least the number of spikes in the its hand side. In addition, its content must match the regular expression. If some target indicators *in*, in_{all} or in_j appear in the right hand side of a spiking rule, then the membrane must have child membranes (in the latter case, a child membrane labelled by j must exist).

The **execution phase** passes from configuration 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, 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.

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 MeCoSim site.

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).

Contents

Case Studies

Example 1

cSN P systems were defined, and the software tools developed for the design and simulation of cSN P systems were described. Let us see some examples illustrating their behaviour, showing the corresponding P-Lingua files specifying the solutions for the computer tools.
The first system considered is given formally 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 more spikes, repeatedly, but when the second rule is used (this may happen also in the first step) the number of spikes becomes odd and no further rule can be used. Therefore, $N_{in}(\Pi) = \{2n+1 \mid n \geq 0\}$.

Case Studies

Example 1

This system is introduced in the simulator developed as follows:

```
@model<cell_like_snp>

def main()
{
   @mu = [ ]'1;

   @ms(1) = a*2;

   [a*2]'1  --> (a*4;here) "(a{2})+";
   [a*2]'1  --> (a;here) "(a{2})+";

   @mout = 1;
}
```

Case Studies
Example 1

It could have been expressed, omitting the explicit indicators *here*:

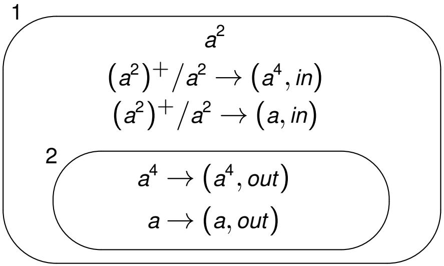
```
@model<cell_like_snp>

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 more compact way in `@mu`.

Case Studies

Example 2

The next example replaces the target indication *here* by some rules with target indicators *in* and *out*, albeit at the cost of using one further membrane, as shown in the figure:


1
a^2
$(a^2)^+/a^2 \rightarrow (a^4, in)$
$(a^2)^+/a^2 \rightarrow (a, in)$
2
$a^4 \rightarrow (a^4, out)$
$a \rightarrow (a, out)$

Case Studies

Example 2

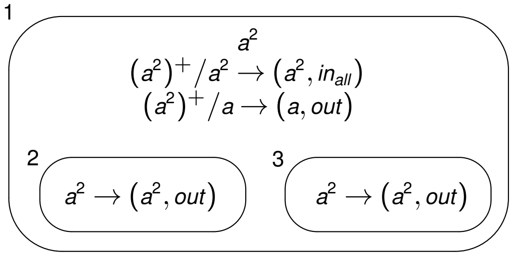
The corresponding P-Lingua file would be the following:

```
@model<cell_like_snp>

def main()
{
   @mu = [a*2 []'2 ]'1;

   [a*2]'1  --> (a*4;in) "(a{2})+";
   [a*2]'1  --> (a;in) "(a{2})+";

   [a*4]'2  --> (a*4;out);
   [a]'2  --> (a;out);

   @mout = 1;
}
```

Case Studies

Example 3

A third example is proposed, replacing the previous rules, producing 4 spikes and consuming 2 at each step, by the use of the target indication in_{all}, as indicated in the figure:



The same set (odd natural numbers) is obtained, but this time one spike is sent out to the environment. However, the output membrane is still membrane 1, as in the previous examples.

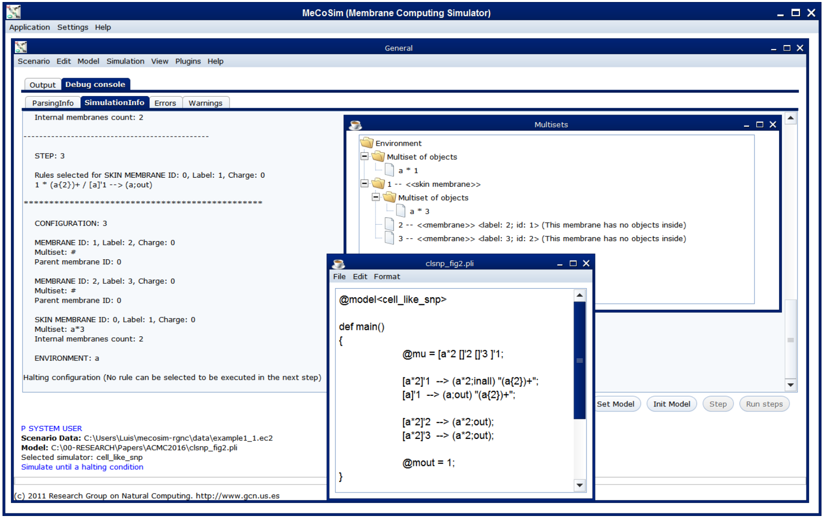
Case Studies
Example 3

It is 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;
}
```

Case Studies

This last example loaded in MeCoSim, including *step by step* information, P-Lingua file editor and the multisets viewer:


MeCoSim (Membrane Computing Simulator)
Application Settings Help
General
Scenario Edit Model Simulation View Plugins Help
Output Debug console
ParsingInfo SimulationInfo Errors Warnings
Multisets
clsnp_fig2.pli
File Edit Format
Set Model Init Model Step Run steps

```
Internal membranes count: 2

---------------------------------------------

STEP: 3

Rules selected for SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
1 * (a{2})+ / [a]'1 --> (a;out)

*************************************************

CONFIGURATION: 3

MEMBRANE ID: 1, Label: 2, Charge: 0
Multiset: #
Parent membrane ID: 0

MEMBRANE ID: 2, Label: 3, Charge: 0
Multiset: #
Parent membrane ID: 0

SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
Multiset: a*3
Internal membranes count: 2

ENVIRONMENT: a

Halting configuration (No rule can be selected to be executed in the next step)
```

```
Environment
  Multiset of objects
    a * 1
  1 -- <<skin membrane>>
    Multiset of objects
      a * 3
    2 -- <<membrane>> <label: 2; id: 1> (This membrane has no objects inside)
    3 -- <<membrane>> <label: 3; id: 2> (This membrane has no objects inside)
```

```
@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;
}
```

P SYSTEM USER
Scenario Data: C:\Users\Luis\mecosim-rgnc\data\example1_1.ec2
Model: C:\00-RESEARCH\Papers\ACMC2016\clsnp_fig2.pli
Selected simulator: cell_like_snp
Simulate until a halting condition

(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Contents

Conclusions

Cell-like spiking neural P systems set a bridge between cell-like P systems and spiking neural P systems. They are 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. Software tools can play a relevant role as assistants for new computing models and solutions. P-Lingua and MeCoSim provide a common infrastructure with facilities for design, analysis, simulation...

A set of tools was developed to incorporate **cSN P systems** within this framework, including a **language based on P–Lingua**, parsing and debugging facilities. 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.

Conclusions

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.

My sincere thanks

Thank you very much for your attention!