A P–Lingua Based Simulator for P Systems with Symport/Antiport Rules

Luis F. Macías-Ramos, Luis Valencia-Cabrera

Research Group on Natural Computing, Department of Computer Sci. and Artificial Intelligence

University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

lfmaciasr@us.es, lvalencia@us.es

Bosheng Song, Tao Song, Linqiang Pan[*]

Key Laboratory of Image Information Processing and Intelligent Control

School of Automation

Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

boshengsong@163.com, songtao0608@hotmail.com, lqpan@mail.hust.edu.cn

Mario J. Pérez-Jiménez

Research Group on Natural Computing, Department of Computer Sci. and Artificial Intelligence

University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

marper@us.es

Abstract. Inspired by mitosis process and membrane fission processes, cell-like P systems with symport/antiport rules and membrane division rules or membrane separation rules have been introduced, respectively. These computation systems have two key features: the ability to have infinite copies of some objects (within an active environment) and to generate an exponential workspace in polynomial time. In this work, we extend the `P-Lingua` framework for simulating that kind of P systems taking into account these two features. Consequently, a new simulator has been developed and included in pLinguaCore library. The functioning of the simulator has been checked by simulating efficient solutions to `SAT` problem using a family of cell-like P systems with symport/antiport rules and membrane division rules or membrane separation rules. The corresponding `MeCoSim` based application is also provided.

[*]Address for correspondence: Huazhong University of Science and Technology, Wuhan 430074, Hubei, China.

Keywords: Bio-inspired Computing, Membrane Computing, P System, `P-Lingua`, `MeCoSim`.

1. Introduction

Membrane computing is a relatively young branch of natural computing, which takes inspiration from the structure and functioning of living cells to provide parallel and distributed computational models, called membrane systems or P systems.

Since the first P system was introduced in [8], many variants have been developed, which can be divided into three categories: cell-like systems, inspired by the hierarchical membrane structure of eukaryotic cells [8]; tissue-like systems, inspired by the way in which cells organize and communicate within a net-like structure in tissues [5]; and neural-like systems, inspired by the way in which the neurons in the brain exchange information by means of the propagation of spikes [3].

As the number of new variants of P systems increases, it becomes necessary to develop simulation tools in order to assist researchers in relevant tasks such as (1) designing P system families to efficiently solve computationally hard problems, (2) formal verification of such solutions, and (3) the execution of computational devices modelling real-life phenomena within the paradigm of membrane computing. Following this need, `P-Lingua` framework was introduced [1, 15], providing both parsers and implementations of simulation algorithms for many existing variants of P systems. `P-Lingua` project is alive and new versions of the software are released in a fast development cycle.

Cell-like P systems with symport/antiport rules were introduced in [9], aiming to abstract the biological phenomenon of trans-membrane transport of couples of chemical substances, in the same or in opposite directions. Tissue-like P systems with symport/antiport rules were introduced in [10] by abstracting networks of elementary membranes such that some of them are linked by "communication channels". Let us recall some relevant aspects of membrane division and membrane separation rules. In both cases, the object that triggers the rule is consumed and two new membranes with the same label are created. In the case of membrane division, the remaining objects are *replicated* in the new membranes. In the case of membrane separation, the remaining objects are *distributed* according to a prefixed partition. Both cell-like systems with symport/antiport rules (without support for infinite copies of objects in the environment) and tissue-like systems with symport/antiport rules (with support for infinite copies of objects in the environment) with cell division or cell separation were already supported in the latest release of `P-Lingua` [2, 6, 13].

In this paper, the `P-Lingua` framework is extended to support the simulation of cell-like P systems with symport/antiport rules as communication rules (with support for infinity copies of objects in the environment) and with either membrane division rules or membrane separation rules.

This paper is structured as follows. Section 2 briefly recalls formal aspects of cell-like P systems with symport/antiport rules and with membrane division or membrane separation rules. Section 3 describes the extension of `P-Lingua` defined to support the simulation of such P systems. Section 4 is devoted to the corresponding simulation algorithm. In order to illustrate the functioning of the simulator, section 5 introduces an efficient solution to the `SAT` problem by using cell-like P systems with symport/antiport rules of length at most 3 and membrane division rules, while section 6 deals with a solution to `SAT` problem by using cell-like P systems with symport/antiport rules of length at most 3 and membrane separation rules (see [4] for details). Finally, conclusions and future work are given in section 7.

2. P systems with symport/antiport rules and membrane division rules or membrane separation rules

In this section, we recall some formal aspects of cell-like P sytems with symport/antiport rules, where membrane division rules inspired by the mitosis process, or membrane separation rules inspired by the membrane fission process, are allowed. Specifically, we introduce the recognizer version of such P systems. For a comprehensive and self-contained description, please refer to [4]. The reader is assumed to be familiar with basics of membrane computing (see [11]).

Definition 2.1. A recognizer P system with symport/antiport rules and membrane division of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \mathcal{E}, \Sigma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \ldots, \mathcal{R}_q, i_{in}, i_{out})$$

where:

1. Γ is a finite (working) *alphabet* which has two distinguished symbols `yes` and `no`;
2. $\mathcal{E}$ is an alphabet strictly contained in Γ, called the alphabet of the environment;
3. Σ is an (input) alphabet strictly contained in Γ such that $\mathcal{E} \subseteq \Gamma \setminus \Sigma$
4. μ is a rooted tree whose nodes are injectively labelled with $1, \ldots, q$ (the root is labelled by 1);
5. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over $\Gamma \setminus \Sigma$ such that at least one copy of `yes` or `no` is present in some of them;
6. $\mathcal{R}_i$, $1 \leq i \leq q$, are finite sets of rules over Γ of the following forms:
   - ⋆ *Symport rules*: $(u\,,\, out)$ or $(u\,,\, in)$, where u is a finite multiset over Γ such that $|u| > 0$, being $|u|$ the *length* of the string u, that is, the number of ocurrences in u of symbols from Γ;
   - ⋆ *Antiport rules*: $(u\,,\, out\,;\, v\,,\, in)$, where u, v are finite multisets over Γ such that $|u| > 0$ and $|v| > 0$;
   - ⋆ *Division rules*: $[\,a\,]_i \to [\,b\,]_i\,[\,c\,]_i$, where $a, b, c \in \Gamma$, $i \in \{2, \ldots, q\}$, and i is the label of a leaf of the tree μ;
7. $i_{in} \in \{1, \ldots, q\}$ and $i_{out} = 0$ (the label of the environment is 0);

and with the following properties:

- All computations halt;
- if $\mathcal{C}$ is a computation of Π, then either symbol `yes` or symbol `no` (but not both) must have been released into the environment, and only at the last step of the halting computation.

In a similar way, we can define a P system with symport/antiport rules and membrane separation rules (instead of membrane division rules). The syntax of this kind of rules associated with a membrane i is the following:

⋆ *Separation rules*: $[\, a \,]_i \to [\, \Gamma_1 \,]_i \, [\, \Gamma_2 \,]_i$, where $\{\Gamma_1, \Gamma_2\}$ is a given partition of Γ, $a \in \Gamma$, $i \in \{2, \dots, q\}$, $i \neq i_{out}$, and i is the label of a leaf of the tree μ (all separation rules are associated with the *same* partition of the working alphabet).

With respect to the semantics of such P systems, the rules in a system are applied in a non-deterministic maximally parallel manner (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added), with the following important remark: when a membrane i is divided (resp., separated), the division rule (resp., separation rule) is the only one from $\mathcal{R}_i$ which is applied for that membrane at that step. The new membranes resulting from division (resp., separation) can participate in the interaction with other membranes or the environment by means of communication rules at the next step – providing that they are not divided (resp., separated) once again.

We denote by **CDC** (resp., **CSC**) the class of recognizer P systems with symport/antiport rules and membrane division rules (resp., membrane separation rules). We denote by **CDC**(k) (resp., **CSC**(k)) the class of recognizer P systems with symport/antiport rules and membrane division rules (resp. membrane separation rules) such that the communication rules have length at most k.

3. P–Lingua syntax for P systems with membrane division rules or membrane separation rules

Taking the `P-Lingua` syntax for tissue-like P systems with cell separation rules introduced in [13] as a starting point, we specify the `P-Lingua` syntax for cell-like P systems with symport/antiport rules and with either membrane division rules or membrane separation rules.

Definition of P system model
In order to define a cell-like P system with either membrane division rules or membrane separation rules, the first line of the `P-Lingua` file should be as follows: `@model<infEnv_symport_antiport>`.

Definition of the environment in the membrane structure
In order to specify the environment in the `P-Lingua` syntax, a virtual membrane has to be defined. This membrane will be labelled with the `environment` label and will be placed as the outer-most membrane, in the following way: `@mu = [ ...  ]'environment;`, where `...` stands for the definition of the membrane structure as usual (starting with the skin membrane).

Definition of the alphabet of the environment
In order to define the objects initially placed in environment (each appearing in an arbitrary number of copies), the following sentence must be written: `@msInfEnv += OBJECTS;`, where `OBJECTS` is a comma-separated list of objects.

Using the += operator enables cumulative addition of objects to the corresponding multiset through a sequence of several sentences like the one stated above. Alternatively, the operator = can be used to express the contents of the multiset with a single sentence.

Definition of the partition of the working alphabet
In order to define the partition Γ_1 and Γ_2 of the working alphabet Γ, associated with the P system with membrane separation, the following pair of sentences is used:

`@ms1 += OBJECTS1;` and `@ms2 += OBJECTS2;`

with the use of operators += and = as stated above. Let us recall that Γ_1 and Γ_2 verify that $\Gamma_1, \Gamma_2 \neq \emptyset, \Gamma_1 \cap \Gamma_2 = \emptyset, \Gamma_1 \cup \Gamma_2 = \Gamma$.

Definition of symport/antiport rules
Symport rules of type $(u\,,\,in) \in \mathcal{R}_i$ are defined as `u[]'i --> [u]'i;`, while rules of type $(u\,,\,out) \in \mathcal{R}_i$ are defined as `[u]'i --> u[]'i;`.
Antiport rules of the form $(u, out; v, in) \in \mathcal{R}_i$ are defined in the following way: `u[v]'i --> v[u]'i;`.

Definition of membrane division rules
Membrane division rules $[\,a\,]_i \to [\,b\,]_i\,[\,c\,]_i$ are defined with the syntax: `[a]'i --> [b]'i[c]'i;`

Definition of membrane separation rules
Membrane separation rules $[\,a\,]_i \to [\,\Gamma_1\,]_i\,[\,\Gamma_2\,]_i$ are defined with the syntax: `[a]'i --> []'i[]'i;`.

Definition of membrane separation rules
Membrane separation rules $[\,a\,]_i \to [\,\Gamma_1\,]_i\,[\,\Gamma_2\,]_i$ are defined with the syntax: `[a]'i --> []'i[]'i;`.

The following is a couple of small examples illustrating the previously detailed syntax

```
@model<infEnv_symport_antiport>
def main()
{
@mu = [ [ [ []'2 ]'1]'0 ]'environment; /* defining membrane structure */
@ms(1) = a,b,c; /* initial multiset for membrane 1 */
@ms(2) = e,f; /* initial multiset for membrane 2 */
@msInfEnv = d; /* environment objects with arbitrary number of copies */
[a]'1 --> [b]'1[c]'1; /* division rule */
}
```

In this example, a simple cell-like symport/antiport P system with cell division is defined, where a division rule is fired by object a which is turned into objects b and c respectively.

```
@model<infEnv_symport_antiport>
def main()
{
@mu = [ [ [ []'2 ]'1]'0 ]'environment; /* defining membrane structure */
@ms(1) = a,b,c,e,f; /* initial multiset for membrane 1 */
@ms(2) = g,h; /* initial multiset for membrane 2 */
@msInfEnv = d; /* environment objects with arbitrary number of copies */
@ms1 += b,c; /* first partition */
@ms2 += e,f; /* second partition */

[a]'1 --> []'1[]'1; /* separation rule */
}
```

In this example, a simple cell-like symport/antiport P system with cell separation is defined, where a separation rule is fired by object a which is consumed.

4. A simulator for P systems in CDC and CSC

Simulators for computational models are usually very interesting assistants to help in design tasks and the formal verification of solutions to decision problems defined in such models. Simulators work with an inference engine implemented through an algorithm capturing the semantics of the corresponding P system model. In fact, that algorithm must reproduce only one possible computation of such P system. Let us recall that when a family of recognizer P systems is used to solve a decision problem, each member of the family processing an instance is confluent, in the sense that all computations for a given input must generate the same output. Therefore, our simulator is suitable to work with this kind of P systems.

In this section, pLinguaCore library is expanded to include a built-in simulator for P systems in **CDC** and **CSC**, and the corresponding simulation algorithm is also introduced.

In what follows, for simplicity, we denote $(u, in)_i$ ($(u, out)_i$ or $(u\,,\,out\,;\,v\,,\,in)_i$, respectively) instead of $(u\,,\,in) \in \mathcal{R}_i$ ($(u\,,\,out) \in \mathcal{R}_i$ or $(u\,,\,out\,;\,v\,,\,in) \in \mathcal{R}_i$, respectively).

4.1. Simulation algorithm

The simulation algorithm described below generates one possible computation of a P system in the class **CDC** or **CSC**.

```
I. Initialization

  1. Let $C_0$ be the initial configuration with $q$ membranes denoted by $m_1, \ldots, m_q$
  2. Let $m_0$ be a virtual membrane with label $0$ representing the environment, where all initial
     objects have infinite multiplicity
  3. Let $R_{sel} = \{\}$ be a set of tuples containing the selected rules to be executed at each step
     of computation, the identifiers of the membranes involved and the number of times that
     each rule will be executed (omitted if 1)
  4. Let $C_t = C_0$ be the current configuration
  5. Let $SetSymAnt = \{\}$ be the set of membranes that are executing a symport/antiport rule in
     the current configuration
  6. Let $SetDivSep = \{\}$ be the set of membranes that are executing a division or separation
     rule in the current configuration

II. Selection of symport/antiport rules

  1. For each membrane $m_i \in C_t$ with label $i$ do

     (a) For each send-in symport rule $(u\,,\,in)_i$ do

         • Let $m_k$ be the parent membrane of $m_i$
         • Let $M$ be the greatest number such that the multiset of $m_k$ contains $M$ copies
           of the multiset $u$
         • Remove $M$ copies of $u$ from the multiset of $m_k$
         • Add $\langle m_i, m_k, (u\,,\,in)_i, M\rangle$ to $R_{sel}$
         • Add $m_i$ to $SetSymAnt$

     (b) For each send-out symport rule $(u\,,\,out)_i$ do

         • Let $m_k$ be the parent membrane of $m_i$
         • Let $M$ be the greatest number such that the multiset of $m_i$ contains $M$ copies
           of the multiset $u$
         • Remove $M$ copies of $u$ from the multiset of $m_i$
```

```
            • Add $\langle m_i, m_k, (u\,,\,out)_i, M\rangle$ to $R_{sel}$
            • Add $m_i$ to $SetSymAnt$

        (c) For each antiport rule $(u\,,\,out\,;\,v\,,\,in)_i$ do

            • Let $m_k$ be the parent membrane of $m_i$
            • Let $M$ be the greatest number such that the multiset of $m_i$ contains
              $M$ copies of the multiset $u$
            • Let $N$ be the greatest number (considering that $N \leq M$) such that
              the multiset of $m_k$ contains $N$ copies of the multiset $v$
            • Remove $N$ copies of $u$ from the multiset of $m_i$
            • Remove $N$ copies of $v$ from the multiset of $m_k$
            • Add $\langle m_i, m_k, (u\,,\,out\,;\,v\,,\,in)_i, N\rangle$ to $R_{sel}$
            • Add $m_i$ to $SetSymAnt$

III. Selection of division and separation rules

    1. For each membrane $m_i$ at configuration $C_t$ with label $i$ not contained in
       $SetSymAnt$ and not contained in $SetDivSep$ do

        (a) If the simulated P system is in the class CDC, then

            i. For each division rule $[a]_i \to [b]_i[c]_i$ do

                • If $a$ is contained in the multiset of $m_i$, then

                    – Remove one instance of $a$ from the multiset of $m_i$
                    – Add $\langle m_i, [a]_i \to [b]_i[c]_i\rangle$ to $R_{sel}$
                    – Add $m_i$ to $SetDivSep$

        (b) If the simulated P system is in the class CSC, then

            i. For each separation rule $[a]_i \to [\Gamma_1]_i[\Gamma_2]_i$ do

                • If $a$ is contained in the multiset of $m_i$, then

                    – Remove one instance of $a$ from the multiset of $m_i$
                    – Add $\langle m_i, [a]_i \to [\Gamma_1]_i[\Gamma_2]_i\rangle$ to $R_{sel}$
                    – Add $m_i$ to $SetDivSep$

IV. Execution of rules

    1. For each tuple $\langle m_i, m_k, (u\,,\,in)_i, M\rangle$ from $R_{sel}$ do

        (a) Add $M$ copies of $u$ to the multiset of $m_i$

    2. For each tuple $\langle m_i, m_k, (u\,,\,out)_i, M\rangle$ from $R_{sel}$ do

        (a) Add $M$ copies of $u$ to the multiset of $m_k$

    3. For each tuple $\langle m_i, m_k, (u\,,\,out\,;\,v\,,\,in)_i, N\rangle$ from $R_{sel}$ do

        (a) Add $N$ copies of $v$ to the multiset of $m_i$
        (b) Add $N$ copies of $u$ to the multiset of $m_k$

    4. For each tuple $\langle m_i, [a]_i \to [b]_i[c]_i\rangle$ from $R_{sel}$ do

        (a) Create a new cell $m'_i$ with label $i$ and empty multiset
        (b) Copy in the multiset of $m'_i$ the objects that are contained in the
            multiset of $m_i$
        (c) Add one instance of $b$ to the multiset of $m_i$
        (d) Add one instance of $c$ to the multiset of $m'_i$
```

```
  5. For each tuple $\langle m_i, [a]_i \to [\Gamma_1]_i[\Gamma_2]_i \rangle$ from $R_{sel}$ do
     (a) Create a new cell $m'_i$ with label $i$ and empty multiset
     (b) Copy in the multiset of $m'_i$ the objects of $\Gamma_2$ that are contained in
         the multiset of $m_i$
     (c) Remove from the multiset of $m_i$ the objects of $\Gamma_2$

V. Ending
  1. If $R_{sel} \neq \emptyset$, then
     • Let $C_{t+1} = C_t$
     • Let $R_{sel} = \{\}$
     • Let $SetSymAnt = \{\}$
     • Let $SetDivSep = \{\}$
     • Go to II
  2. End
```

We consider a total order in the finite set of rules. At each computation step, for each membrane, the algorithm processes the symport/antiport rules first and then goes through the membrane division or membrane separation rules (depending on the kind of P system specified) for those membranes not executing any of the previously processed symport/antiport rules. This strategy is motivated by the intuition that division and separation rules add more descriptive complexity than symport/antiport rules, so we give them "lower priority".

5. Solving SAT by a family recognizer P systems from **CDC**(3)

5.1. Description of a family of recognizer P systems from **CDC**(3) solving SAT

In order to check the functioning of the simulator defined in section 4, we give a solution of SAT by using recognizer P systems with symport/antiport rules and membrane division rules from **CDC**(3). The solution follows a brute force algorithm and consists of the following phases:

- *Generation stage*: All possible truth assignments associated with the input formula φ are produced by using membrane division rules in an adequate way. Thus, an exponential amount of space (in terms of number of cells of the system) is generated. Each membrane codifies one truth assignment.

- *Checking stage*: Inside each membrane, it is checked whether or not the formula φ is satisfiable by the truth assignment encoded by that membrane.

- *Output stage*: The system sends the right answer to the environment, depending on the existence or not of some membrane codifying a truth assignment making the given formula true.

Let us consider the polynomial time computable function $\langle n, m \rangle = ((n+m)(n+m+1)/2) + n$, which is a bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. We define a family $\mathbf{\Pi} = \{\Pi(\langle n, m \rangle) \mid n, m \in \mathbb{N}\}$ of tissue P systems from **CDC**(3), such that $\Pi(\langle n, m \rangle)$ will process all instances $\varphi = C_1 \wedge \ldots \wedge C_m$ of SAT with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{C_1, \ldots, C_m\}$, provided that the appropriate input multiset $cod(\varphi) = \{x_{i,j} \mid x_i \in C_j\} \cup \{\overline{x}_{i,j} \mid \neg x_i \in C_j\}$ is supplied to the system.

For each $n, m \in \mathbb{N}$, the recognizer P system $\Pi(\langle n, m\rangle)$ from **CDC**(3) is defined as follows.

(1) Working alphabet:

$\Gamma = \Sigma \cup \mathcal{E} \cup \{f_0, b, c\} \cup \{f'_r \mid 0 \le r \le n+2m+1\} \cup \{a_i \mid 1 \le i \le n\} \cup \{\delta_{j,0} \mid 0 \le j \le m\}$,

where the input alphabet is $\Sigma = \{x_{i,j}, \overline{x}_{i,j} \mid 1 \le i \le n \wedge 1 \le j \le m\}$, and the alphabet of the environment is

$$\begin{aligned} \mathcal{E} \;=\; & \{S\} \cup \{d_{i,j,k}, \overline{d}_{i,j,k} \mid 1 \le i \le n \wedge 1 \le j \le m \wedge 1 \le k \le n-1\} \cup \\ & \{T_i, F_i \mid 1 \le i \le n\} \cup \{f_r \mid 1 \le r \le n+2m\} \cup \{E_j \mid 0 \le j \le m\} \\ & \{\delta_{j,r} \mid 0 \le j \le m \wedge 1 \le r \le n-1\} \cup \{e_{i,j}, \overline{e}_{i,j} \mid 1 \le i \le n \wedge 1 \le j \le m\}. \end{aligned}$$

(2) Membrane structure: $\mu = [\,[\;\;]_2\,[\;\;]_3]_1$. The input membrane is the membrane labelled by 1.

(3) Initial multisets

$$\begin{aligned} \mathcal{M}_1 &= \{f_0, \texttt{yes}\} \cup \{f'_r \mid 1 \le r \le n+2m+1\} \cup \{\delta_{j,0} \mid 0 \le j \le m\}; \\ \mathcal{M}_2 &= \{a_i \mid 1 \le i \le n\} \cup \{b, c\}; \\ \mathcal{M}_3 &= \{f'_0, \texttt{no}\}. \end{aligned}$$

(4) Rules in $\mathcal{R}_1$:

- Rules to generate 2^{n-1} copies of each object $x_{i,j}$ and $\overline{x}_{i,j}$ of $cod(\varphi)$ in the membrane 1 of $\mathcal{C}_{n+1}$, but changing its name x by e, and denote it by $(cod(\varphi))_e^{2^n}$.

$$\begin{aligned} &(x_{i,j}\,,\, out\,;\, d^2_{i,j,1}\,,\, in), 1 \le i \le n, 1 \le j \le m; \\ &(\overline{x}_{i,j}\,,\, out\,;\, \overline{d}^2_{i,j,1}\,,\, in), 1 \le i \le n, 1 \le j \le m; \\ &(d_{i,j,k}\,,\, out\,;\, d^2_{i,j,k+1}\,,\, in), 1 \le i \le n, 1 \le j \le m, 1 \le k \le n-2; \\ &(\overline{d}_{i,j,k}\,,\, out\,;\, \overline{d}^2_{i,j,k+1}\,,\, in), 1 \le i \le n, 1 \le j \le m, 1 \le k \le n-2; \\ &(d_{i,j,n-1}\,,\, out\,;\, e_{i,j}\,,\, in), 1 \le i \le n, 1 \le j \le m; \\ &(\overline{d}_{i,j,n-1}\,,\, out\,;\, \overline{e}_{i,j}\,,\, in), 1 \le i \le n, 1 \le j \le m. \end{aligned}$$

- Output rule with **positive** answer: $(E_0\, f_{n+2m}\, \texttt{yes}\,,\, out)$.
- Output rule with **negative** answer: $(f_{n+2m}\, \texttt{no}\,,\, out)$.
- Rules to generate in the membrane 1 of $\mathcal{C}_{3n}$ the objects $E_1^{2^n}$, and in the membrane 1 of $\mathcal{C}_{3n+1}$ the objects $E_0^{2^n}, E_2^{2^n}, \ldots, E_p^{2^n}$.

$$\begin{aligned} &(\delta_{j,r}\,,\, out\,;\, \delta^2_{j,r+1}\,,\, in), 0 \le j \le m, 0 \le r \le n-2; \\ &(\delta_{j,n-1}\,,\, out\,; E^2_j\,,\, in),\; 0 \le j \le m; \\ &(f_i\,, out\,;\, f_{i+1}\,, in), 0 \le i \le n+2m-1. \end{aligned}$$

(5) Rules in $\mathcal{R}_2$:

- Division rules: $[\,a_i\,]_2 \rightarrow [\,b\,]_2\ [\,c\,]_2,\ 1 \leq i \leq n.$
- Checking rules:

$$\begin{aligned}
&(b\,,\,out\,;\,E_1\,,\,in);\ (c\,,\,out\,;\,E_0\,,\,in);\\
&(E_j\,T_i\,,\,out\,;\,e_{i,j}\,,\,in), 1 \leq i \leq n, 1 \leq j \leq m;\\
&(E_j\,F_i\,,\,out\,;\,\overline{e}_{i,j}\,,\,in), 1 \leq i \leq n, 1 \leq j \leq m;\\
&(e_{i,j}\,,\,out\,;\,E_{j+1}\,T_i\,,\,in), 1 \leq i \leq n, 1 \leq j \leq m-1;\\
&(\overline{e}_{i,j}\,,\,out\,;\,E_{j+1}\,F_i\,,\,in), 1 \leq i \leq n, 1 \leq j \leq m-1;\\
&(e_{i,m}\,E_0\,,\,out), 1 \leq i \leq n;\\
&(\overline{e}_{i,m}\,E_0\,,\,out), 1 \leq i \leq n.
\end{aligned}$$

(6) Rules in $\mathcal{R}_3$: $(f'_r\,,\,out\,;\,f'_{r+1}\,,\,in),\ 0 \leq r \leq n+2m$, and $(f'_{n+2m+1}\,\texttt{no}\,,\,out)$.

The P-Lingua source code that defines a cell-like P system belonging to the family specified above is available at [14]. Once a new simulator is available, it is interesting to provide a visual environment oriented to end users that eases the virtual experimentation process. Consequently, a MeCoSim based custom application for the given solution to SAT problem has been provided. The corresponding source files can also be found at [14].

In order to conduct simulation experiments, we can download the corresponding MeCoSim application. Then, according to the MeCoSim user guide, we do the following.

- We launch the application (see Fig. 1).
- We input the number of variables (n) and clauses (m) in the corresponding table (see Fig. 2).
- We input the formula by using the SAT equation plugin and copy it to the corresponding table (see Figs. 3 and 4).
- From here, we initialize the model from the Debug label; also we can check the different configurations (see Fig. 5).
- We can simulate the model until the end and get the corresponding result graphically (see Fig. 6). The chart label will read "yes" or "no" depending of the decision problem result.

5.2. Simulation results

We have simulated several P systems of the defined family solving different instances of the SAT problem. Some results are shown in Table 1, including the execution time for each simulation (we denote $\neg x$ by $\overline{x}$, $l_1 \vee l_2$ by $l_1 + l_2$ and $C_1 \wedge C_2$ by $C_1 \cdot C_2$, where x is a propositional variable, l_i are literals and C_j are clauses). Execution times have been calculated averaging the times obtained after performing three simulations for each instance.

Table 1. Formula satisfiability and simulation time

Formula	**n**	**m**	**SAT**	**Time (s)**
$(\bar{x}_1 + \bar{x}_2) \cdot x_1 \cdot x_2$	2	3	F	0,121
$(\bar{x}_1 + \bar{x}_2) \cdot x_2 \cdot (\bar{x}_1 + x_2)$	2	3	T	0,099
$(x_1 + x_2) \cdot (x_1 + x_2 + \bar{x}_3) \cdot \bar{x}_1 \cdot \bar{x}_2$	3	4	F	0,219
$(\bar{x}_1 + x_2) \cdot \bar{x}_1 \cdot x_3 \cdot (\bar{x}_1 + x_3)$	3	4	T	0,201
$(x_1 + x_4) \cdot (x_1 + \bar{x}_4) \cdot x_3 \cdot (x_2 + \bar{x}_3 + x_4) \cdot \bar{x}_1$	4	5	F	0,315
$(x_3 + \bar{x}_4) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4) \cdot (x_1 + x_2) \cdot (\bar{x}_1 + x_2 + x_3 + x_4) \cdot (\bar{x}_1 + x_3)$	4	5	T	0,325
$(x_1 + \bar{x}_2 + x_3 + x_5) \cdot (\bar{x}_1 + x_4) \cdot (\bar{x}_2 + \bar{x}_4) \cdot x_4 \cdot x_2 \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4)$	5	6	F	0,570
$(x_3 + x_4) \cdot (x_4 + \bar{x}_5) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4) \cdot (x_1 + \bar{x}_2 + x_4) \cdot (x_1 + \bar{x}_3 + x_4) \cdot (x_3 + x_5)$	5	6	T	0,564
$(x_3 + x_5 + x_6) \cdot (x_3 + \bar{x}_4 + x_5 + \bar{x}_6) \cdot \bar{x}_3 \cdot \bar{x}_6 \cdot (x_1 + \bar{x}_2 + \bar{x}_3 + x_5 + x_6) \cdot (x_1 + x_4 + x_5) \cdot (\bar{x}_5 + x_6)$	6	7	F	0,932
$(\bar{x}_1 + \bar{x}_2 + x_5) \cdot (x_2 + x_3) \cdot (x_3 + \bar{x}_5 + \bar{x}_6) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4 + x_5 + x_6) \cdot (\bar{x}_2 + \bar{x}_3) \cdot (x_2 + x_3 + x_6) \cdot (x_1 + \bar{x}_2 + x_3 + x_4 + x_5 + x_6)$	6	7	T	1,017
$(\bar{x}_5 + \bar{x}_6 + \bar{x}_7) \cdot (x_3 + \bar{x}_4 + x_7) \cdot (\bar{x}_1 + x_3 + x_5 + x_6 + \bar{x}_7) \cdot (x_1 + x_3 + \bar{x}_5 + x_6 + x_7) \cdot (x_2 + x_6) \cdot (x_2 + \bar{x}_6) \cdot \bar{x}_2 \cdot (x_2 + x_3 + x_4 + \bar{x}_5 + x_7)$	7	8	F	1,974
$(\bar{x}_2 + x_5 + x_6 + x_7) \cdot (x_2 + \bar{x}_4 + \bar{x}_5 + \bar{x}_7) \cdot (x_1 + x_2 + \bar{x}_3 + \bar{x}_6 + x_7) \cdot (x_1 + x_2 + x_3 + \bar{x}_5 + x_6 + \bar{x}_7) \cdot (\bar{x}_3 + \bar{x}_5 + x_6 + \bar{x}_7) \cdot (x_1 + x_2 + \bar{x}_3 + \bar{x}_7) \cdot (\bar{x}_1 + x_2 + \bar{x}_4 + \bar{x}_6) \cdot (x_3 + x_5 + x_6 + \bar{x}_7)$	7	8	T	1,804
$(x_3 + x_4 + \bar{x}_6 + \bar{x}_8) \cdot (x_6 + \bar{x}_7) \cdot (\bar{x}_2 + x_3 + \bar{x}_4 + x_5 + x_8) \cdot x_7 \cdot (x_1 + \bar{x}_2 + x_5 + \bar{x}_7 + \bar{x}_8) \cdot (x_2 + x_7 + x_8) \cdot (\bar{x}_6 + \bar{x}_7) \cdot (x_1 + x_5 + \bar{x}_8) \cdot (x_1 + \bar{x}_4 + x_5 + \bar{x}_6 + x_7)$	8	9	F	3,530
$(x_1 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7 + \bar{x}_8) \cdot (x_2 + x_3 + x_4 + \bar{x}_6 + \bar{x}_7 + x_8) \cdot (x_3 + x_4 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7 + \bar{x}_8) \cdot (\bar{x}_1 + \bar{x}_3 + \bar{x}_4 + \bar{x}_5 + x_6 + \bar{x}_7 + x_8) \cdot (\bar{x}_3 + \bar{x}_7) \cdot (x_4 + x_5 + \bar{x}_7) \cdot (x_1 + x_3 + \bar{x}_4) \cdot (x_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_4 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7) \cdot (x_4 + \bar{x}_5 + \bar{x}_6 + x_7 + \bar{x}_8)$	8	9	T	3,632
$(\bar{x}_2 + \bar{x}_3 + x_5 + x_7) \cdot (x_2 + x_5 + x_6 + x_7 + x_9) \cdot (\bar{x}_3 + x_5 + x_7 + x_8) \cdot (x_1 + \bar{x}_4 + \bar{x}_5 + x_6 + x_8) \cdot (\bar{x}_2 + x_3 + x_5 + x_7 + x_8 + \bar{x}_9) \cdot (\bar{x}_2 + \bar{x}_4 + x_7 + x_9) \cdot (\bar{x}_2 + x_4 + x_6 + x_9) \cdot x_1 \cdot x_5 \cdot (\bar{x}_1 + \bar{x}_5)$	9	10	F	8,262
$(x_3 + x_8) \cdot (x_1 + \bar{x}_2 + x_5 + \bar{x}_6 + x_9) \cdot (x_3 + x_6 + x_9) \cdot (x_3 + x_5 + \bar{x}_6 + \bar{x}_8) \cdot (x_1 + x_2 + \bar{x}_5 + x_7 + \bar{x}_8 + \bar{x}_9) \cdot (\bar{x}_1 + x_2 + \bar{x}_4 + x_5 + \bar{x}_6 + \bar{x}_7 + x_9) \cdot (x_1 + x_2 + x_4 + \bar{x}_6 + x_8 + \bar{x}_9) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4 + x_7 + \bar{x}_8) \cdot (\bar{x}_1 + x_2 + x_3 + x_5 + \bar{x}_6 + x_8 + \bar{x}_9) \cdot (x_2 + \bar{x}_3 + x_4 + \bar{x}_6 + \bar{x}_7 + \bar{x}_9)$	9	10	T	7,993

6. Solving SAT by a family of recognizer P systems from CSC(3)

Let us recall that with a P system family from **CDC**(3) all possible truth assignments associated to a formula with n variables can be directly generated by using membrane division rules fired by objects $a_i, 1 \leq i \leq n$, producing objects T_i, F_i and replicating the rest of the membrane content. Nevertheless, this cannot be directly accomplished with a P system family from **CSC**(3), since applying a membrane separation rule an object is consumed while the rest of the membrane content is distributed among the two newly created membranes. To achieve the desired result, it is necessary to develop a mechanism to generate an exponential amount of objects in the membrane to be separated and that can be accomplished through the skin membrane by exchanging objects with the environment by using antiport rules of length 3. These rules allow incorporating to the skin two objects from the environment in exchange for an object from the skin.

The **CSC** simulator has played a key role to design and formally verify that a $\mathbf{CSC}(3)$ family solves SAT problem in polynomial time due to the extraordinary hardness of the family design. We have proceeded as follows. The solution has been structured in different modules with each of them being checked by using the simulator with several relevant instances. Finally, the module has been incorporated into the general solution. With respect to the formal verification, the simulator was used to check that the identified invariants were corroborated in the corresponding configurations.

In order to check the functioning of the simulator presented in section 4, we perform virtual experiments on the solution of `SAT` by using a family of recognizer P systems with symport/antiport rules and membrane separation rules from $\mathbf{CSC}(3)$ (see [4] for the details of the solution).

The `P-Lingua` source code that defines a cell-like P system belonging to the family specified above and the corresponding `MeCoSim` custom application source files can be found at [14].

6.1. Results of simulation

We have simulated several P systems of the defined family solving the same instances of the `SAT` problem as in the previous section. The simulation results are shown in Table 2.

Table 2. Formula satisfiability and simulation time

Formula	**n**	**m**	**SAT**	**Time (s)**
$(\bar{x}_1 + \bar{x}_2) \cdot x_1 \cdot x_2$	2	3	F	0,233
$(\bar{x}_1 + \bar{x}_2) \cdot x_2 \cdot (\bar{x}_1 + x_2)$	2	3	T	0,224
$(x_1 + x_2) \cdot (x_1 + x_2 + \bar{x}_3) \cdot \bar{x}_1 \cdot \bar{x}_2$	3	4	F	0,491
$(\bar{x}_1 + x_2) \cdot \bar{x}_1 \cdot x_3 \cdot (\bar{x}_1 + x_3)$	3	4	T	0,487
$(x_1 + x_4) \cdot (x_1 + \bar{x}_4) \cdot x_3 \cdot (x_2 + \bar{x}_3 + x_4) \cdot \bar{x}_1$	4	5	F	0,827
$(x_3 + \bar{x}_4) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4) \cdot (x_1 + x_2) \cdot (\bar{x}_1 + x_2 + x_3 + x_4) \cdot (\bar{x}_1 + x_3)$	4	5	T	0,981
$(x_1 + \bar{x}_2 + x_3 + x_5) \cdot (\bar{x}_1 + x_4) \cdot (\bar{x}_2 + \bar{x}_4) \cdot x_4 \cdot x_2 \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4)$	5	6	F	2,369
$(x_3 + x_4) \cdot (x_4 + \bar{x}_5) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4) \cdot (x_1 + \bar{x}_2 + x_4) \cdot (x_1 + \bar{x}_3 + x_4) \cdot (x_3 + x_5)$	5	6	T	2,312
$(x_3 + x_5 + x_6) \cdot (x_3 + \bar{x}_4 + x_5 + \bar{x}_6) \cdot \bar{x}_3 \cdot \bar{x}_6 \cdot (x_1 + \bar{x}_2 + \bar{x}_3 + x_5 + x_6) \cdot (x_1 + x_4 + x_5) \cdot (\bar{x}_5 + x_6)$	6	7	F	4,877
$(\bar{x}_1 + \bar{x}_2 + x_5) \cdot (x_2 + x_3) \cdot (x_3 + \bar{x}_5 + \bar{x}_6) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4 + x_5 + x_6) \cdot (\bar{x}_2 + \bar{x}_3) \cdot (x_2 + x_3 + x_6) \cdot (x_1 + \bar{x}_2 + x_3 + x_4 + x_5 + x_6)$	6	7	T	4,195
$(\bar{x}_5 + \bar{x}_6 + \bar{x}_7) \cdot (x_3 + \bar{x}_4 + x_7) \cdot (\bar{x}_1 + x_3 + x_5 + x_6 + \bar{x}_7) \cdot (x_1 + x_3 + \bar{x}_5 + x_6 + x_7) \cdot (x_2 + x_6) \cdot (x_2 + \bar{x}_6) \cdot \bar{x}_2 \cdot (x_2 + x_3 + x_4 + \bar{x}_5 + x_7)$	7	8	F	10,320
$(\bar{x}_2 + x_5 + x_6 + x_7) \cdot (x_2 + \bar{x}_4 + \bar{x}_5 + \bar{x}_7) \cdot (x_1 + x_2 + \bar{x}_3 + \bar{x}_6 + x_7) \cdot (x_1 + x_2 + x_3 + \bar{x}_5 + x_6 + \bar{x}_7) \cdot (\bar{x}_3 + \bar{x}_5 + x_6 + \bar{x}_7) \cdot (x_1 + x_2 + \bar{x}_3 + \bar{x}_7) \cdot (\bar{x}_1 + x_2 + \bar{x}_4 + \bar{x}_6) \cdot (x_3 + x_5 + x_6 + \bar{x}_7)$	7	8	T	8,862
$(x_3 + x_4 + \bar{x}_6 + \bar{x}_8) \cdot (x_6 + \bar{x}_7) \cdot (\bar{x}_2 + x_3 + \bar{x}_4 + x_5 + x_8) \cdot x_7 \cdot (x_1 + \bar{x}_2 + x_5 + \bar{x}_7 + \bar{x}_8) \cdot (x_2 + x_7 + x_8) \cdot (\bar{x}_6 + \bar{x}_7) \cdot (x_1 + x_5 + \bar{x}_8) \cdot (x_1 + \bar{x}_4 + x_5 + \bar{x}_6 + x_7)$	8	9	F	16,364
$(x_1 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7 + \bar{x}_8) \cdot (x_2 + x_3 + x_4 + \bar{x}_6 + \bar{x}_7 + x_8) \cdot (x_3 + x_4 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7 + \bar{x}_8) \cdot (\bar{x}_1 + \bar{x}_3 + \bar{x}_4 + \bar{x}_5 + x_6 + \bar{x}_7 + x_8) \cdot (\bar{x}_3 + \bar{x}_7) \cdot (x_4 + x_5 + \bar{x}_7) \cdot (x_1 + x_3 + \bar{x}_4) \cdot (x_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_4 + \bar{x}_5 + \bar{x}_6 + \bar{x}_7) \cdot (x_4 + \bar{x}_5 + \bar{x}_6 + x_7 + \bar{x}_8)$	8	9	T	18,856
$(\bar{x}_2 + \bar{x}_3 + x_5 + x_7) \cdot (x_2 + x_5 + x_6 + x_7 + x_9) \cdot (\bar{x}_3 + x_5 + x_7 + x_8) \cdot (x_1 + \bar{x}_4 + \bar{x}_5 + x_6 + x_8) \cdot (\bar{x}_2 + x_3 + x_5 + x_7 + x_8 + \bar{x}_9) \cdot (\bar{x}_2 + \bar{x}_4 + x_7 + x_9) \cdot (\bar{x}_2 + x_4 + x_6 + x_9) \cdot x_1 \cdot x_5 \cdot (\bar{x}_1 + \bar{x}_5)$	9	10	F	34,669
$(x_3 + x_8) \cdot (x_1 + \bar{x}_2 + x_5 + \bar{x}_6 + x_9) \cdot (x_3 + x_6 + x_9) \cdot (x_3 + x_5 + \bar{x}_6 + \bar{x}_8) \cdot (x_1 + x_2 + \bar{x}_5 + x_7 + \bar{x}_8 + \bar{x}_9) \cdot (\bar{x}_1 + x_2 + \bar{x}_4 + x_5 + \bar{x}_6 + \bar{x}_7 + x_9) \cdot (x_1 + x_2 + x_4 + \bar{x}_6 + x_8 + \bar{x}_9) \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4 + x_7 + \bar{x}_8) \cdot (\bar{x}_1 + x_2 + x_3 + x_5 + \bar{x}_6 + x_8 + \bar{x}_9) \cdot (x_2 + \bar{x}_3 + x_4 + \bar{x}_6 + \bar{x}_7 + \bar{x}_9)$	9	10	T	36,450

Clearly, the simulation time of the solution to the SAT problem by using a family of P systems in **CSC**(3) takes much more time than the simulation of the solution by P systems in **CDC**(3). This can be explained by the following fact. In the P systems of **CDC**(3), a replication of objects by means of division rules takes place. Nevertheless, in the P systems of **CSC**(3), there is no replication of objects, but a distribution of them, consequently, in order to generate an exponential amount of some objects, it is necessary to use the skin membrane, interacting with the environment by using antiport rules with length 3 (in a computation step an object is released into the environment and, simultaneously, two objects enter the system).

7. Conclusions and future work

In this paper, an extension of the `P-Lingua` software for the simulation of cell-like P systems with symport/antiport rules and membrane division rules or membrane separation rules is described. A new simulator has been designed and implemented, and included into pLinguaCore library. The functioning of the simulator has been checked by executing two polynomial time solutions to SAT problem by means of two families of cell-like P systems with symport/antiport rules: one of them using membrane division rules and another using membrane separation rules.

It is worth pointing out that the developed simulator is based on sequential technology, so the inherent parallelism of P systems cannot be exploited. This limitation cannot be overcome because a real implementation of P systems does not exist, but some performance improvements can be achieved by means of some parallel architectures and programming models, such as *GPGPU* (General-Purpose Computing on Graphics Processing Units). Specifically, the technology *NVIDIA GPU* with *CUDA* (Compute Unified Device Architecture) has been considered for the development of parallel simulators of P systems such as tissue P systems with cell division [7]. It is still of interest to develop parallel architectures for simulating P systems in the classes **CDC** and **CSC**.

Acknowledgements

The work of L. Pan, T. Song and B. Song was supported by National Natural Science Foundation of China (61033003, 91130034 and 61320106005). The work of M.J. Pérez-Jiménez, L.F. Macías-Ramos and L. Valencia-Cabrera was supported by Project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

References

[1] García-Quismondo, M., Gutiérrez-Escudero, R., del Amor, M. A. M., Orejuela-Pinedo, E. F., Pérez-Hurtado, I.: P-Lingua 2.0: A software framework for cell-like P systems, *International Journal of Computers, Communications and Control*, **IV**, 2009, 234–243.

[2] García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-Lingua 2.0. In: G. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa (eds.) Membrane Computing, *Lecture Notes in Computer Science*, vol. 5957, pp. 264–288. Springer Berlin Heidelberg (2010).

[3] Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems, *Fundamenta Informaticae*, **71**(2-3), 2006, 279–308.

[4] Macías-Ramos, L.F., Song, B., Valencia-Cabrera, L., Pan, L., Pérez-Jiménez, M.J.: Membrane fission: a computational complexity perspective, *Complexity*, in press.

[5] Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P Systems, *Theoretical Computer Science*, **296**(2), 2003, 295–326.

[6] Martínez-del Amor, M. A., Pérez-Hurtado, I., Pérez-Jiménez, M. J., Riscos-Núñez, A.: A P-Lingua based simulator for tissue P systems, *The Journal of Logic and Algebraic Programming*, **79**(6), 2010, 374 – 382.

[7] Martínez-del Amor, M. A., García-Quismondo, M.,Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M. J. Simulating P systems on GPU devices: a survey. *Fundamenta Informaticae*, **136** (3), 2015, 269–284.

[8] Păun, Gh.: Computing with membranes, *Journal of Computer and System Sciences*, **61**, 2000, 108–143.

[9] Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport, *New Generation Computing*, **20**(3), 2002, 295–305.

[10] Păun, A., Păun, Gh., Rozenberg, G.: Computing by communication in networks of membranes, *International Journal of Foundations of Computer Science*, **13**, 2002, 779–798.

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

[12] 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, *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, IEEE, 2010.

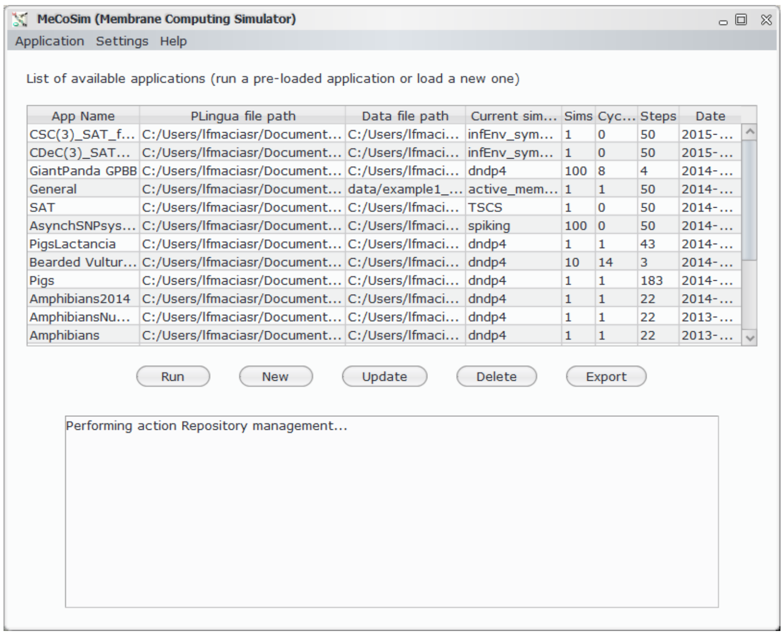
[13] Pérez-Hurtado, I., Valencia-Cabrera, L., Chacón, J. M., Riscos-Núñez, A., Pérez-Jiménez, M. J.: A P-Lingua based simulator for tissue P systems with cell separation, *Romanian Journal of Information Science and Technology*, **17**, 2014, 89–102.

[14] The MeCoSim Web Site: `http://www.p-lingua.org/mecosim/`.

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

A.

Figures Appendix

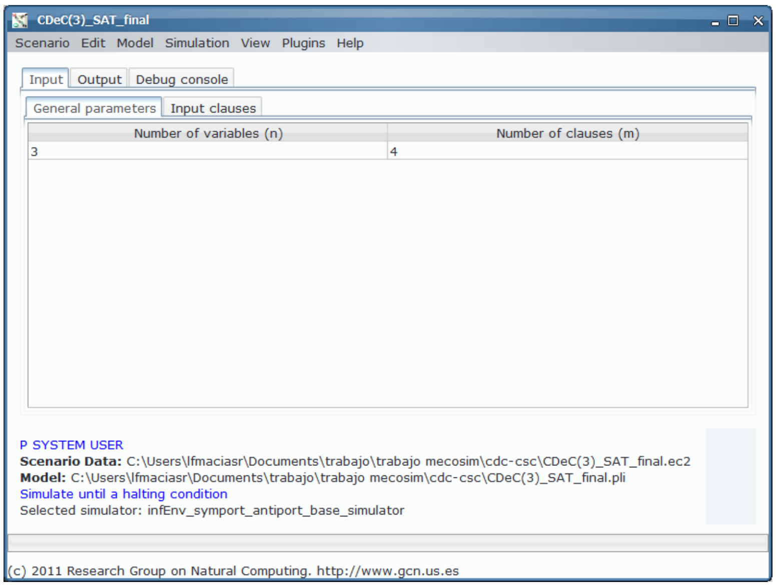

MeCoSim (Membrane Computing Simulator)
Application Settings Help
Run
New
Update
Delete
Export

List of available applications (run a pre-loaded application or load a new one)

App Name	PLingua file path	Data file path	Current sim...	Sims	Cyc...	Steps	Date
CSC(3)_SAT_f...	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	infEnv_sym...	1	0	50	2015-...
CDeC(3)_SAT...	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	infEnv_sym...	1	0	50	2015-...
GiantPanda GPBB	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	100	8	4	2014-...
General	C:/Users/lfmaciasr/Document...	data/example1_...	active_mem...	1	1	50	2014-...
SAT	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	TSCS	1	0	50	2014-...
AsynchSNPsys...	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	spiking	100	0	50	2014-...
PigsLactancia	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	1	1	43	2014-...
Bearded Vultur...	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	10	14	3	2014-...
Pigs	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	1	1	183	2014-...
Amphibians2014	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	1	1	22	2014-...
AmphibiansNu...	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	1	1	22	2013-...
Amphibians	C:/Users/lfmaciasr/Document...	C:/Users/lfmaci...	dndp4	1	1	22	2013-...

Performing action Repository management...

Figure 1. Launching application from MeCoSim main screen

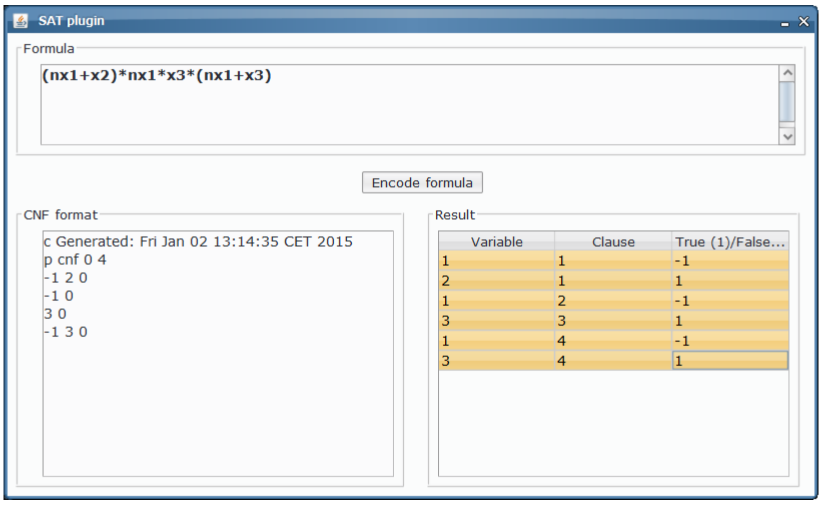

CDeC(3)_SAT_final
Scenario Edit Model Simulation View Plugins Help
Input Output Debug console
General parameters Input clauses

Number of variables (n)	Number of clauses (m)
3	4

P SYSTEM USER
Scenario Data: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.ec2
Model: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.pli
Simulate until a halting condition
Selected simulator: infEnv_symport_antiport_base_simulator

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

Figure 2. Inputting the number of variables and clauses

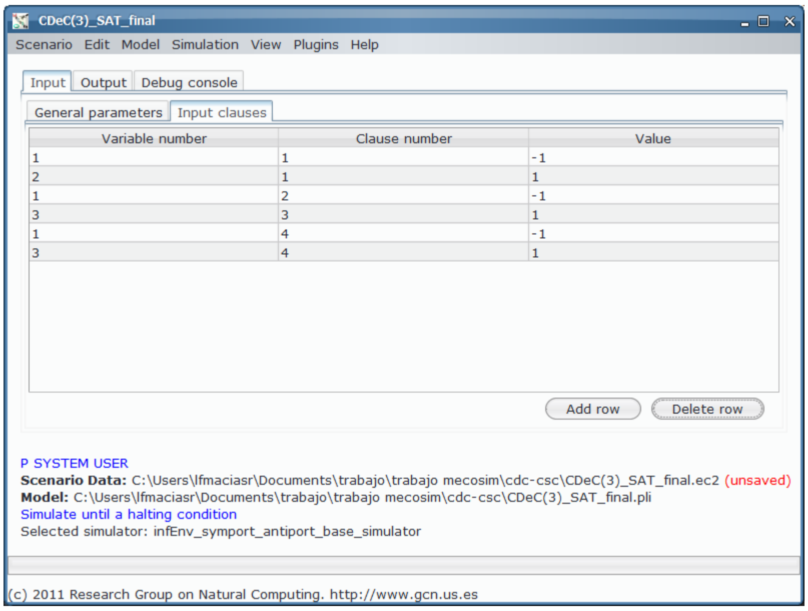


```
(nx1+x2)*nx1*x3*(nx1+x3)
```

```
c Generated: Fri Jan 02 13:14:35 CET 2015
p cnf 0 4
-1 2 0
-1 0
3 0
-1 3 0
```

Variable	Clause	True (1)/False...
1	1	-1
2	1	1
1	2	-1
3	3	1
1	4	-1
3	4	1

Figure 3. Inputting the formula with the formula plugin

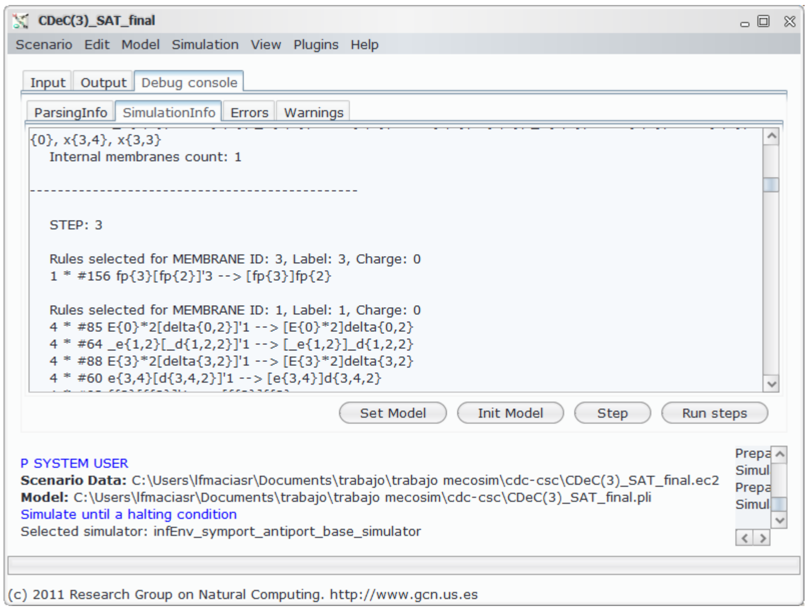

CDeC(3)_SAT_final
Scenario Edit Model Simulation View Plugins Help
Input Output Debug console
General parameters Input clauses
Add row
Delete row

Variable number	Clause number	Value
1	1	-1
2	1	1
1	2	-1
3	3	1
1	4	-1
3	4	1

P SYSTEM USER
Scenario Data: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.ec2 (unsaved)
Model: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.pli
Simulate until a halting condition
Selected simulator: infEnv_symport_antiport_base_simulator

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

Figure 4. Copying the formula to the clauses table


CDeC(3)_SAT_final
Scenario Edit Model Simulation View Plugins Help
Input Output Debug console
ParsingInfo SimulationInfo Errors Warnings

```
{0}, x{3,4}, x{3,3}
   Internal membranes count: 1

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


   STEP: 3

   Rules selected for MEMBRANE ID: 3, Label: 3, Charge: 0
   1 * #156 fp{3}[fp{2}]'3 --> [fp{3}]fp{2}

   Rules selected for MEMBRANE ID: 1, Label: 1, Charge: 0
   4 * #85 E{0}*2[delta{0,2}]'1 --> [E{0}*2]delta{0,2}
   4 * #64 _e{1,2}[_d{1,2,2}]'1 --> [_e{1,2}]_d{1,2,2}
   4 * #88 E{3}*2[delta{3,2}]'1 --> [E{3}*2]delta{3,2}
   4 * #60 e{3,4}[d{3,4,2}]'1 --> [e{3,4}]d{3,4,2}
```

Set Model
Init Model
Step
Run steps

P SYSTEM USER
Scenario Data: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.ec2
Model: C:\Users\lfmaciasr\Documents\trabajo\trabajo mecosim\cdc-csc\CDeC(3)_SAT_final.pli
Simulate until a halting condition
Selected simulator: infEnv_symport_antiport_base_simulator

Prepa
Simul
Prepa
Simul

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

Figure 5. Initializing the model and checking configuration

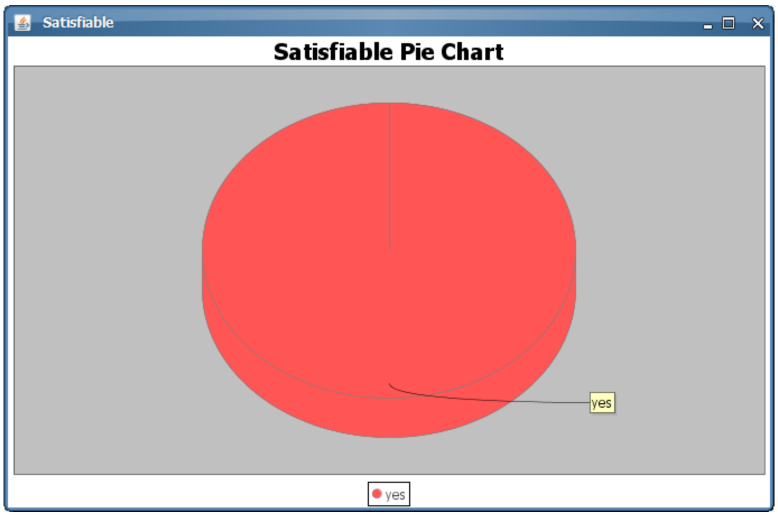


Figure 6. Graphical result check - the chart label will read “yes” or “no” depending of the decision problem result