A Prolog Simulator for Deterministic P Systems with Active Membranes

Andrés CORDÓN-FRANCO, Miguel A. GUTIÉRREZ-NARANJO, Mario J. PÉREZ-JIMÉNEZ, Fernando SANCHO-CAPARRINI

Dpto. de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, España
E-mail: {acordon, magutier, marper, fsancho}@us.es

Abstract. In this paper we propose a new way to represent P systems with active membranes based on Logic Programming techniques. This representation allows us to express the set of rules and the configuration of the P system in each step of the evolution as literals of an appropriate language of first order logic. We provide a Prolog program to simulate the evolution of these P systems and present some auxiliary tools to simulate the evolution of a P system with active membranes using 2-division which solves the SAT problem following the techniques presented in [7].

1 Introduction

In [3] a new model of computation, called *P Systems*, within the framework of *Natural Computing* (bio-inspired computing), was introduced by G. Păun.

It is based upon the notion of *membrane structure* that is used to enclose *computing cells* in order to make them independent computing units. Also, a membrane serves as a communication channel between a given cell and other cells "adjacent" to it. This model comes from the observation that the processes which take place in the complex structure of a living cell can be considered as *computations*.

Since these computing devices were introduced several variants have been considered. A fairly complete compendium about P systems can be found in [2].

The different variants of P systems found in the literature are generally thought as generating devices. Many of them have been proved to be computationally complete: they compute all Turing computable sets of natural numbers or all recursively enumerable languages, depending on the variant considered.

The model we study here, P systems with active membranes, works with symbol–objects, and it provides rules of division for membranes. In particular, *P systems with active membranes* are studied in [2], section 7.2.

The main goal of this paper is to show a representation of P systems with active membranes based on Logic Programming techniques.

The paper is organized as follows: Section 2 briefly presents some ideas about the convenience of using Prolog as base for this representation; Section 3 introduces the way to represent all basic ingredients of this model; Section 4 studies the designed algorithm to simulate deterministic P systems with active membranes; Section 5 presents as an example the solution given to SAT problem using this model in [7]; Section 6 shows a standard work session with the interface provided with the simulator; finally, Section 7 presents some conclusions and future work about the possibilities of this new simulator and representation techniques using Logic Programming.

2 Why Prolog?

Choosing a programming language for an effective implementation of a P system simulator is not an easy decision. The language has to be expressive enough to handle symbolic knowledge in a natural way and the ability of evolving the different configurations following a set of rules. Prolog[1] has both features.

On one hand, the based-tree data structure and the use of infix operators defined *ad hoc* by the programmer allow us to *simulate* the natural language and the user can follow the evolution of the system without any knowledge of Prolog (see sections 3.2 and 3.3). On the other hand, Prolog programs are sets of facts and rules and basic mechanisms of Prolog are pattern matching and automatic backtracking, so the design of the inference engine to do the evolutions has a natural treatment from a programmer point of view.

Finally, the use of Prolog as language programming has other desirable side effects which are out of the scope of this paper. Prolog fits into all kind of symbolic reasoning and the use of this representation can be a way to link P systems to other deeply studied fields in Artificial Intelligence.

3 A Logic Programming representation of P systems with active membrane

Let us remember that a *P system with active membranes* is a construct:

$$\Pi = (V, H, \mu, w_1, \ldots, w_m, R)$$

where:

1. $m \geq 1$, is the initial *degree* of the system;

2. V is the alphabet of *symbol-objects*;

3. H is a finite set of *labels* for membranes;

4. μ is a *membrane structure*, of m membranes, labelled (not necessarily in a one-to-one manner) with elements of H;

5. $w_1, \ldots, w_m$ are strings over V, describing the initial *multisets* of objects placed in the m regions of μ;

6. R is a finite set of *evolution rules*, of the following forms:

   (a) $[_h x \rightarrow y]_h^\alpha$, for $h \in H$, $\alpha \in \{+, -, 0\}$, $x \in V$, $y \in V^*$. This is an object evolution rule, associated with a membrane labelled with h and depending on the polarity of that membrane, but not directly involving the membrane.

[1]A good starting point can be [1] or [9].

(b) $x[_h\ ]_h^{\alpha_1} \rightarrow [_h y]_h^{\alpha_2}$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $x, y \in V$. An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.

(c) $[_h x]_h^{\alpha_1} \rightarrow [_h\ ]_h^{\alpha_2} y$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $x, y \in V$. An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.

(d) $[_h x]_h^{\alpha} \rightarrow y$, for $h \in H$, $\alpha \in \{+, -, 0\}$, $x, y \in V$. A membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.

(e) $[_h x]_h^{\alpha_1} \rightarrow [_h y]_h^{\alpha_2} [_h z]_h^{\alpha_3}$, for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $x, y, z \in V$. A membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object which can evolve by one rule of any form, should evolve.

- If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.

- All objects and membranes not specified in a rule and which do not evolve remain unchanged to the next step.

- If at the same time a membrane h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.

- The rules associated with membranes labelled with h are used for all copies of this membrane. At one step, a membrane labelled with h can be the subject of *only* one rule of types (b)-(e).

In order to give a formal representation in Prolog of the basic structures of P systems with active membranes using 2-division, the following representation will be considered.

3.1 Representation of membrane structures

A given configuration will be expressed by mean of a labelled tree, where:

1. $< >$ is the *position* to denote the root of the tree and it will be associated to the skin;

2. if $< i_1, \ldots, i_n >$ is the position of a membrane h, then $< i, i_1, \ldots, i_n >$ will denote the *position* of the i-th inner membrane to h.

There exists one difference between the above representation and the one we use in Prolog:

In our Prolog representation if in one step of the computation a membrane with label $< i_1, \ldots, i_n >$ has k inner membranes, its children do not have to be labelled with $< 1, i_1, \ldots, i_n >$, $< 2, i_1, \ldots, i_n >$, $\ldots$, $< k-1, i_1, \ldots, i_n >$, $< k, i_1, \ldots, i_n >$.

This is because if one child membrane is dissolved the other ones are not re-labelled and. Besides, new membranes obtained by division are labelled with new indexes, not by filling the holes of previously dissolved membranes.

3.2 Representation of configurations

Let us remember that to give a configuration for a P system with active membranes consists in making explicit the membrane structure and the content of every membrane present in this structure.

In our model we will represent the configuration in one step of the evolution as a set of one-literal clauses, each of them representing each alive membrane. Hence, in this representation each clause will show label, position, polarity, multiset of objects and current step of the computation, as well as the P system this membrane belongs to. In this way, the set of clauses give information about the contents of the membranes and the membrane structure (by means of the position of each one).

In a general form, to denote that in the t-th step of its evolution the P system, P, has a membrane at position $[pos]$ with label h, polarity α and m as multiset, we will write

$$P \texttt{ :: } h \texttt{ ec } \alpha \texttt{ at } [pos] \texttt{ with } m \texttt{ at\_time } t$$

Note that we can use the user-friendly representation of a Prolog literal, instead of the functional representation.

In a general way, if $m = \{\{x_1, \ldots, x_n\}\}$ (with not necessarily $x_i \neq x_j$), then we will denote $m = [x_1, \ldots, x_n]$.

3.3 Description of the rules

By means of some new function symbols, the rules are also represented as literals. Following we present the general form of the different rules showed above:

(a) $[_h x \to y]_h^\alpha$

P `rule` x `evolves_to` $[y]$ `in` h `ec` α

(b) $x[_h\ ]_h^{\alpha_1} \to [_h y]_h^{\alpha_2}$

P `rule` x `out_of` h `ec` α_1 `sends_in` y `of` h `ec` α_2

(c) $[_h x]_h^{\alpha_1} \to [_h\ ]_h^{\alpha_2} y$

P `rule` x `inside_of` h `ec` α_1 `sends_out` y `of` h `ec` α_2

(d) $[_h x]_h^\alpha \to y$

P `rule` x `inside_of` h `ec` α `dissolves_and_sends_out` y

(e) $[_h x]_h^{\alpha_1} \to [_h y]_h^{\alpha_2} [_h z]_h^{\alpha_3}$

P `rule` x `inside_of` h `ec` α_1 `divides_into` y `inside_of` h `ec` α_2
`and` z `inside_of` h `ec` α_3

4 The algorithm

The Prolog algorithm to simulate a P system works in a natural way. The input of the program is the initial configuration of the membranes (which is represented as a set of literals with predicate symbol `**`, all of them `at_time 0`) and an appropriate set of rules.

- **Step 1: Initialization**. At the beginning, all the membranes are set to *applicable* and their objects are split into two multiset: one **usable** multiset, containing all the objects of the initial membrane, and one **used** multiset which is empty.
- **Step 2: Transition**. If there exists an applicable membrane satisfying the condition of a rule, then the rule is applied in the following way:
  - **(a)-step:** At this stage, only rules of type (a) are checked. The object which shoots the rule is removed from the **usable** and the result multiset by the application of the rule is added to **used**, to prevent one object can be used by two different rules at the same step. After the *evolution* step, the membrane remains to be *applicable* and new evolution rules can be applied.

    This stage ends when no more rules of type (a) can be applied.
  - **Non-(a)-step**: At this stage, only one rule of the other types (not (a)) can be applied. Let us remember that this simulation only works with deterministic P systems (in fact, it works with *confluent* ones). The action depends on the kind of rule to apply:
    - **Send out rule**: The element which shoots the rule is removed from **usable** multiset and the new one is added to the **used** multiset of the father membrane. Both membranes changes to *not applicable* mode. If the element is sent out of the skin, then it is marked with the property **outside**.
    - **Send in rule**: It is the reciprocal of the previous one. The element which shoots the rule is removed from **usable** in the father membrane and the new one is added to the **used** multiset. Both membranes changes to *not applicable* mode.
    - **Dissolution rule**: The element which shoots the rule belongs to **usable** and the new element obtained together the rest of the elements of the membrane are added to the **used** multiset of the father membrane. The membranes inside of the dissolved membranes become to be children of the father one and their positions are arranged to be correct. The father membrane changes to *not applicable* mode.
    - **Division rule**: The element which shoots the rule belongs to **usable** and the division creates two new membranes in *not applicable* mode. One of them keeps the original position and the second one gets a position which has not been occupied by any membrane.
  - **End**: When no more rules can be applied to *applicable* membranes, a new configuration (with `at_time` incremented in 1) is stored. In this moment no membrane has *applicable* or *not applicable* state. These modes only have validity during the evolution. At this stage, the P system is ready for a new step of the evolution.
- **Step 3: End of computation**. The evolution of the P system finishes when there are no rules to be applied.

5 A study case: SAT problem

5.1 General solution to SAT by P systems with active membranes

Propositional Satisfiability is the problem of determining, for a formula of the propositional calculus, if there is an assignment of truth values to its variables for which that formula evaluates to true. By **SAT** we mean the problem of propositional satisfiability for formulas in conjunctive normal form (CNF).

Following [7] we present a family of recognizers P systems with active membranes using 2–division (see [2], section 7.2) solving the **SAT** problem in linear time.
Let us suppose that $\varphi = C_1 \wedge \cdots \wedge C_m$ in CNF, and $Var(\varphi) = \{x_1, \dots, x_n\}$.
For each $(m, n) \in \mathbf{N}^2$ we consider the recognizer P system

$$(\Pi(\langle m, n\rangle), \Sigma(m, n), i(m, n))$$

where $\Sigma(m, n) = \{x_{i,j}, \overline{x}_{i,j} : \ 1 \leq i \leq m, \ 1 \leq j \leq n\}$, $i(m, n) = 2$ and $\Pi(\langle n, m\rangle) = (\Gamma(m, n), \{1, 2\}, [_1 \ [_2 \ ]_2 \ ]_1, w_1, w_2, R)$ is defined as follows:
$\Gamma(m, n) = \Sigma(m, n) \ \cup \{c_k : \ 1 \leq k \leq m + 2\} \ \cup \{d_k : \ 1 \leq k \leq 3n + 2m + 3\} \ \cup$
$\{r_{i,k} : \ 0 \leq i \leq m, 1 \leq k \leq 2n\} \ \cup \{e, t\} \ \cup \{Yes, No\}$
We will say that every internal membrane with label 2 is an *internal membrane*.
The initial content of each membrane is: $w_1 = \emptyset$ y $w_2 = \{d_1\}$.
The set of rules, R, is given by:

(a) $\{[_2 d_k]_2^0 \to [_2 d_k]_2^+ [_2 d_k]_2^- \ : \ 1 \leq k \leq n\}$.

By using a rule of (a), a membrane with label 2 is divided into two membranes with the same label, but with different polarizations. These rules allow us to duplicate, in one step, the total number of internal membranes.

(b) $\{[_2 x_{i,1} \to r_{i,1}]_2^+, \ [_2 \ \overline{x}_{i,1} \to r_{i,1}]_2^- \ : \ 1 \leq i \leq m\}$
$\{[_2 x_{i,1} \to \lambda]_2^-, \ [_2 \ \overline{x}_{i,1} \to \lambda]_2^+ \ : \ 1 \leq i \leq m\}$

The rules of (b) try to implement a process allowing to the internal membranes to encode the *assignment* of a variable and, simultaneously, to check the value of all clauses by this assignment, in such a way that if the clause is true then an object $r_{i,1}$ will appear in the membrane. In other case, the object encoding the variable will disappear.

(c) $\{[_2 x_{i,j} \to x_{i,j-1}]_2^+, \ [_2 x_{i,j} \to x_{i,j-1}]_2^- \ : \ 1 \leq i \leq m, \ 2 \leq j \leq n\}$
$\{[_2 \ \overline{x}_{i,j} \to \ \overline{x}_{i,j-1}]_2^+, \ [_2 \ \overline{x}_{i,j} \to \ \overline{x}_{i,j-1}]_2^- \ : \ 1 \leq i \leq m, \ 2 \leq j \leq n\}$

The check process described previously is always made with respect to the *first* variable appearing in the internal membrane. Hence, the rules of (c) take charge of making a cyclic path through all the variables to get that, initially, the first variable is x_1, then x_2, and so on.

(d) $\{[_2 d_k]_2^+ \to [_2 \ ]_2^0 d_k, \ , \ [_2 d_k]_2^- \to [_2 \ ]_2^0 d_k : \ 1 \leq k \leq n\}$.
$\{ \ d_k [_2 \ ]_2^0 \to [_2 d_{k+1}]_2^0 : \ 1 \leq k \leq n - 1\}$.

The rules of (d) are used as controllers of the generating process of the assignments and the encoding of the satisfied clauses: the objects d are sent out to the skin at the same time the checking is made and they come back to the internal membranes to start the division of these membranes.

(e) $\{[_2 r_{i,k} \to r_{i,k+1}]_2^0 : \ 1 \le i \le m, \ 1 \le k \le 2n-1\}$.

The use of objects r in the rules (i), (j) and (k) makes necessary to perform a rotation of these objects. This is the mission of the rules of (e).

(f) $\{[_1 d_k \to d_{k+1}]_1^0 : \ n \le k \le 3n-3\}$; $[_1 d_{3n-2} \to d_{3n-1} e_0]_1^0$.

Through the counter-objects d, the rules of (f) *control* the rotation of the elements $r_{i,k}$ in the internal membranes.

(g) $e[_2 \ ]_2^0 \to [_2 c_1]_2^+$; $[_1 d_{3n-1} \to d_{3n}]_1^0$.

The application of the rules of (g) will show that the system is ready to check which clauses are true by the assignment encoded by an internal membrane.

(h) $\{[_1 d_k \to d_{k+1}]_1^0 : \ 3n \le k \le 3n+2m+2\}$.

The rules of (h) supply counters in the skin through objects d; in such a way that if objects d_{3n+2m} appear then it shows the end of the checking of the clauses. The objects d_k, with $3n+2m+1 \le k \le 3n+2m+3$, will control the final stage of the computation.

(i) $[_2 r_{1,2n}]_2^+ \to [_2 \ ]_2^- r_{1,2n}$.

The applicability of the rule (i) encodes the fact that an internal membrane makes true the clause *represented* by the object $r_{1,2n}$, through a change in the sign of its polarization. Because of this, we must relabel the values of r representing the different internal membranes. This is made by means of the rules (j).

(j) $\{[_2 r_{i,2n} \to r_{i-1,2n}]_2^- : \ 1 \le i \le m\}$.

(k) $r_{1,2n}[_2 \ ]_2^- \to [_2 r_{0,2n}]_2^+$.

By using the rule (k) the task of making explicit the assignments that make true the clause encoded in that moment of the execution by the object $r_{1,2n}$ is ended.

(l) $\{[_2 c_k \to c_{k+1}]_2^- : \ 1 \le k \le m\}$.

The presence of objects c_k (with $2 \le k \le m+1$) in the internal membranes shows that the assignments making true every clauses are being determined.

(m) $[_2 c_{m+1}]_2^+ \to [_2 \ ]_2^+ c_{m+1}$.

The rule (m) sends to skin the objects c_{m+1} appearing in the internal membranes.

(n) $[_1 c_{m+1} \to c_{m+2} t]_1^0$.

By using the rule (n) the objects c_{m+1} in the skin evolve to objects $c_{m+2}t$. The objects t in the skin are produced simultaneously with the appearance of the objects $d_{3n+2m+2}$ in the skin, and will show that there exists some assignment making true the formula.

(*o*) $[_1 t\ ]_1^0 \rightarrow [_1\ ]_1^+ t.$

The rule (*o*) sends out of the system an object t changing the polarization of the skin to positive (then no could evolve the objects t remaining in the skin). Hence, an object c_{m+2} can exit the skin producing an object Yes. This object is then sent to the environment through the rule (*p*), telling us that the formula is satisfiable, and the computation stops.

(*p*) $[_1 c_{m+2}]_1^+ \rightarrow [_1\ ]_1^- Yes.$

The applicability of the rule (*p*) changes the polarization in the skin to negative in order that the objects c_{m+2} remaining in it could not continue evolving.

(*q*) $[_1 d_{3n+2m+3}]_1^0 \rightarrow [_1\ ]_1^+ No.$

By the rule (*q*) the object $d_{3n+2m+3}$ only evolves when the skin has a neutral charge (this is the case when the formula is not satisfiable). Then the system will evolve sending to the environment an object No and changing the polarization of the skin to positive, in order that objects $d_{3n+2m+3}$ remaining in the skin, could not evolve.

In [7] it is proved that the family $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbf{N}^+}$ solves the **SAT** problem in linear time. As input data for this P system, we consider the multiset

$$\{x_{i,j} :\ x_j \in C_i\} \cup \{\overline{x}_{i,j} :\ \ \neg x_j \in C_i\}$$

The execution of the P system with above input can be structured in four stages: a stage of *generation* of all assignments; a stage of *synchronization*; a stage of *checking* the assignments with regard to the formula; and a stage of *output*.
The *generating stage* is controlled by the objects d_i, with $1 \leq i \leq n$.

- The presence in the skin of one object d_i, with $1 \leq i \leq n$, will show that all possible partial assignments associated with $\{x_1, \ldots, x_i\}$ have been generated.
- In this stage, simultaneously to the consideration of partial assignments (each one associated with each internal membrane created by division) we will encode in every internal membrane all the clauses being true by the assignment *represented* by the membrane (through the objects $r_{i,k}$).
- The object d_1 appears in the skin after the execution of 2 steps. From the appearance of d_i in the skin to the appearance of d_{i+1}, with $1 \leq i \leq n-1$, 3 steps have been executed.
- This stage ends when the object d_n appears in the skin.

Hence, the total number of steps in the generating stage is $3n - 1$.
The *synchronization stage* has the goal of unifying the second subindexes of the objects $r_{i,k}$, to make them equal to $2n$.

- This stage starts with the evolution of the object d_n in the skin.
- In every step of this stage the object d_i, with $n \leq i \leq 3n-1$, of the skin evolves to d_{i+1}.

- This stage ends as soon as the object d_{3n} appears in the skin, that is the moment when each internal membrane has positive charge and contains one object c_1 (obtained by using the first rule of (g)).

Therefore, the synchronization stage needs a total of $2n$ steps.
The *checking stage* has the goal to determine how many (and which) clauses are *true* in every internal membrane (that is, by the assignment *represented* by it). This stage is controlled by the objects c_i, with $1 \leq i \leq m+1$, and it starts after the presence of c_1 in the internal membranes.

- The presence of an object c_i in an internal membrane shows that the clauses $C_1, \ldots, C_{i-1}$ are true by the assignment represented by such membrane.
- From every c_i (with $1 \leq i \leq m$) the object c_{i+1} is obtained in *some* membranes after the execution of 2 steps.
- The checking stage ends as soon as the object d_{3n+2m} appears in the skin.

Therefore, the total number of steps of this stage is $2m$.
The *output stage* starts immediately after the appearance of the object d_{3n+2m} in the skin and it is controlled by the objects c_{m+1} and c_{m+2}.

- To produce the output Yes the object c_{m+1} must have been produced in some internal membrane of the configuration $\mathcal{C}_{5n+2m-1}$. Then, after 4 steps the system returns Yes to the environment, through the evolution of objects c_{m+2} present in the skin, and when it has positive charge.
- To produce the output No, no object c_{m+1} appears in any internal membrane to the configuration $\mathcal{C}_{5n+2m-1}$. Then after 4 steps the system returns No to the environment, through the evolution of objects $d_{3n+2m+3}$ present in the skin, and when it has neutral charge.

Therefore, the total number of steps in the output stage is 4.

5.2 A Prolog session for $m = n = 2$

In this section we show a session for one instance of the problem. We consider two clauses: $C_1 = x_1 \wedge x_2$ and $C_2 = x_1 \wedge \neg x_2$. First, we will build the initial configuration and the set of rules[2].

The initial configuration has two membranes: the skin at position $<>$, label `e1`, electrical charge `0` and empty content and an inner membrane. The inner membrane contains the information relative to this instance of the problem: `xi_j` represents that x_j is a variable in C_i and `zi_j` represents that $\neg x_j$ occurs in C_j. So, we consider an inner membrane at position $< 1 >$ whit label `e2`, electric charge 0 and the multiset codifying the information of the literals. We call `p1` to this P system and we write `at_time 0` to denote that P system has not evolved yet.

The Prolog representation of this initial configuration is

```
p1 :: e1 ec 0 at [] with [] at_time 0.
p1 :: e2 ec 0 at [1] with [x2_1, z2_2, x1_1, x1_2, d1] at_time 0.
```

[2]The Prolog simulator provides the facilities to build automatically the initial configuration and the set of rules for any instance of the SAT problem.

The set of rules only depends on the set of clauses and the set of variables and it is generated by the simulator. The set of rules are numbered. We have 52 rules in this example[3].

```
% Set (a)

p1 rule d1 inside_of e2 ec 0 divides_into d1 inside_of
e2 ec 1 and d1 inside_of e2 ec-1 ** 1.

p1 rule d2 inside_of e2 ec 0 divides_into d2 inside_of
e2 ec 1 and d2 inside_of e2 ec-1 ** 2.

% Set (b)

p1 rule x1_1 evolves_to [r1_1]in e2 ec 1 ** 3.
p1 rule z1_1 evolves_to [r1_1]in e2 ec-1 ** 4.
   ...

% Set (c)

p1 rule x1_2 evolves_to [x1_1]in e2 ec 1 ** 11.
p1 rule x1_2 evolves_to [x1_1]in e2 ec-1 ** 12.
p1 rule z1_2 evolves_to [z1_1]in e2 ec 1 ** 13.
   ...

% Set (d)

p1 rule d1 inside_of e2 ec  1 sends_out d1 of e2 ec 0 ** 19.
p1 rule d1 inside_of e2 ec -1 sends_out d1 of e2 ec 0 ** 20.
   ...

% Set (e)

p1 rule r1_1 evolves_to [r1_2] in e2 ec 0 ** 24.
p1 rule r1_2 evolves_to [r1_3] in e2 ec 0 ** 25.
   ...

% Set (f)

p1 rule d2 evolves_to [d3] in e1 ec 0 ** 30.
p1 rule d3 evolves_to [d4] in e1 ec 0 ** 31.

p1 rule d4 evolves_to [d5, e] in e1 ec 0 ** 32.


% Set (g)
```

[3] The number after ** is the ordinal

```
p1 rule e out_of e2 ec 0 sends_in c1 into e2 ec 1 ** 33.
p1 rule d5 evolves_to[d6]in e1 ec 0 ** 34.

% Set (h)

p1 rule d6 evolves_to [d7] in e1 ec 0 ** 35.
p1 rule d7 evolves_to [d8] in e1 ec 0 ** 36.
   ...
p1 rule d12 evolves_to [d13]in e1 ec 0 ** 41.

% Set (i)

p1 rule r1_4 inside_of e2 ec 1 sends_out r1_4 of e2 ec-1 ** 42.

% Set (j)

p1 rule r1_4 evolves_to [r0_4] in e2 ec -1 ** 43.
p1 rule r2_4 evolves_to [r1_4] in e2 ec -1 ** 44.

% Set (k)

p1 rule r1_4 out_of e2 ec-1 sends_in r0_4 into e2 ec 1 ** 45.

% Set (l)

p1 rule c1 evolves_to [c2] in e2 ec -1 ** 46.
p1 rule c2 evolves_to [c3] in e2 ec -1 ** 47.

% Set (m)

p1 rule c3 inside_of e2 ec 1 sends_out c3 of e2 ec 1 ** 48.

% Set (n)

p1 rule c3 evolves_to [c4, t] in e1 ec 0 ** 49.

% Set (o)

p1 rule t inside_of e1 ec 0 sends_out t of e1 ec 1 ** 50.

% Set (p)

p1 rule c4 inside_of e1 ec 1 sends_out yes of e1 ec-1 ** 51.

% Set (q)

p1 rule d13 inside_of e1 ec 0 sends_out no of e1 ec 1 ** 52.
```

To start with the simulation of the evolution of the P system `p1` from the time 0 we type the following command.

```
?- evolve(p1,0).
```

The simulator returns the configuration at time 1, the set of rules and additional information about the elements outside the skin.

```
p1 :: e1 ec  0 at [] with [] at_time 1
p1 :: e2 ec -1 at [2] with [x2_1,z2_2, x1_1, x1_2, d1] at_time 1
p1 :: e2 ec  1 at [1] with [x2_1,z2_2, x1_1, x1_2, d1] at_time 1

Used rules in the step 0: [1]
```

In this step only the first rule has been applied. We follow the evolution.

```
?- evolve(p1,1).

p1 :: e1 ec 0 at [] with [d1, d1] at_time 2
p1 :: e2 ec 0 at [1] with [r1_1, r2_1, x1_1, z2_1] at_time 2
p1 :: e2 ec 0 at [2] with [x1_1, z2_1] at_time 2

Used rules in the step 1: [3, 5, 7, 9, 11, 12, 17, 18, 19, 20]
```

where 10 rules have been applied.

The simulator also allows us to go to a configuration at time `N` without showing the previous steps.

```
?- configuration(p1,5).

p1 :: e1 ec 0 at [] with [d2, d2, d2, d2] at_time 5
p1 :: e2 ec 0 at [1] with [r1_3, r2_3, r1_1] at_time 5
p1 :: e2 ec 0 at [2] with [r1_1] at_time 5
p1 :: e2 ec 0 at [3] with [r2_1] at_time 5
p1 :: e2 ec 0 at [4] with [r1_3, r2_3, r2_1] at_time 5
```

In this example the generating stage ends at time 5. We can follow the evolution and obtain the configuration at time 9. This is the first configuration where the element `d6` appears in the skin.

```
?- configuration(p1,9).

p1 :: e1 ec 0 at [] with [d6, d6, d6, d6] at_time 9
p1 :: e2 ec 1 at [1] with [r1_4, r2_4, r1_4, c1] at_time 9
p1 :: e2 ec 1 at [2] with [r1_4, c1] at_time 9
p1 :: e2 ec 1 at [3] with [r2_4, c1] at_time 9
p1 :: e2 ec 1 at [4] with [r1_4, r2_4, r2_4, c1] at_time 9
```

The synchronization stage has ended. In the next step begins the checking stage which ends at time 13 when the element `d10` appears in the skin.

```
?- configuration(p1,13).

p1 :: e1 ec 0 at [] with [d10, d10, d10, d10] at_time 13
```

```
p1 :: e2 ec 1 at [1] with [r0_4, r0_4, r0_4, c3] at_time 13
p1 :: e2 ec 1 at [2] with [r0_4, c2] at_time 13
p1 :: e2 ec 1 at [3] with [r2_4, c1] at_time 13
p1 :: e2 ec 1 at [4] with [r0_4, r0_4, r0_4, c3] at_time 13
```

After that, the output stage starts. At the step 16 the element `t` is sent out of the skin.

```
?- evolve(p1,15).

p1 :: e1 ec 1 at [] with [c4, c4, t, d13, d13, d13, d13] at_time 16
p1 :: e2 ec 1 at [1] with [r0_4, r0_4, r0_4] at_time 16
p1 :: e2 ec 1 at [2] with [r0_4, c2] at_time 16
p1 :: e2 ec 1 at [3] with [r2_4, c1] at_time 16
p1 :: e2 ec 1 at [4] with [r0_4, r0_4, r0_4] at_time 16

Used rules in the step 15: [41, 41, 41, 41, 50]

outside(t).
```

Finally, in the next step the system sends out `yes`

```
?- evolve(p1,16).

p1 :: e1 ec -1 at [] with [c4, t, d13, d13, d13, d13] at_time 17
p1 :: e2 ec  1 at [1] with [r0_4, r0_4, r0_4] at_time 17
p1 :: e2 ec  1 at [2] with [r0_4, c2] at_time 17
p1 :: e2 ec  1 at [3] with [r2_4, c1] at_time 17
p1 :: e2 ec  1 at [4] with [r0_4, r0_4, r0_4] at_time 17

Used rules in the step 16: [51]

outside(t).
outside(yes).
```

To check the system we try to evolve one more time, though this is a halting configuration.

```
?- evolve(p1,17).
No more evolution!
The P-system p1 has already reached a halting configuration at
 step 17
```

6 Conclusions and future work

We have presented an *effective* tool for making experiments with P systems to solve concrete problems. One example is the problem SAT, but the simulator is for general purpose, covering all the deterministic P systems with active membranes using 2-division.

The chosen logic representation is so user-friendly that the user can follow the evolution of the system without any knowledge about Prolog language. Nevertheless, the representation of rules and membranes as literals in a clausal language allows a natural treatment of the objects from a programmer's point of view.

The idea of representing the P systems as an unordered set of clauses endows them with a flexibility which can be used for new purposes. This can be a good starting point towards new unexplored perspectives in P systems, such as the merge of two P systems or the evolution of a population of P systems in the way of genetic algorithms can be.

Representing P systems as Prolog programs suggests new relations between P systems and other deeply studied techniques in Artificial Intelligence. Different aspects related to heuristics, search in spaces of states or learning can be linked to the theory of P systems via this representation.

Acknowledgement

Research supported by project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

[1] Bratko, I.: *PROLOG Programming for Artificial Intelligence*, Third edition. Addison-Wesley, 2001.

[2] Păun, G.: *Membrane Computing. An Introduction*, Springer-Verlag, Berln, 2002.

[3] Păun, G.: Computing with membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108–143.

[4] Păun, G.; Rozenberg, G.: A guide to membrane computing, *Theoretical Computer Sciences*, 287, 2002, 73–100.

[5] Păun, G.; Rozenberg, G.; Salomaa, A.: Membrane computing with external output, *Fundamenta Informaticae*, **41**(3), 2000, 313–340.

[6] Pérez Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: *Teoría de la Complejidad en modelos de computación celular con membranas*, Editorial Kronos, Sevilla, 2002.

[7] Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: The polynomial complexity class with active membranes. Submitted.

[8] The P Systems Web Page: `http://psystems.disco.unimib.it/`

[9] Logic Programming: `http://www.afm.sbu.ac.uk/logic-prog/`