A CLIPS Simulator for Recognizer P Systems with Active Membranes

Mario PÉREZ-JIMÉNEZ, Francisco José ROMERO-CAMPERO

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {`Mario.Perez, Francisco-Jose.Romero`}`@cs.us.es`

Abstract. In this paper we propose a new way to represent recognizer P systems with active membranes based on Production Systems techniques. This representation allows us to express the set of rules and the configurations in each step of the evolution as facts in a knowledge base. We provide a CLIPS program to simulate the evolutions of this variant of P systems.

1 Introduction

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

This model starts from the observation that the processes which take place in the complex structure of a living cell can be considered as computations. 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.

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, recognizer P systems with active membranes (a recognizing language device), 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 Production Systems techniques.

Production Systems were first introduced by Post in 1943 in order to describe rewrite rules for symbol strings and are nowadays used as the basis for many rule-based systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb", which specify a set of actions to be performed for a given situation. A rule is composed of a *left hand side portion* (if part) and a *right hand side portion* (then part). The LHS of a rule is a series of patterns which specify the facts (or data) which cause the rule to be applicable. The process of matching facts to patterns is called pattern matching. There exists a mechanism, called the *inference engine*, which automatically matches facts against patterns and determines which rules are applicable. The RHS of a rule is the set of actions to be executed when the rule is applicable. The actions of applicable rules are executed when the inference engine is instructed to begin execution. By using a *resolution strategy*, the inference engine selects a rule and then the actions of the selected rule are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

The paper is organized as follows: Section 2 briefly presents some ideas about the analogies between P systems and Production Systems. Section 3 recalls recognizer P systems with active membranes, input membrane and external output. Section 4 introduces CLIPS as a programming language used in Production Systems. Section 5 introduces the way to represent all basic ingredients of this variant of P systems. Section 6 studies the designed algorithm to simulate recognizer P systems and Section 7 presents a session with the simulator. Section 8 presents some conclusions.

2 P Systems versus Production Systems

In this section we will discuss some analogies between Production Systems and P systems. Basically, a Production System consists in tree components:

- *Working Memory*: A set of "facts" consisting of positive literals defining what's known to be true about the world.
- *Rules*: An unordered set of user-defined "if-then" rules of the form:

  $$if\ \ P_1 \wedge ... \wedge P_m\ \ then\ \ Action_1, ..., Action_n$$

  where the P_is are facts that determine the conditions when this rule is applicable. Each Action adds or deletes a fact from the Working Memory.
- *Inference Engine*: Procedure for inferring changes (additions and deletions) to Working Memory.

Usually a cycle of three phases takes place: match, conflict resolution, and action (in that order).

1. *Match the current Working Memory with the rule-base (Rete algorithm)*: Construct Conflict Set, the set of all possible (rule,list-of-facts) pairs such that rule is one of the rules and list-of-facts is a subset of facts in Working Memory that unify with the antecedent part (i.e., Left-hand side) of the given rule.

```
If Conflict Set is empty then
Halt
  else
Go to 2
```

2. *Conflict Resolution*: Instead of trying all applicable rules in the Conflict set, select one for execution using a conflict resolution strategy.

3. *Act/Fire*: Execute the actions associated with the RHS part of the selected rule, after making the substitutions used during unification of LHS part with the list-of-facts.

In P systems we have rules that are applicable only under certain conditions (as in Production systems). The application of a rule in P systems implies the evolution of elements and/or membranes, that is a change in the current "configuration of the world". This is the case in Production Systems where the application of a rule implies changes in the Working Memory. P systems are in general non deterministic, so they have some kind of mechanism, similar to the conflict resolution phase in Production Systems, that chooses which rules from the set of applicable rules are actually applied.

Because of all the analogies mentioned above it seems natural to think of P systems as rule-based systems within the framework of Production Systems and to try to represent and simulate them in a straightforward way using Production Systems techniques.

3 Recognizer P systems with Active Membranes, Input Membrane and External Output

Recall that a decision problem, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

Definition 3.1 *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where:*

- Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \dots, p$*, and initial multisets* $\mathcal{M}_1, \dots, \mathcal{M}_p$ *associated with them.*

- Σ *is an (input) alphabet strictly contained in* Γ.

- *The initial multisets are over* $\Gamma - \Sigma$.

- i_Π *is the label of a distinguished (input) membrane.*

Definition 3.2 *Let* (Π, Σ, i_Π) *be a P system with input. Let* Γ *be the working alphabet of* Π*,* μ *the membrane structure and* $\mathcal{M}_1, \dots, \mathcal{M}_p$ *the initial multisets of* Π*. Let* m *be a multiset over* Σ*. The* initial configuration of (Π, Σ, i_Π) with input m *is* (μ_0, M_0)*, where* $\mu_0 = \mu$*,* $M_0(j) = \mathcal{M}_j$*, for each* $j \neq i_\Pi$*, and* $M_0(i_\Pi) = \mathcal{M}_{i_\Pi} \cup m$.

The computations of a P system with input $m \in M(\Sigma)$, a multiset over Σ, are defined in a natural way. The only novelty is that the initial configuration must be the initial configuration of the system associated with the input multiset $m \in M(\Sigma)$.

In the case of P systems with input and with external output, the concept of computation is introduced in a similar way but with a small change. In the configurations, we will not work directly with the membrane structure μ but with another structure associated with it including, in some sense, the environment.

Definition 3.3 *Let $\mu = (V(\mu), E(\mu))$ be a membrane structure. The* membrane structure with environment *associated with μ is the rooted tree $Ext(\mu)$ such that: (a) the root of the tree is a new node that we will denote env; (b) the set of nodes is $V(\mu) \cup \{env\}$; and (c) the set of edges is $E(\mu) \cup \{\{env, skin\}\}$. The node env is called the* environment *of the structure μ.*

Note that we have only included a new node representing the environment which is only connected with the skin, while the original membrane structure remains unchanged. In this way, every configuration of the system informs about the contents of the environment.

Definition 3.4 *A* language accepting P system *is a P system with input, (Π, Σ, i_Π), and with external output, such that the output alphabet contains only two elements: Yes and No.*

This definition is stated in a general way, but in this paper P systems within the active membrane model will be used. We refer to [2] (see chapter 7) for a detailed definition of evolution rules, transition steps, and configurations in this model.

Now let us define the *Output* function for our P systems. Given a computation $\mathcal{C} = \{C^i\}_{i<r}$, we will denote by M^j_{env} the content of the environment in the configuration C^j.

Definition 3.5 *The output of a computation $\mathcal{C} = \{C^i\}_{i<r}$ is:*

$$Output(\mathcal{C}) = \begin{cases} Yes, & if\ \mathcal{C}\ is\ halting,\ Yes \in M^{r-1}_{env}\ and\ No \notin M^{r-1}_{env}, \\ No, & if\ \mathcal{C}\ is\ halting,\ No \in M^{r-1}_{env}\ and\ Yes \notin M^{r-1}_{env}, \\ not\ defined, & otherwise. \end{cases}$$

If $\mathcal{C}$ satisfies any of the two first conditions, then we say that it is a *successful computation.*

Definition 3.6 *A language accepting P system is said to be* valid *if for every halting computation, and only for them, one symbol Yes or one symbol No (but not both) is sent out (in the last step of the computation).*

Definition 3.7 *We say that $\mathcal{C}$ is an* accepting *computation (respectively,* rejecting *computation) if the object Yes (respectively, No) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$; that is, $Output(\mathcal{C}) = Yes$ (respectively, $Output(\mathcal{C}) = No$).*

Definition 3.8 *A* language recognizer P system *is a valid language accepting P system such that all its computations halt.*

This recognizer systems are specially suitable when trying to solve decision problems.

In this paper we will deal with recognizer P systems with active membranes, input membrane, and external output. We recall the fact that a P system with active membranes is a tuple

$$\Pi = (\Sigma, H, \mu, \omega_1, \dots, \omega_m, R),$$

where:

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

2. Σ 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. $\omega_1, \ldots, \omega_m$ are strings over Σ, 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) $[\, a \rightarrow \omega\,]_h^\alpha$ for $h \in H$,$\alpha \in \{+,-,0\}$, $a \in \Sigma$, $\omega \in \Sigma^*$, *object evolution rules*: 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.

   (b) $a\, [\;\,]_h^{\alpha_1} \rightarrow [\, b\,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+,-,0\}$, $a, b \in \Sigma$, *communication rules (send in rules)*: 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) $[\, a\,]_h^{\alpha_1} \rightarrow b\; [\;\;\,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+,-,0\}$, $a, b \in \Sigma$, *communication rules (send out rules)*: 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) $[\, a\,]_h^\alpha \rightarrow b$ for $h \in H$, $\alpha \in \{+,-,0\}$, $a, b \in \Sigma$, *dissolving rules*: A membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.

   (e) $[\, a\,]_h^{\alpha_1} \rightarrow [\, b\,]_h^{\alpha_2}\; [\, c\,]_h^{\alpha_3}$ for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}$, $a, b, c \in \Sigma$, *division rules for elementary membranes*: An elementary 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.

- 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).

4 CLIPS as a Programming Language for Production Systems

CLIPS is the tool we have chosen to developed a simulator of this variant of P systems. Here we present a short introduction to this programming language; for more information, tutorials, user guide, etc., visit the CLIPS Web Page at [10].

CLIPS is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. The origins of the C Language Integrated Production System (CLIPS) date back to 1984 at NASA's Johnson Space Center. Originally, the primary representation methodology in CLIPS was a forward chaining rule language based on the Rete algorithm (hence the Production System part of the CLIPS acronym).

CLIPS is now widely used throughout the government, industry, and academia. Its key features are:

- *Knowledge Representation*: CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented, and procedural.
- *Portability*: CLIPS is written in C for portability and speed and has been installed on many different operating systems without code changes.
- *Integration/Extensibility*: CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, Java, FORTRAN, and ADA.
- *Low Cost*: CLIPS is maintained as public domain software.

5 A CLIPS Simulator for Recognizer P Systems with Active Membranes

In order to design a simulator in CLIPS for recognizer P systems with active membranes we have to give a formal representation for their basic structures, that is, for the membrane structure with the content (multiset) of each membrane and for the evolution rules.

5.1 Representation of Membranes Structures

A membrane will be represented using an unordered fact in the Working Memory as follow:

```
(deftemplate membrane
   (slot number)
   (slot father)
   (slot children)
   (slot configuration)
   (slot evolved)
   (slot label)
   (slot polarization)
   (multislot multiset))
```

Using this representation we have:

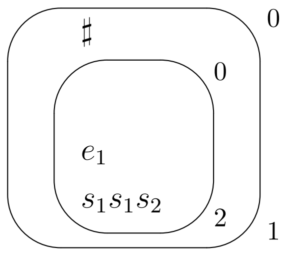

Figure 1: Example configuration

- Each membrane will be identified in a one-one manner by the slot *number*.
- The membrane structure will be represented using the slots *father*, which will store the number of the father membrane, and *children*, which will store the number of each of the child membranes.
- During the simulation of a step in the evolution of a P system, we will keep two copies of the same membrane, one for the current configuration and one for the next configuration. The slot configuration will be used to differentiate between these two copies.
- The slot *evolved* will be used to show whether or not the membrane has already evolved in the current step and so it cannot evolve anymore in this step of the computation.
- The slots *label*, *polarization*, and *multiset* will store the label, the polarization and the multiset of the membrane.

Note that the environment will be represented as the membrane number 0, which has no father, no polarization, no label and one child membrane, the skin, whose number will be 1.

For example, the following unordered facts represent the configuration given above:

```
(membrane (number 0) (children 1) (multiset))
(membrane (number 1) (father 0) (children 2)
          (configuration current)(evolved 0)
          (label 1) (polarization 0) (multiset #))
(membrane (number 2) (father 1) (children )
          (configuration current)(evolved 0)
          (label 2) (polarization 0) (multiset e 1 , s 1 , s 1 , s 2  ))
```

5.2 Representation of Rules

The rules are represented as facts in the Working Memory. We will represent each type of rule as an unordered fact which will store the relevant features of the rules, that is label, polarization and objects.

Each type will be represented using the following templates:

- Evolution rules: $[a \to \omega]_h^\alpha$

```
(deftemplate rule-a
   (slot label)
   (slot polarization)
   (multislot left-element)
   (multislot right-elements))
```

For example, the rule $[s_0 \to S]_2^+$ will be represented by the following fact:

```
(rule-a (label 2) (polarization +)
        (left-element s0) (right-elements S))
```

- Send *in* communication rules: $a\,[\ \ ]_h^{\alpha_1} \to [\,b\,]_h^{\alpha_2}$

```
(deftemplate rule-b
   (slot label)
   (slot left-pol)
   (multislot left-element)
   (slot right-pol)
   (multislot right-element))
```

For example, the rule $d_1\,[\ \ ]_2^0 \to [\,d_2\,]_2^0$ will be represented by the following fact:

```
(rule-b (label 2) (left-pol 0)(left-element d1)
                  (right-pol 0)(right-element d2))
```

- Send *out* communication rules: $[\,a\,]_h^{\alpha_1} \to b\,[\ \ ]_h^{\alpha_2}$

```
(deftemplate rule-c
   (slot label)
   (slot left-pol)
   (multislot left-element)
   (slot right-pol)
   (multislot right-element))
```

For example, the rule $[\,z\,]_2^+ \to \sharp\,[\ \ ]_2^0$ will be represented by the following fact:

```
(rule-c (label 2) (left-pol +)(left-element z)
                  (right-pol 0)(right-element #))
```

- Dissolution rules: $[\,a\,]_h^{\alpha} \to b$

```
(deftemplate rule-d
   (slot label)
   (slot polarization)
   (multislot left-element)
   (multislot right-element))
```

For example, the rule $[\, b_{n+1}\,]_2^0 \to YES$ will be represented by the following fact:

```
(rule-d (label 2) (polarization 0)
                  (left-element b_n+1) (right-element YES))
```

- Division rules: $[\, a\,]_h^{\alpha_1} \to [\, b\,]_h^{\alpha_2}\, [\, c\,]_h^{\alpha_3}$

```
(deftemplate rule-e
   (slot label)
   (slot left-pol)
   (multislot left-element)
   (slot right-pol-1)
   (multislot right-element-1)
   (slot right-pol-2)
   (multislot right-element-2))
```

For example, the rule $[\, e_n\,]_2^0 \to [\, e_{n+1}\,]_2^+\, [\, e_{n+1}\,]_2^0$ will be represented by the following fact:

```
(rule-e (label 2) (left-pol 0) (left-element e_n)
                  (right-pol-1 +) (right-element-1 e_{n+1})
                  (right-pol-2 0) (right-element-2 e_{n+1}))
```

6 The Algorithm

The CLIPS algorithm to simulate a recognizer P system works in a natural way. We will briefly describe it:

Step 1: Initialization. Once the initial configuration associated to the input has been constructed, a copy of each membrane with (`configuration current`) will be made. In this new copy we will set the slot *configuration* to next. With these two copies we will be able to handle the current configuration to get to the next configuration.

Step 2: Transition. If there exists a membrane with (`configuration current`) satisfying the condition of a rule, then the rule is applied in the following way:

Evolution involving only elements: At this stage, only rules of type (a) can be applied. If there exists one membrane with the (`configuration current`) satisfying the conditions of a rule of type (a) then the object which tiggers the rule is removed from the membrane with (`configuration current`) and the RHS part of the rule is added to the membrane with (`configuration next`). The following CLIPS rule simulates the application of a type (a) rule:

```
(defrule elements-evolution
   (rule-a (label ?lb) (polarization ?pol)
           (left-element $?left-element)
           (right-elements $?right-elements))
   ?membrane0 <- (membrane (configuration current)
                           (number ?n)
```

```
                          (label ?lb)
                          (polarization ?pol)
                          (multiset $?B0
                                    , $?left-element ,
                                    $?F0))
   ?membrane1 <- (membrane (configuration next)
                          (number ?n)
                          (multiset $?B1
                                    , $?left-element ,
                                    $?F1))
  =>
   (modify ?membrana0 (contenido $?B0 , $?F0))
   (modify ?membrana1 (contenido $?B1 , $?right-elements , $?F1))))
```

This stage ends when no more rules of type (a) can be applied.

Evolution involving membranes: At this stage, only one rule of the other types (not (a)) can be applied to the same membrane. Let us remember that this simulation only works with recognizer P systems (in fact, it works with confluent ones). The action depends on the kind of rule to apply:

- *Send in rule*: If there exists one membrane with the (`configuration current`) and (evolved 0) satisfying the conditions of a rule of type (c) and its father contains the LHS object of the rule, then this object is removed from both the father membrane with (`configuration current`) and with (`configuration next`), the RHS object of the rule is added to the copy of the membrane with (`configuration next`) and the slot evolved of the copy of the membrane with (`configuration current`) is set to 1 in order to avoid further evolution involving this membrane in this step. The following CLIPS rule simulates the application of a type (b) rule:

```
(defrule send-in
   (rule-b (label ?lb) (left-pol ?left-pol)
           (left-element $?left-element)
           (right-pol ?right-pol) (right-element $?right-element))
   ?membrane0 <- (membrane (evolved 0)
                          (configuration current)(number ?n)
                          (label ?lb)
                          (polarization ?left-pol)
                          (father ?f))
   ?membrane1 <- (membrane (configuration next)(number ?n)
                          (multiset $?cont))

   ?father0 <- (membrane (configuration current) (number ?f)
                        (multiset $?B0
                                  , $?left-element ,
                                  $?F0))
   ?father1 <- (membrane (configuration next) (number ?f)
                        (multiset $?B1
                                  , $?left-element ,
                                  $?F1))
  =>
   (modify ?membrane0 (evolved 1))
   (modify ?membrane1 (multiset $?cont $?right-element ,)
                    (polarization ?right-pol))
   (modify ?father1 (multiset $?B1 , $?F1))
   (modify ?father0 (multiset $?B0 , $?F0)))
```

- *Send out rule*: If there exists one membrane with the (`configuration current`) satisfying the conditions of a rule of type (b), then the object which tiggers the rule is removed from the membrane with (`configuration current`), the slot evolved is set to 1 and the RHS object of the rule is added to the copy of the father membrane with (`configuration next`). The following CLIPS rule simulates the application of a type (c) rule:

```
(defrule send-out
   (rule-c (label ?lb) (left-pol ?left-pol)
                       (left-element $?left-element)
                       (right-pol ?right-pol)
                       (right-element $?right-element))
   ?membrane0 <- (membrane (evolved 0)(configuration current)
                           (number ?n) (father ?f)
                           (label  ?lb) (polarization ?left-pol)
                           (multiset $?B0 , $?left-element , $F0))
   ?membrane1 <- (membrane (configuration next)(number ?n)
                           (multiset $?B1 , $?left-element , $?F1))
   ?father1 <- (membrane (configuration next)
                         (number ?f) (multiset $?cont))
  =>
   (modify ?membrane0 (evolved 1)
                      (multiset $?B0 , $?F0))
   (modify ?membrane1 (multiset $?B1 , $?B1)
                      (polarization ?right-pol))
   (modify ?father1 (multiset $?cont $?right-element ,)))
```

- *Dissolving rule*: If there exists one membrane with (`configuration current`) and (evolved 0) satisfying the conditions of a rule of type (d), then the copies of this membrane with (`configuration current`) and (`configuration next`) are removed, the multiset of the membrane is added to the copy of its father membrane with (`configuration next`) replacing the LHS element of the rule by the RHS object and we update the corresponding membrane structure. The following CLIPS rule simulates the application of a type (d) rule:

```
(defrule dissolution
   (rule-d (label ?lb) (polarization ?pol)
                       (left-element $?left-element)
                       (right-element $?right-element))
   ?membrane0 <- (membrane (evolved 0) (configuration current)
                           (number ?n)
                           (label ?lb) (polarization ?pol)
                           (multiset $?B , $?left-element , $?F)
                           (father ?f) (children $?children))
   ?membrane1 <- (membrane (number ?n) (configuration next))
   ?father <- (membrane (number ?f) (configuration next)
                        (multiset $?cont ,)
                        (children $?cb ?n $?cf))
  =>
   (retract ?membrane0 ?membrane1)
   (modify ?father (multiset $?cont $?B , $?right-element , $?F)
                   (children $?cp $?children $?cf))
   (assert (change-father ?n ?f)))
```

- Division rule: If there exists one membrane with (`configuration current`) and (evolved 0) satisfying the conditions of a rule of type (e), then the copies of this membrane with (`configuration current`) and (`configuration next`) are removed, two new membranes with new identification numbers are created with the same label, the corresponding polarization and the resulting multiset after replacing the LHS object of the rule by the corresponding element in the RHS; finally, we remove from the father membrane the old identification number and add the identification numbers of the new membranes. The following CLIPS rule simulates the application of a type (e) rule:

```
(defrule division
   (rule-e (label ?lb) (left-pol ?left-pol)
           (left-element $?left-element)
           (right-pol-1 ?right-pol-1)
           (right-element-1 $?right-element-1)
           (right-pol-2 ?right-pol-2)
           (right-element-2 $?right-element-2))
   ?membrane0 <- (membrane (evolved 0) (configuration current)
                   (number ?n1)
                   (label ?lb)
                   (polarization ?left-pol)
                   (multiset $?B0 , $?left-element , $?F0))
   ?father1 <- (membrane (configuration next)
                        (number ?f)
                        (children $?cb ?n1 $?cf))
   ?membrane1 <- (membrane (configuration next)
                           (number ?n1)
                           (multiset $?B1
                                     , $?left-element ,
                                     $?F1))
  =>
   (retract ?membrane1 ?membrane0)
   (bind ?*num* (+ ?*num* 1))
   (assert (membrane (evolved 0) (configuration next)
                     (number ?*num*)
                     (label ?lb)
                     (polarization ?right-pol-1)
                     (multiset $?B1
                               , $?right-element-1 ,
                               $?F1)
                     (children )
                     (father ?f)))
   (bind ?*num* (+ ?*num* 1))
   (assert (membrane (evolved 0) (configuration next)
                     (number ?*num*)
                     (label ?lb)
                     (polarization ?right-pol-2)
                     (multiset $?B1
                              , $?right-element-2 ,
                              $?F1)
                     (children )
                     (father ?f)))
   (modify ?father1 (children $?Hb (- ?*num* 1) ?*num* $?Hf)))
```

7 A Case Study: VALIDITY Problem

7.1 General Solution to VALIDITY by Recognizer P Systems with Active Membranes

The **VALIDITY** problem is the following one: *Given a boolean formula in conjunctive normal form, to determine whether or not is a tautology; that is, whether it evaluates true on every assignment to its variables.*

Following [6] we present a family of recognizers P systems with active membranes using 2–division (see [2], section 7.2) solving the **VALIDITY** problem in *linear* time. Let us suppose that $\varphi = C_1 \wedge \dots C_m$ 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 $\Sigma(m,n) = \{x_{i,j}, \overline{x}_{i,j} : \ 1 \le i \le m, \ 1 \le j \le n\}$, $i(m,n) = 2$, and $\Pi(\langle n,m\rangle) = (\Gamma(m,n), \{1,2\}, [_1 \ [_2 \ ]_2 \ ]_1, w_1, w_2, R)$ is defined as follows:

$$\begin{aligned} \Gamma(m,n) &= \Sigma(m,n) \cup \{c_k : \ 1 \le k \le m+1\} \cup \{d_k : \ 1 \le k \le 3n+2m+2\} \\ &\cup \ \{e_k : \ 0 \le k \le 3\} \cup \{r_{i,k} : \ 0 \le i \le m, 1 \le k \le 2n\} \cup \{Yes, No\}. \end{aligned}$$

We say that every membrane with label 2 is an *internal membrane.*

The initial content of each membrane is: $w_1 = \{e_1\}$ and $w_2 = \{d_1\}$.

The set of rules, R, is given by:

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

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

The rules of (b) try to implement a process allowing the internal membranes to encode the *assignment* to 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) $\{[_2 x_{i,j} \to x_{i,j-1}]_2^+, \ [_2 x_{i,j} \to x_{i,j-1}]_2^- : \ 1 \le i \le m, \ 2 \le j \le n\}$,
$\{[_2 \ \overline{x}_{i,j} \to \ \overline{x}_{i,j-1}]_2^+, \ [_2 \ \overline{x}_{i,j} \to \ \overline{x}_{i,j-1}]_2^- : \ 1 \le i \le m, \ 2 \le j \le n\}$.

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

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) $e_0[_2\ ]_2^0 \to [_2 c_1]_2^-$; $[_1 d_{3n-1} \to d_{3n}]_1^0$.

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

The rules of (h) supply counters in the skin, through objects d, in such a way that, if the object d_{3n+2m} appears, then it shows the end of the checking of the clauses. The object $d_{3n+2m+1}$ (and the object $d_{3n+2m+2}$) 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 re-label 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 clause are being determined.

(m) $[_2 c_{m+1}]_2^- \to [_2\ ]_2^+ c_{m+1}$; $[_1 c_{m+1}]_1^0 \to [_1\ ]_1^+ c_{m+1}$.

The first of the rules of (m) sends to the skin the objects c_{m+1} appearing in the internal membranes of the $(5n+2m-2)$-th step of the computation, changing the polarization of these membranes to positive.

The second of the rules of (m) sends out of the system one object c_{m+1} when it has neutral charge.

(n) $d_{3n+2m+1}[_2\ ]_2^+ \to [_2 d_{3n+2m+2}]_2^+$; $d_{3n+2m+1}[_2\ ]_2^- \to [_2 d_{3n+2m+2}]_2^-$.

The rules of (n) produce the evolution of objects $d_{3n+2m+1}$ to objects $d_{3n+2m+2}$ in the internal membranes with non neutral charge, with no change of their polarization.

(o) $[_2 d_{3n+2m+2}]_2^- \to [_2\ ]_2^- d_{3n+2m+2}$.

The rule of (o) returns to the skin one object $d_{3n+2m+2}$ of an internal membrane with negative charge; this will represent that some assignment makes false the formula.

(p) $[_1 d_{3n+2m+2}]_1^+ \to [_1\ ]_1^- d_{3n+2m+2}$; $[_1 d_{3n+2m+2}]_1^0 \to [_1\ ]_1^- d_{3n+2m+2}$.

By using the rules of (p) objects $d_{3n+2m+2}$ are sent out of the system and, also, they produce a change in the polarization. Hence, they allow us to control the charge of the skin regarding to the presence of objects $d_{3n+2m+2}$ in it.

(q) $\{[_1 e_k \to\ e_{k+1}]_1^+ :\ 1 \leq k \leq 2\}$.

The objects e_1, e_2, and e_3 control the last stage of the computation. The rules of (q), with the rules of (r), produce an accepting computation or a rejecting one.

(r) $[_1 e_3]_1^+ \to\ [_1\ ]_1^+ Yes$; $[_1 e_3]_1^- \to\ [_1\ ]_1^- No$; $[_1 e_1]_1^- \to\ [_1\ ]_1^- No$.

In [6] it is proved that the family $\Pi = (\Pi(t))_{t\in\mathbb{N}^+}$ solves VALIDITY problem in linear tiem. As input data for this P system, we consider the multiset:

$$\{x_{ij}\ :\ x_j \in C_i\} \cup \{\overline{x}_{ij}\ :\ \neg x_j \in C_i\}.$$

The execution of the P system with input given above 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 e_1, e_2, and e_3.

- To produce the output Yes, the object c_{m+1} must have been produced in every internal membrane of the configuration $\mathcal{C}_{5n+2m-1}$. Then, after 5 steps, the system returns Yes to the environment, through the evolution of objects e_3 present in the skin, and when it has positive charge.
- To produce the output No, two cases can occur:
  (a) There is *no* internal membrane of the configuration $\mathcal{C}_{5n+2m-1}$ where the object c_{m+1} is present; in this case, after 5 steps, the system returns No to the environment, through the evolution of the object e_1 present in the skin membrane and when it has negative charge.
  (b) There are k internal membranes (with $1 \leq k < 2^n$) of the configuration $\mathcal{C}_{5n+2m-1}$ where the object c_{m+1} is present; in this case, after 5 steps, the system returns No to the environment, through the evolution of objects e_3 present in the skin and when it has negative charge.

Therefore, the total number of steps in the output stage is 5.

7.2 A CLIPS Session for $m = n = 2$

In this section we present a session with the simulator for a instance of the problem. We consider the formula: $\varphi = (x_1 \vee \neg x_1) \wedge (x_2 \vee \neg x_2)$.

```
CLIPS> (load ‘‘SIMULATOR.clp’’)

CLIPS> (reset)
CLIPS> (run)

******************************************************************************
* THIS IS PROGRAM WRITTEN IN CLIPS THAT SIMULATES COMPUTATIONS OF P-SYSTEMS. *
******************************************************************************

The P-system can be introduced using the keyboard or a file. Choose one
of the following options:
 1 .- If you want to introduce the P-system using the keyboard.
 2 .- If you want to introduce a P-system written in a file.

2

Write the path and the file where the P-system is written:
/home/Fran/validity.clp

         ******************************************************************
         **                                                              **
         **  READ CAREFULLY THE FOLLOWING INFORMATION ABOUT THE P-SYSTEM  **
         **                                                              **
         ******************************************************************

This recognizer P-system with active membranes solves the
VALIDITY problem. We will work with formulas in CNF.
The system has two parameters:
        - n number of variables
        - m number of clauses
The input must be introduced as follows:
Let Form = C 1 ^ C 2 ^ ... ^ C m, a formula in CNF with n
variables x 1 , x 2 , ... , x n. The input of the P-system is given
by the following function:
g(Form) = { x i j : x j in C i} U { -x i j : -x j in C i}
For example:
Form = [ x 1 v -x 2 v x 3 ] ^ [ x 2 v -x 3 ] ^ [ -x 1 v x 3 ]
g(Form) = { x 1 1 , -x 1 2 , x 1 3 , x 2 2 , -x 2 3 , -x 3 1 , x 3 3 }

                     ******************************
                     * P-SYSTEM SUCESSFULLY LOADED *
                     ******************************

Write the value of the parameter n : 2


Write the value of the parameter m : 2

Write the input multiset following the instructions given above:
x 1 1 , x 2 2 , -x 1 1 , -x 2 2

Before the simulation of one computation of the P-system with the given
input you can choose one of the following options:
     1 .- If you want to save the configurations in a file.
     2 .- If you don't want to save the configurations.

2

Now you can choose one of the following options:
     1.- If you want to see each configuration.
     2.- If you only want to see a particular configuration.
     3.- If you only want to see the answer of the system.
1

You can also choose whether or not you want to see the applied rules
in each configuration:
     1.- If you want to see the applied rules.
     2.- If you don't want to see the applied rules.

1

Configuration number: 0

[environment [multiset ]]
[skin     [children 2]
          [label 1]
          [polarity 0]
          [multiset , e 1 ,]]
[membrane
    [number 2]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , d 1 , x 1 1 , x 2 2 , -x 1 1 , -x 2 2 ,]]
```

Above we have the initial configuration with the corresponding input. And now the *generation stage* begins:

```
The applied rules are:

[ d 1 ] 2 0 -> [ d 1 ] 2 + [  d 1 ] 2 -

Configuration number: 1

[environment [multiset ]]
[skin     [children 3 4]
          [label 1]
          [polarity 0]
          [multiset , e 1 ,]]
[membrane
    [number 4]
    [children ]
    [father 1]
    [label 2]
    [polarity -]
    [multiset , d 1 , x 1 1 , x 2 2 , -x 1 1 , -x 2 2 ,]]
[membrane
    [number 3]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , d 1 , x 1 1 , x 2 2 , -x 1 1 , -x 2 2 ,]]

The applied rules are:

[x 1 1 -> r 1 1 ] 2 +
[x 2 2 -> x 2 1 ] 2 +
[-x 1 1 ->  ] 2 +
[-x 2 2 -> -x 2 1 ] 2 +
[-x 1 1 -> r 1 1 ] 2 -
[x 1 1 ->  ] 2 -
[x 2 2 -> x 2 1 ] 2 -
[-x 2 2 -> -x 2 1 ] 2 -
[ d 1 ] 2 + -> d 1 [ ] 2 0
[ d 1 ] 2 - -> d 1 [ ] 2 0

Configuration number: 2

[environment [multiset ]]
[skin     [children 3 4]
          [label 1]
          [polarity 0]
          [multiset , e 1 , d 1 , d 1 ,]]
[membrane
    [number 3]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 1 , x 2 1 , -x 2 1 ,]]
[membrane
    [number 4]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , x 2 1 , r 1 1 , -x 2 1 ,]]

The applied rules are:

[r 1 1 -> r 1 2 ] 2 0
[r 1 1 -> r 1 2 ] 2 0
d 1 [ ] 2 0 -> [ d 2 ]  2 0
d 1 [ ] 2 0 -> [ d 2 ]  2 0

Configuration number: 3
```

```
[environment [multiset ]]
[skin     [children 3 4]
          [label 1]
          [polarity 0]
          [multiset , e 1 ,]]
[membrane
    [number 3]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 2 , x 2 1 , -x 2 1 , d 2 ,]]
[membrane
    [number 4]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , x 2 1 , r 1 2 , -x 2 1 , d 2 ,]]
```

Until now, all the possible assignments to x_1 have been generated. The *generation stage* works in this way to generate all the possible assignments to all the variables. We skip the next configurations until the presence of the object d_2 (d_n) shows that the *synchronization stage* is beginning.

```
Configuration number: 5

[environment [multiset ]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity 0]
          [multiset , e 1 , d 2 , d 2 , d 2 , d 2 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 2 1 , r 1 3 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 3 , r 2 1 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 3 , r 2 1 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 3 , r 2 1 ,]]
```

At the end of the *synchronization stage* we have the second subindexes of the objects $r_{i\ k}$ equal to 4 ($2n$); the *checking stage* begins:

```
The applied rules are:

[d 4 -> d 5 , e 0 ] 1 0
[d 4 -> d 5 , e 0 ] 1 0
[d 4 -> d 5 , e 0 ] 1 0
[d 4 -> d 5 , e 0 ] 1 0
[r 2 3 -> r 2 4 ] 2 0
[r 2 3 -> r 2 4 ] 2 0
[r 2 3 -> r 2 4 ] 2 0
[r 2 3 -> r 2 4 ] 2 0

Configuration number: 8

[environment [multiset ]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity 0]
          [multiset , e 1 , d 5 , e 0 , d 5 , e 0 , d 5 , e 0 , d 5 , e 0 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 4 , r 2 4 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 4 , r 2 4 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 1 4 , r 2 4 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity 0]
    [multiset , r 2 4 , r 1 4 ,]]

The applied rules are:

[d 5 -> d 6 ] 1 0
[d 5 -> d 6 ] 1 0
[d 5 -> d 6 ] 1 0
[d 5 -> d 6 ] 1 0
e 0 [ ] 2 0 -> [ c 1 ]  2 -
e 0 [ ] 2 0 -> [ c 1 ]  2 -
e 0 [ ] 2 0 -> [ c 1 ]  2 -
e 0 [ ] 2 0 -> [ c 1 ]  2 -

Configuration number: 9

[environment [multiset ]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity 0]
          [multiset , e 1 , d 6 , d 6 , d 6 , d 6 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity -]
    [multiset , r 2 4 , r 1 4 , c 1 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity -]
    [multiset , r 1 4 , r 2 4 , c 1 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity -]
    [multiset , r 1 4 , r 2 4 , c 1 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity -]
    [multiset , r 1 4 , r 2 4 , c 1 ,]]
```

This stage goes on until the object c_3 is released into the environment and then the *out stage* takes place:

```
The applied rules are:

d 11 [ ] 2 + -> [ d 12 ]  2 +
d 11 [ ] 2 + -> [ d 12 ]  2 +
d 11 [ ] 2 + -> [ d 12 ]  2 +
d 11 [ ] 2 + -> [ d 12 ]  2 +
[ c 3 ] 1 0 -> c 3 [ ] 1 +


Configuration number: 15

[environment [multiset c 3 ,]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity +]
          [multiset , e 1 , c 3 , c 3 , c 3 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]

The applied rules are:

[e 1 -> e 2 ] 1 +

Configuration number: 16

[environment [multiset c 3 ,]]
[skin     [children 5 6 7 8]
```

```
          [label 1]
          [polarity +]
          [multiset , e 2 , c 3 , c 3 , c 3 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]

The applied rules are:

[e 2 -> e 3 ] 1 +

Configuration number: 17

[environment [multiset c 3 ,]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity +]
          [multiset , e 3 , c 3 , c 3 , c 3 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]

The applied rules are:

[ e 3 ] 1 + -> YES [ ] 1 +

Configuration number: 18

[environment [multiset c 3 , YES ,]]
[skin     [children 5 6 7 8]
          [label 1]
          [polarity +]
          [multiset , c 3 , c 3 , c 3 ,]]
[membrane
    [number 7]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 5]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 8]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]
[membrane
    [number 6]
    [children ]
    [father 1]
    [label 2]
    [polarity +]
    [multiset , r 0 4 , r 0 4 , d 12 ,]]

The system has reached a halting configuration in the step number 18
and the element YES has been released into the environment.
```

At the end, the system halts in the configuration number 18 ($5n+2m+4$) and recognizes that the formula is a tautology.

8 Conclusions and Future Work

In this parper a CLIPS simulator of recognizer P systems with active membranes has been presented. This program provides an effective tool for making experiments with recognizer P systems to solve concrete decision problems. One example is the VALIDITY problem, but the simulator is for a general purpose, covering all the confluent P systems with active membranes using 2-division.

A future work is to provide this CLIPS program with a friendly interface using JAVA, C or Tcl/Tk, programming languages which can easily interact with CLIPS.

Representing P systems using Production Systems techniques can be a starting point to explore analogies between P systems and Production Systems. In this paper we have shown the convenience of simulating P systems within the framework of Production Systems; but the other direction is also interesting. Conversely, due to the analogies between P systems and Production Systems, P systems could be used for providing new insights in the study of Production Systems.

Acknowledgements. This work is supported by the Ministerio de Ciencia y Tecnología of Spain, by the *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds, and by a FPI fellowship (of the second author) from the University of Seville.

References

[1] Cordón-Franco, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Sancho-Caparrini, F.: A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*, to appear

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

[3] Paun, G.: Computing with membranes, *Journal of Computer and Systems Sciences*, 61(1), 2000, 108-143.

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

[5] Paun, 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.: *Solving VALIDITY problem by active membranes with input.* M. Cavaliere, C. Martín-Vide and G.

Paun (eds) Proceedings of the Brainstorming week on Membrane Computing, Report GRLMC 26/03, 279-290

[7] Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: *Teoría de la Complejidad en modelos de computacion celular con membranas.* Ed. Kronos, 2002

[8] Pérez Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems.* Budapest, Hungary, 2003, 284-294

[9] Pérez-Jiménez, M.J., Riscos-Núñez, A.: A linear-time solution for the Knapsack problem using active membranes, C. Martin-Vide, G. Paun, G. Rozenberg, A. Salomaa (eds) *Workshop on Membrane Computing, WMC 2003, Lecture Notes in Computer Science*, 2933, 2004, 250-268.

[10] CLIPS Web Page: http:// www.ghg.net/clips/CLIPS.html

[11] The P Systems Web Page: http://psytems.disco.unimib.it/