IMPROVING ACTIVE RULES PERFORMANCE IN NEW P SYSTEM COMMUNICATION ARCHITECTURES

Juan Alberto de Frutos, Luis Fernández, Carmen Luengo, Alberto Arteta

***Abstract**: Membrane systems are models of computation which are inspired by some basic features of biological membranes. Transition P systems are very simple models. Many hardware and software architectures have been proposed for implementing them. In particular, there are implementations in cluster of processors, in microcontrollers and in specialized hardware. This work proposes an analysis of the P system in order to be able to reduce the execution time of a given evolution step.*

We present a solution for improving the time of working out the active rules subset of a membrane. This task is critical for the entire evolution process efficiency because it is performed inside each membrane in every evolution step. Therefore, we propose to carry out a static analysis over the P system. The collected information is used for obtaining a decision tree for each membrane. During the execution time of the P system, active rules of a membrane will be determined as a result of a classification problem from the corresponding decision tree. By incorporating decision trees for this task, we will notice some improvements.

***Keywords**: Decision Tree, ID3, Active Rules, Transition P System*

***ACM Classification Keywords**: I.2.6 Learning – Decision Tree; D.1.m Miscellaneous – Natural Computing*

Introduction

Membrane Computing was introduced by Gh. Păun in [Păun, 2000a], as a new branch of natural computing, inspired on living cells. Membrane systems establish a formal framework in which a simplified model of cells constitutes a computational device. Starting from a basic model, for Transition P systems many different variants have been considered; and many of them have been proven to be equivalent to the Turing Machine in computational complexity terms.

Membrane Computing has a massively parallel character at two levels: each region applies its rules in a parallel way and every region performs this work simultaneously. In such a way that there are some variants of P systems which can solve complete NP problems in polynomial time. Regarding implementations of Transition P systems, there are several challenges for researches in order to get real implementations of such systems. Next section shows an overview of some implementations for which the present work is going to propose an optimization.

A Transition P System evolves through transitions between two consecutive configurations that are determined by the membrane structure and multisets present inside membranes. It can be considered two sequential phases in every transition step: application of evolution rules inside membranes, and communication among membranes in the system. The present work tries to contribute with improvements in the first phase implementation.

The first task in application of evolution rules inside a membrane phase is to determine the activeness for every rule. A rule is active if it can be applied in the current evolution step; that depends on the P system state in that moment. Then, the subset of active rules will be applied in a maximal parallel and non deterministic way. There are many papers which main goal is to improve the internal parallelism of the membrane in such a way that several active rules can be applied simultaneously in a implementation. However, about optimization of the

process that obtains active rules we can only mention the work done in [Fernández, 2006]. Specifically, Fernández uses decision trees in applicability of evolution rules. One of the goals of the present work is to extend those decision trees in such a way that not only applicability is considered, but also all conditions for a rule to be applied in an evolution step.

The structure of this work is as follows: firstly, related works about architectures for implementing P systems are presented; next, it will be focused on the conditions for an evolution rule to be active; in the following section we will show how a decision tree for obtaining active rules is built; then, a software implementation directed to the proposed architectures; and finally, an efficiency analysis will be carried out, together with our conclusions.

Proposed implementations for P Systems

They can be grouped into three researching lines. First of them tries to achieve a hardware specialized in membrane processing. Second line makes use of a cluster of processors in which membranes are allocated. And the last line is based on microcontrollers PIC16F88 adapted to membrane processing. As we are going to propose a software implementation for obtaining active rules in this paper, we will focus on the last two lines in the following.

Architectures based on a cluster of computers

Computers are connected by a local net. Each one houses several membranes, reaching a certain degree of parallelism. In this line we can mention the works carried out in [Syropoulos, 2003], [Ciobanu, 2004], [Tejedor, 2007], [Bravo, 2007a], [Bravo, 2007b] and [Bravo, 2008].

Syropoulos et al. propose a distributed implementation by using Java Remote Method Invocation (RMI), while Ciobanu et al. use Message Passing Interface (MPI). These authors do not carry out a detailed analysis of the communication phase; however Ciobanu noticed the possibility of network congestion when the number of membranes grows in the system. That is why the works presented by Tejedor et al. [Tejedor, 2007] and Bravo et al. [Bravo, 2007a], [Bravo, 2007b] and [Bravo, 2008] try particularly to tackle the bottleneck communication problem.

The architecture proposed in [Tejedor, 2007] avoids communication collisions and reduces the number and length of external communications. It is based on four pillars: several membranes are placed at each processor which evolve, at worst sequentially; in every processor a proxy is in charge of communications; processors are placed in a tree topology in order to minimize the number of communications; and it is established a communication order through a token. As conclusion, Tejedor et al. states that "if it is possible to make that application time be N faster times [...] the number of membranes that would be run in a processor would be multiplied by $\sqrt{N}$, the number of required processors would be divided by the same factor and the time required to perform an evolution step would improve approximately with the same factor $\sqrt{N}$. The goal of the present work fits in this context that is to reduce the application time inside a membrane.

More recently, Bravo et al. in [[Bravo, 2007a], [Bravo, 2007b] and [Bravo, 2008], have proposed three variants of the architecture proposed by Tejedor et al, in which the total evolution time has been progressively reduced.

Architectures based on microcontrollers

This line of implementing P systems has been proposed by Gutierrez et al. in [Gutierrez, 2006], [Gutierrez, 2007] and [Gutierrez, 2008]. It consists in a low cost hardware based on microcontrollers PIC16F88 that making use of external memory modules is able to solve the problem of small capacity of storage in these devices. It means a flexible solution due to microcontrollers allow to be software programmed. Figure 1 contains a picture with a real

implementation. The represented microcontroller has been adapted to perform membrane execution. Besides, it has been designed to be connected up to with 254 additional microcontrollers.

Fig. 1. A microcontroller PIC16F88 for implementing P systems.

Conditions for a rule to be active

Formally, a transition P system of degree *m* is a construct of the form

$$\Pi = (U, \mu, \omega_1, \ldots, \omega_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_0),$$ where:

- *U* is the alphabet of objects
- μ is a membrane structure, consisting of *m* membranes, labelled with *1,2,...., m*. It is a hierarchically arranged set of membranes, contained in a distinguished external membrane, called skin membrane. Several membranes can be placed inside a parent membrane; and finally, a membrane without any other membrane inside is said to be elementary.
- $\omega_i \,|1<=i<=m$ are strings over *U*, representing multisets of objects placed inside the membrane with label *i*.
- $R_i \,|1<=i<=m$ are finite sets of evolution rules associated to the membrane with label *i*. Rules have the form $u \rightarrow v$, $u \rightarrow v\delta$ or $u \rightarrow v\tau$, with $u \in U^+$ and $v \in (U^+ \times TAR)^*$, where *TAR={here, out}* $\cup$ *{in$_j$ | 1 <= i <= m}*. Symbol δ represents membrane dissolution, while symbol τ represents membrane inhibition. ρ_i, *1 <= i <= m*, are priority relations defined over R_i, the set of rules of membrane *i*.
- i_0 represents the label of the membrane considered as output membrane.

Evolution rules able to be applied in any evolution step of the P system must accomplish three requisites at that moment: useful, applicable and active. A rule r_j in membrane *i* is useful in a evolution step if all targets are adjacent to membrane *i* and not inhibited. For example in figure 2, evolution rule r_4 in membrane 1 has membrane 4 as a target, and then it is not useful at the beginning. But if membrane 2 is dissolved then membrane 4 and 5 become adjacent to membrane 1, and rule r_4 useful. On the other hand, a useful rule r_j is applicable in membrane *i* if its antecedent is included in the membrane multiset. Finally, an applicable rule r_j is active in membrane *i* if there is no other applicable rule with higher priority in membrane *i*. Active rules will be applied in a parallel and non-deterministic way in the current evolution step.

Obtaining useful rules from usefulness states

The usefulness state concept at membranes of a P System was introduced in [Frutos, 2007]. This state allows any membrane to know the set of child membranes with which communication is feasible, that is to say, adjacent and not inhibited membranes. This set of child membranes constitute the membrane context, which changes dynamically as membranes are dissolved or inhibited in the P system.

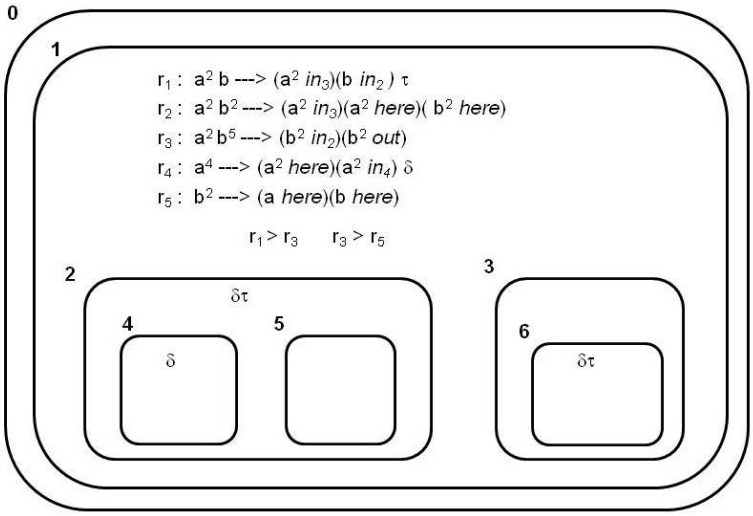

Fig.2. Example of transition P System

The set of usefulness states for a membrane *i* in a Transition P system can be obtained statically at analysis time, as it is shown in [Frutos, 2007]. A usefulness state *j* in membrane *i*, q_j^i , represents a valid context for that membrane, $C(q_j^i)$, which means a context that can be reached in any evolution step. Figure 2 represents an example of Transition P system. Only rules associated to membrane 1 are detailed. Symbol δ in membranes 2, 4 and 6 represents the possibility of these membranes to be dissolved by application of some rules inside them. The symbol τ represents the possibility of inhibition for membranes 2, 3 and 6 by the same cause. Usefulness states for membrane 1, together with their corresponding contexts, are depicted in first and second column of table 1.

Usefulness states	Context	Useful rules	Useful rules when permeable
q_0^1	{2,3}	r_1, r_2, r_5	r_3
q_1^1	{4,5,3}	r_2, r_4, r_5	
q_2^1	{5,3}	r_2, r_5	
q_3^1	{3}	r_2, r_5	

Table 1. Usefulness states for membrane 1

From a given usefulness state q_j^i , it can be obtained statically the set of useful rules in the following way:

- An evolution rule $u \to v\xi, where\ \xi \in \{\delta, \tau, \lambda\}$ is useful in q_j^i if $\forall\, t\,\text{arg}et\ in_k \in v, k \in C(q_j^i)$, what it means that all child targets are included in the context.
- In addition, If target *out* belongs to *v* then the evolution rule will be useful only if membrane *i* is permeable, otherwise communication with its father membrane is not feasible.

Third and fourth column in table 1 represents useful rules for every usefulness states. Rule r_3 will be useful in q_0^1 only if membrane 1 is not inhibited. To sum up, useful rules can be obtained from the usefulness state and the permeability state of the membrane.

Tables defining transitions among usefulness states can be also defined at analysis time, as it is explained in [Frutos, 2007]. During system execution, a membrane makes a transition when its context changes, i.e. a child membrane changes its permeability. In such a way that useful rules for any membrane are always available from its usefulness and permeability states.

Regarding implementations, problems arise when membranes have a high number of states, which causes transition tables to grow up. That is why in [Frutos, 2007] transition tables are avoided by encoding usefulness states; hence transitions are carried out directly in the code. According to this idea, an important definition is introduced:

Total Context for membrane ***i***. It is the set made up of all membranes that eventually can become children of membrane i. Therefore, all contexts are included in the total context.

$$TC(i) = Children(i) \bigcup_{i_k \in ChildrenDis(i)} TC(i_k)$$

Children of a membrane *i* is the set of all children in μ, the initial structure; and *ChildrenDis* for a membrane *i* is the set of all children in μ that can be dissolved. For instance, in our example of figure 2, TC(1) = {2, 4, 5, 3}.

Each one of usefulness states for a membrane *i* is encoded by *TC(i)*, depending on its context, with binary logic. The value 1 represents that the membrane belongs to the state context. Thus, the usefulness state q_0^1 in our example, which represents the context {2, 3}, is encoded as 1001. Transitional logic among usefulness states is described in [Frutos, 2007] without making use of transition tables. Besides, we want to emphasize the work carried out in [Frutos, 2008], which describes in an exhaustive way an implementation for usefulness states updating in some architectures based on a cluster of processors.

Decision trees for active rules

A decision tree is a tool that allows determining the class which one element belongs to, depending on the values of some attributes or properties of the element. Each non-leaf node of a decision tree corresponds to an input attribute, and each arc to a possible value of that attribute. A leaf node corresponds to the expected value of the output attribute, that is to say, the element classification. An element is classified by starting at the root node of the decision tree, testing the attribute specified by this node and moving down the tree branch corresponding to the value of the attribute. This process is repeated until a leaf node is reached.

There are a lot of algorithms to generate decision trees. Specifically ID3 is an outstanding algorithm belonging to TDIDT family (Top-Down Induction of Decision Trees). It is based on entropy information, a measure of uncertainty. Entropy is used to determine how informative a particular input attribute is about the output attribute. By using this, algorithm ID3 generates an optimum decision tree from a non incremental set of instances and without repetitions.

E	a_1	.....	a_j	C
e_1	v_{11}	.....	v_{1j}	c_1
⋮	⋮		⋮	⋮
e_i	v_{i1}	.....	v_{ij}	c_i

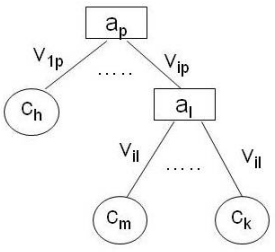

Fig. 3. Example of instances for ID3 algorithm, together with the obtained tree

Algorithm ID3 uses as input a set of data about instances such as figure 3. Each row represents an instance (e_i), which includes a value (v_{ij}) for each attribute (a_j), and the decision or classification for that instance (c_i). Then ID3 algorithm obtains a decision tree making use of entropy property of attributes. Fernandez et al. in [Fernández, 2006] proposed incorporating decision trees in the calculus of evolution rules applicability. A decision tree for a membrane is able to classify the current multiset of objects in the membrane, obtaining the subset of applicable rules for the current evolution step. In this way, it is reduced the number of checks necessary for determining the subset of applicable rules. Nevertheless, Fernandez et al. do not take into account conditions for rules to be useful and active, which are also necessary to determine the subset of rules that can be applied in an evolution step. The present work, supporting in usefulness states analysis, tries to extend the advantage of decision trees. In such a way that a decision tree for a membrane obtained at analysis time can determine the subset of active rules at execution time. With this aim we make the following considerations:

- The elements to be classified are all different states or situations of the membrane in the P system at any time. These situations are characterized by:
  - The membrane usefulness state, due to the set of useful rules depends on it.
  - The membrane permeability state, due to the set of useful rules depends on it. However, only information about inhibition is needed.
  - The membrane multiset in order to determine the set of applicable rules.

  Active rules subset is determined from applicable rules bearing in mind priority relations among them. Nevertheless, these relations are static, that is to say, they do not depend on the state of the P system.
- The set of attributes A_i is established as properties necessary to define a situation of the membrane *i* in the P System. Specifically, the set A_i consist of the following attributes:
  - Attributes necessary to set up the usefulness state of membrane *i*. Thus, there will be one attribute for each membrane belonging to membrane *i* total context. The associated value will be true if the represented child membrane belongs to the current usefulness state.

    $$A_i \supset \{a \equiv m_j \mid j \in TC(i)\}$$

  - One attribute more to determine inhibition in the permeability state of membrane *i*. The value true will be used when membrane *i* is inhibited.

    $$A_i \supset \{a \equiv I\}$$

  - Furthermore, as proposed in [Fernández, 2006], we consider another set of attributes for applicability of rules. These attributes represents the set of weight checks between objects from the membrane multiset and objects from antecedent of every evolution rule. Neither repetitions nor checks with zero are considered. An attribute value will be true in case of the multiset weight is greater or equal to antecedent weight in the evolution rule.

    $$A_i \supset \{a \equiv \mid \omega \mid_u \geq k \;\; \mid \;\; \mid input(r) \mid_u = k \wedge k \neq 0 \;\; \forall r \in R \; \forall u \in U\}$$

    $\mid \omega \mid_u$ represents the weight of a symbol *u* in ω, and *input(r)* represents the antecedent multiset of rule *r*.
- Finally, classification will be carried out into subsets of evolution rules, i.e. a membrane *i* state will be classified into the corresponding subset of active rules. Hence, classes to be considered are:

  $$C = \{c \equiv R_{AC} \mid \; R_{AC} \in \mathrm{P}(R)\}$$

To obtain a decision tree in analysis time under these conditions, we have to start from a set of data, representing instances. Every instance consists of the corresponding attributes and classification. Besides, training data will be

complete, that is, every possible membrane situation will be represented as an instance. Thus, the decision tree will be used for classifying the current membrane state, and the resulting class will be correct anyway.

As every possible membrane state has to be considered, and taking into account the set of attributes A_i, the amount of instances in the training data for membrane *i* is the following:

$$|E_i| = |Q_i| * \prod_{u \in U} \left(|C_i^u| + 1 \right)$$

Where $|Q_i|$ is the number of usefulness states for membrane *i*, and $|C_i^u|$ is the number of different checks with symbol *u* in any rule antecedent of membrane *i*, that is, attributes with the form $|\omega|_u >= k$. Besides, if membrane *i* has inhibiting capability, this value has to be multiplied by 2 in order to consider attribute *I*. Anyway, what it is important to note is that the needed information for every instance is available at analysis time.

Coming back to the example of P system introduced in figure 2, attributes for membrane 1 are the following:

- As *TC(1)* = {2,4,5,3}, four attributes are needed: m_2, m_4, m_5, m_3.
- As membrane 1 has inhibiting capability by means of r_1, an attribute *I* has to be included.
- Finally, the set of different checks for applicability in evolution rules are: $|\omega|_a >= 4$, $|\omega|_a >= 2$, $|\omega|_b >= 5$, $|\omega|_b >= 2$ and $|\omega|_b >= 1$. Therefore, we consider another five attributes.

The resulting training data are shown in table 2. Each instance is named with the corresponding usefulness state, followed with *I* (inhibited) or *NI* (not inhibited), and finally the applicability state. As example, $1001NIa^2b^5$, represents the instance with the following properties for membrane 1: the usefulness state is *1001* (membranes 2 and 3 as context), the permeability state is permeable, the amount of objects *a* in *ω* is greater than 2, but not greater than 4; and finally, there are 5 or more objects *b* in *ω.*

E	m_2	m_4	m_5	m_3	*I*	$\|\omega\|_a >= 4$	$\|\omega\|_a >= 2$	$\|\omega\|_b >= 5$	$\|\omega\|_b >= 2$	$\|\omega\|_b >= 1$	*C*
$1001NIa^0b^0$	Yes	No	No	Yes	No	No	No	No	No	No	∅
$1001NIa^0b^1$	Yes	No	No	Yes	No	No	No	No	No	Yes	∅
$1001NIa^0b^2$	Yes	No	No	Yes	No	No	No	No	Yes	Yes	$\{r_5\}$
$1001NIa^0b^5$	Yes	No	No	Yes	No	No	No	Yes	Yes	Yes	$\{r_5\}$
$1001NIa^2b^0$	Yes	No	No	Yes	No	No	Yes	No	No	No	∅
$1001NIa^2b^1$	Yes	No	No	Yes	No	No	Yes	No	No	Yes	$\{r_1\}$
$1001NIa^2b^2$	Yes	No	No	Yes	No	No	Yes	No	Yes	Yes	$\{r_1,r_2\}$
$1001NIa^2b^5$	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes	$\{r_1,r_2\}$
$1001NIa^4b^0$	Yes	No	No	Yes	No	Yes	Yes	No	No	No	∅
$1001NIa^4b^1$	Yes	No	No	Yes	No	Yes	Yes	No	No	Yes	$\{r_1\}$
$1001NIa^4b^2$	Yes	No	No	Yes	No	Yes	Yes	No	Yes	Yes	$\{r_1,r_2\}$
$1001NIa^4b^5$	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	$\{r_1,r_2\}$
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Table 2. Instances of membrane 1 for ID3 algorithm

In table 2 only a few number of instances for membrane 1 are represented. The total amount can be worked out as it is shown below:

$$|E_1| = 2 \cdot |Q_1| \cdot \prod_{u \in U} \left(|C_1^u| + 1 \right) = 2 \cdot |Q_1| \cdot (|C_1^a| + 1) \cdot (|C_1^b| + 1) = 2 \cdot 4 \cdot 3 \cdot 4 = 96 \text{ instances}$$

Instance classification

Every instance is classified into the corresponding set of active rules. This task is carried out at analysis time as:

$$C = Max\ (Useful\ Rules \cap Applicable\ Rules)$$

- Firstly, useful rules are obtained from the current usefulness state and the current permeability state, as it is shown in table 1.
- Secondly, attributes related to checks of objects weights in multiset determine the applicability property of rules, as it is shown in [Fernandez, 2006]. The intersection with the set of useful rules gets the set of useful and applicable rules (*C'*).
- Lastly, priorities among rules have to be considered in order to get the active rules subset, which implies to work out the maximal over the priority relation of *C'*. With this aim, we have to work out a transitivity matrix (*M*) expressing the priority relation. Then *C'* has to be multiplied by *M*, obtaining the set of rules with less priority that any rule in *C'*. Therefore, the maximal of *C'* is obtained by removing those rules:

$$C = Max(C') = C' \wedge \neg\ (C' \times M)$$

For example, we are going to classify instance $1001NIa^2b^5$. First, as usefulness state is *1001* the corresponding state in table 1 is q_0^1. With normal permeability, the set of useful rules is $\{r_1, r_2, r_3, r_5\}$. Second, with two or more symbols *a* (but not four or more) and five or more symbols *b*, the set of applicable rules is $\{r_1, r_2, r_3, r_5\}$. Therefore, intersection between useful an applicable rules is $C' = \{r_1, r_2, r_3, r_5\}$.

With regard to priorities, membrane 1 has two relations: $r_1 > r_3$ and $r_3 > r_5$, obtaining the following transitivity matrix *M*:

	r_1	r_2	r_3	r_4	r_5
r_1	0	0	1	0	1
r_2	0	0	0	0	0
r_3	0	0	0	0	1
r_4	0	0	0	0	0
r_5	0	0	0	0	0

Now we represent C' = $\{r_1, r_2, r_3, r_5\}$ with binary logic, obtaining *11101*. Then C is obtained as follows:

$$C = Max(11101) = (11101) \wedge \neg((11101) \times M) = 11000 \quad \text{----->} \quad \{r_1, r_2\}.$$

As conclusion, all necessary data for every instance are available at analysis time in the P system. Thus ID3 algorithm can be applied obtaining a decision tree, which determines classification of any situation of the P system into an active rules subset. Specifically, decision tree corresponding to membrane 1 in our example is depicted in figure 4.

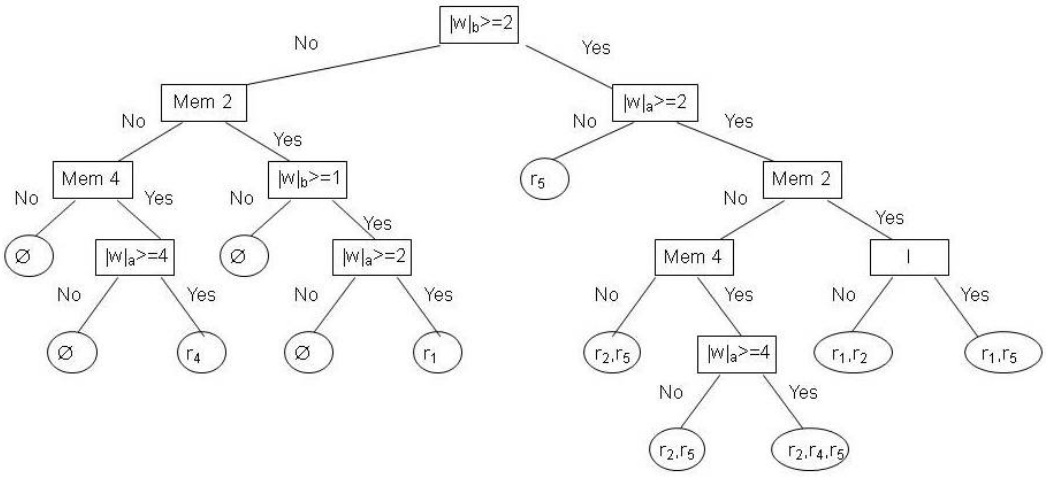

Fig.4. Decision tree obtained by ID3 algorithm for membrane 1

A software implementation for decision trees

Computation of active evolution rules subset is critical for the whole evolution process efficiency, because it is performed inside each membrane in every evolution step. That is why software implementation should be carried out in a low level with an assembly language. Efficiency is considerably increased, mainly by using registers in a convenient way. PC assembly language utilizes some registers with general purpose, such as EAX, EBX, ECX, EDX, with 32 bits. Below we show how these registers are used in our implementation.

EAX: it will be used to return the subset of active rules with binary logic. Coming back to membrane 1 of our example in figure 1, only five lower bits will be used for representing rules r_5, r_4, r_3, r_2 and r_1, respectively.

EBX: usefulness state and permeability state will be stored in this register. In this way, only six lower bits of EBX will be used for membrane 1 in our example. Four of them for storing the current usefulness state; and the last two bits for membrane inhibition and dissolution respectively, as it is represented in figure 5.

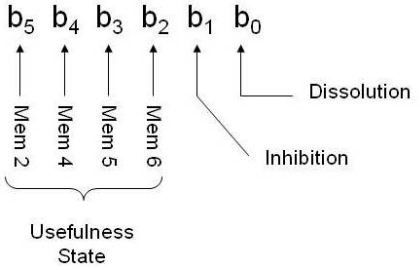

Fig. 5. Usefulness and permeability states in EBX register.

ECX: it will be used for storing ω, the membrane multiset. Specifically, in our example:

- CH (higher 16 bits) will contain $|\omega|_a$, the amount of symbols *a* in ω.
- CL (lower 16 bits) will contain $|\omega|_b$, the amount of symbols *b* in ω.

```
    CMP CL, 2          ; |w|b>=2 ?
    JAE et1
    MOV EAX, 0000H     ; ∅
    TEST EBX, 0020H    ; Mem 2 ?
    JNZ et2
    TEST EBX, 0010H    ; Mem 4 ?
    JZ et3
    CMP CH, 4          ; |w|a>=4 ?
    JB et3
    MOV EAX, 0008H     ; {r4}
et3 RET
et2 CMP CL, 1          ; |w|b>=1 ?
    JB et4
    CMP CH, 2          ; |w|a>=1 ?
    JB et4
    MOV EAX, 0001H     ; {r1}
et4 RET
```

```
et1 MOV EAX, 0010H     ; {r5}
    CMP CH, 2          ; |w|a>=2 ?
    JB et5
    TEST EBX, 0020H    ; Mem 2 ?
    JNZ et6
    MOV EAX, 0012H     ; {r2, r5}
    TEST EBX, 0010H    ; Mem 4 ?
    JZ et7
    CMP CH, 4          ; |w|a>=4 ?
    JB et7
    MOV EAX, 001AH     ;{r2,r4,r5}
et7 RET
et6 MOV EAX, 0003H     ; {r1, r2}
    TEST EBX, 0002H    ; I ?
    JZ et5
    MOV EBX, 0011H;    ; {r1, r5}
et5 RET
```

Fig. 6. Assembly code for the membrane 1 decision tree

Taking into account these conditions, an algorithm can be generated automatically for a given decision tree at analysis time. Going on with the decision tree represented in figure 4, the corresponding code which obtains the active rules subset for membrane 1 can be the represented in figure 6.

In general, for implementing a leaf node of the decision tree, assembly code stores previously the suitable set of active rules in EAX. However, optimization in the code is possible. For instance, we can observe in the decision tree of figure 4 four leaves which subset is ∅, whereas only one storing instruction for that subset in EAX can be appreciated in the assembly code. This improvement can be achieved by carrying out a previous analysis into the decision tree. The goal is to determine the best points in decision tree for loading an active rules subset in EAX. This analysis is performed with a two-way algorithm. Firstly, the decision tree is analyzed upward, carrying out the process represented in figure 7 in every non-leaf node *i*.

```
j <--- Child_Left (i)
k <--- Child_Right (i)
IF Is_Leaf(j) OR Is_Leaf(k) THEN BEGIN
        IF Is_Leaf(j) THEN EAX(i) <--- Active_Set(j)
        IF Is_Leaf(k) THEN EAX(i) <--- EAX(i) +
Active_Set(k)
    END
ELSE EAX(i) <--- Common_Sets (EAX(j), EAX(k))
```

Fig. 7. Bottom up algorithm for determining subsets to be preloaded in EAX

After performing this process over the decision tree in figure 4, results are shown in figure 8. In every non-leaf node it is represented subsets of active rules that could be stored in EAX when implementing that node. Now, decision tree is analyzed from root to leaves choosing a subset in every non-leaf node. When a non-leaf node is analyzed, it is important to take into account the subset already loaded in the upper node. If this subset is considered as a possible subset to be loaded in the current node, then no loading is performed. Results are depicted in figure 9, corresponding with the assembly code in figure 6.

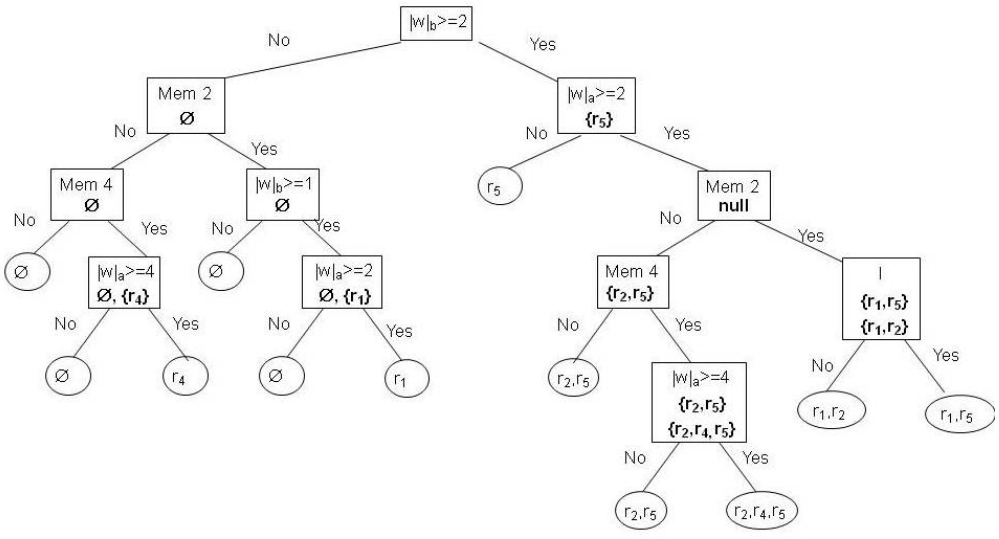

Fig. 8. Bottom up analysis in decision tree for membrane 1

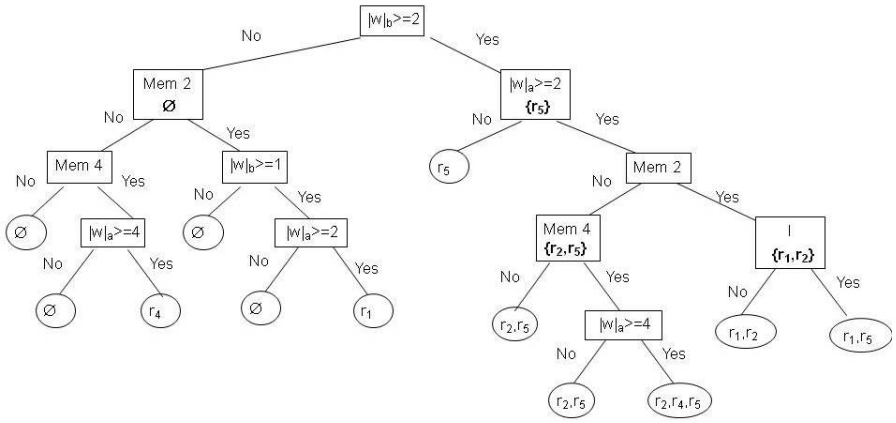

Fig. 9. Top down analysis in decision tree for membrane 1

To sum up, a P system is statically analyzed, obtaining a particular assembly code for each membrane. As result, this software is a suitable solution for architectures based on a cluster of processors, such as [Syropoulos, 2003], [Tejedor, 2007], [Bravo, 2007a], [Bravo, 2007b] and [Bravo, 2008]. In all these architectures a membrane evolves in a single process. Such process would contain the assembly code obtained as proposed in this paper; and the membrane would get active rules from the decision tree in every evolution step.

As regards architecture based on microcontrollers PIC16F88, proposed by Gutierrez et al. in [Gutierrez, 2006], [Gutierrez, 2007] and [Gutierrez, 2008], we have to bear in mind that these devices are software programmed. More precisely, in [Gutierrez, 2007] Gutierrez et al. made use of the Microchip MPLAB IDE integrated Environment, in which the tool MPASM allows to work with assembly language. Therefore, this architecture could get profit from the solution proposed in this paper. Besides, this solution could especially fit here, due to the lack of memory for data in microcontrollers. The implementation proposed in this paper does not require space for additional data; moreover, it reduces data by avoiding transitivity matrix for priorities.

Analysis of results

This section presents an analysis of efficiency in the proposed solution based on decision trees. In this way, we are going to compare it with classical solutions for obtaining active rules. This kind of solutions consists of three sequential algorithms, obtaining useful, applicable and active evolution rules respectively.

Classical algorithm for useful rules (Figure 10). Complexity order is *O(n)*, where *n* represents the amount of checks. In the worst case $n = |R| * (|TC| + 1)$. For every evolution rule algorithm must check every target in the membrane usefulness state. Additionally, it has to be checked for every evolution rule if there is a target *out* when the membrane is inhibited. As usefulness state length is equal to total context length, it has to be performed $|TC| + 1$ checks for every rule.

```
R_U <--- ∅
FOR EACH r_i in R DO BEGIN
    j <--- 1
    useful <--- true
    WHILE j <= length (TC) AND useful DO BEGIN
        IF (TC(j) ⊂ TAR(r_i) ) AND UsefulnessState(j) = 0      (* Impossible target *)
                        THEN useful <--- false
        j <--- j + 1
    END
    IF useful THEN
        IF (out ⊄ TAR(r_i) ) OR (PermeabilityState ≠ Inhibited)
            THEN R_U <--- R_U ∪ { r_i }
END
```

Fig 10. Algorithm for obtaining useful rules

Classical algorithm for applicable rules (Figure 11). It is also an algorithm with *O(n)* complexity. In this case, the amount of checks is equal to $|R|*|U|$ in the worst case. For every rule, the algorithm must check the weight of every symbol in the multiset.

```
R_A <--- ∅
FOR EACH r_i in R_U DO BEGIN
    applicable <--- true
    FOR EACH u in U DO
        IF (| input(r_i) |_u > | ω |_u ) THEN BEGIN      (* Not enough symbols in ω *)
            applicable <--- false
            break
        END
    END
IF applicable THEN R_A <--- R_A ∪ { r_i }
END
```

Fig. 11. Algorithm for obtaining applicable rules

Classical algorithm for active rules. As it is detailed in a previous section, this algorithm calculates the maximal over the priority relation of useful and applicable rules. With this aim, it firstly multiplies a vector with these rules by the transitivity matrix expressing priorities. Then, the resulting subset of rules has to be eliminated, obtaining the corresponding maximal. Multiplication complexity is $O(n^2)$, where *n* is equal to $|R|$; meanwhile complexity for subtracting rules is *O(n)*, where *n* is also $|R|$.

As regards solution based on decision trees proposed in this paper, the implemented algorithm starts in the root and finishes in a leaf. Therefore computational complexity of the algorithm is *O(n);* *n* represents the amount of check operations, that in the worst case it is the longest branch of the decision tree. At the most, a branch length will be the amount of attributes considered in ID3 algorithm, due to each step of ID3 algorithm chooses an attribute from the subset of not chosen attributes. Then, the greatest amount of checks is equal to $|TC|+1+\sum_{u\in U}|C_i^u|$ i.e. the amount of attributes. Table 3 summarizes these results.

	Useful rules	Applicable rules	Active rules
Classical algorithms	$O(n)$ $n=\|R\|*(\|TC\|+1)$	$O(n)$ $n=\|R\|*\|U\|$	$O(n^2)$ + $O(n)$ $n=\|R\|$
Decision tree	$O(n)$ $n=\|TC\|+1+\sum_{u\in U}\|C_i^u\|$		

Table 3. Complexity order of algorithms.

From these results, we can conclude that decision tree solution performs a fewer number of operations than classical solutions, as it is shown in the following reasoning:

- Decision tree solution performs $|TC|+1$ check operations for obtaining the useful rules subset, meanwhile classical algorithm requires $|R|*(|TC|+1)$ check operations.

- On the other hand, for applicability of rules, decision tree performs $\sum_{u \in U} |C_i^u|$ checks in the worst case, avoiding repeated and redundant checks. For instance, decision tree will never check if $|\omega|_a \geq 3$ after determining that $|\omega|_a \geq 5$. So, the amount of check operations in decision trees will never be greater than $|R| * |U|$.
- There is an obvious difference between both solutions in the analysis of priorities for obtaining active rules. Decision tree does not need any operation, whereas classical algorithms must multiply by the transitivity matrix.

In the following, we are going to apply the previous comparative to a set of P systems published in [Păun, 2000a] and [Păun, 2000b], which are listed in table 4. For every P system, table 4 shows the amount of operations to be performed by classical algorithms and decision tree respectively. Values of the table refer to the addition of the values for every membrane of the P system; and considering always the worst case.

	Classical Algorithms			Decision Tree
P-System	Amount of checks for useful rules	Amount of checks for applicable rules	Amount of operations for active rules	Amount of checks for active rules.
First example in [Păun, 2000a]	24	24	20	10 (15'62%)
Decidability (*n mod k = 0*) in [Păun, 2000a]	6	15	12	7 (21,21%)
Generating n^2 | *n>=1* in [Păun, 2000a]	15	18	20	6 (11,32%)
Generating n^2 | *n>=1* in [Păun, 2000b]	14	19	20	7 (13,20%)

Table 4. Amount of operations in classical algorithms versus decision trees

From table 4 we conclude that decision trees carry out a fewer amount of operations. Even if we consider only check operations, classical algorithms need a greater amount of them. If we also consider operations that classical algorithm perform for applying priorities, the resulting value is significantly lower in decision trees. Specifically, the total amount of operations in decision trees vary from 11,32% to 21,21% of the total amount of operations in classical algorithms. Besides, as the code for decision trees has been specifically optimized for each membrane, we can conclude that decision trees allow a significant reduction in the required time for applying rules inside membranes.

Finally, we are going to add some remarks about memory space required by both types of solutions.

- A decision tree handicap is the need to keep a different code for each membrane; meanwhile classical algorithms make use of a common code. Thus, in architectures based on a cluster of processors, each processor must house as many decision trees as membranes are allocated on it.
- On the other hand, a decision tree advantage is that it does not require space for the transitivity matrix expressing priorities.

- Also, we have to mention that decision tree could grow up significantly. For instance, a membrane with a great deal of rules could mean a great deal of attributes in table for ID3 algorithm; and consequently, a big decision tree. In this case, it may happen that, depending on the architecture used for implementing the P system, memory space could be insufficient; and then, classical algorithms had to be chosen. Anyway, as decision tree code is obtained at analysis time, the best solution for a concrete P system can be also determined at analysis time.

Conclusions

We have proposed a static analysis of transition P systems in order to get a specific code for each membrane. This code can determine the subset of active rules in every evolution step in an optimal way. The analysis we propose consists mainly of determining the set of usefulness states for each membrane. These states are shown in [Frutos, 2007] and [Frutos, 2008]; Furthermore, the analysis considers all situations of applicability for rules, depending on antecedents, as it is explained in [Fernández, 2006]; The analysis also takes into account priorities among rules. As a result, a decision tree is obtained that gathers all information collected during the analysis. In the end, the decision tree is transformed into an optimized assembly code. This also occurs during the analysis.

Some architecture for implementing transition P systems can take advantage of this implementation such as most of the architectures based on a cluster of processors and architectures based on microcontrollers.

An analysis has been carried out in this paper, in which we conclude that decision tree is significantly more efficient than classical solutions for determining active rules. According to the work carried out by Tejedor et al. in [Tejedor, 2007], it is possible to obtain improvements in some architectures based on a cluster of processors. Improvements such as reduction of the total time for an evolution step increase of the number of membranes that could run in a processor and reduction of the required processors.

Bibliography

[Bravo, 2007a] G. Bravo, L. Fernández, F. Arroyo. J.A. Frutos, A Hierarchical Architecture with Parallel Communication for Implementing P systems, Fifth International Conference Information Research and Applications (i. TECH 2007), June 2007, Varna, Bulgary, 168-174.

[Bravo, 2007b] G. Bravo, L. Fernández, F. Arroyo. J. Tejedor, Master Slave Distributed Architecture for Membrane Systems Implementation, 8th WSEAS Int. Conf. on Evolutionary Computing (EC'07), June 2007, Vancouver, Canada.

[Bravo, 2008] G. Bravo, L. Fernández, F. Arroyo, M. A. Pea, Hierarchical Master-Slave Architecture for Membrane Systems Implementation, 13th Int. Symposium on Artificial Life and Robotics (AROB '08), Feb 2008, Beppu, Oitia (Japan).

[Ciobanu, 2004] G.Ciobanu, W. Guo, P Systems Running on a Cluster of Computers. Workshop on Membrane Computing (Gh. Păun, G. Rozemberg, A. Salomaa Eds.), 2004, LNCS 2933, Springer, 123-139

[Fernández, 2006] L. Fernández, F. Arroyo, I. García, G. Bravo, Decision Trees for Applicability of Evolution Rules in Transition P Systems, Fourth International Conference Information Research and Applications (i. TECH 2006) June 2006, Varna, Bulgary.

[Frutos, 2007] J. A. Frutos, L. Fernández, F. Arroyo, G. Bravo, Static Analysis of Usefulness States in Transition P Systems, Fifth International Conference, Information Research and Applications (I.TECH 2007), June 2007, Varna, Bulgary. 174-182.

[Frutos, 2008] J. A. Frutos, F. Arroyo, A. Arteta. Usefulness States in New P System Communication Architectures, Ninth Workshop on Membrane Computing (WMC9), July 2008, Edinburgh, Scotland.

[Gutierrez, 2006] A. Gutierrez, L. Fernández, F. Arroyo, V. Martínez, A Design of a Hardware Architecture Based on Microcontrollers for the Implementation of Membrane Systems, 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, 2006

[Gutierrez, 2007] A. Gutierrez, L. Fernández, F. Arroyo, S. Alonso, Hardware and Software Architecture for Implementing Membrane Systems: A Case of Study to Transition P Systems, The DNA International Meeting on DNA Computing (DNA13), June 2007, Memphis, USA.

[Gutierrez, 2008] A. Gutierrez, L. Fernández, F. Arroyo, Suitability of Using Microcontrollers in Implementing New P System Communication Architectures, 13th Int. Symposium on Artificial Life and Robotics (AROB '08), Feb. 2008, Beppu, Oitia (Japan).

[Păun, 2000a] Gh. Păun, Computing with Membranes, Journal of Computer and System Sciences, 61(1), 2000, 108-143.

[Păun, 2000b] Gh. Păun, G. Rozemberg, A Guide to Membrane Computing, Theoretical Computer Science, vol 287, 2000, 73-100.

[Syropoulos, 2003] A.Syropoulos, E.G. Mamatas, P.C. Alliomes, K.T. Sotiriades, A Distributed Simulation of P Systems. Workshop on Membrane Computing, Tarragona, Spain, 455-460.

[Tejedor, 2007] J. Tejedor, L. Fernández, F. Arroyo, G.Bravo, An Architecture for Attacking the Bottleneck Communication in P Systems, 12th Int. Symposium on Artificial Life and Robotics, Jan 2007, Beppu, Oita, Japan, 500-505.

Authors' Information

Juan Alberto de Frutos Velasco *– Dpto. Lenguajes, Proyectos y Sistemas Informáticos (LPSI) de la Escuela Universitaria de Informática (EUI) de la Universidad Politécnica de Madrid (UPM); Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: jafrutos@eui.upm.es*

Luis Fernández Muñoz *– Dpto. Lenguajes, Proyectos y Sistemas Informáticos (LPSI) de la Escuela Universitaria de Informática (EUI) de la Universidad Politécnica de Madrid (UPM); Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: setillo@eui.upm.es*

Carmen Luengo Velasco – *Dpto. Lenguajes, Proyectos y Sistemas Informáticos (LPSI) de la Escuela Universitaria de Informática (EUI) de la Universidad Politécnica de Madrid (UPM); Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: cluengo@eui.upm.es*

Alberto Arteta Albert – *Dpto. Lenguajes, Proyectos y Sistemas Informáticos (LPSI) de la Escuela Universitaria de Informática (EUI) de la Universidad Politécnica de Madrid (UPM); Ctra. Valencia, km. 7, 28031 Madrid (Spain); e-mail: aarteta@eui.upm.es*