

Representing Multisets and Evolution Rules in Membrane Processors *

Fernando ARROYO, Carmen LUENGO

Dpto. de Lenguajes, Proyectos y Sistemas Informáticos
Escuela Unversitaria de Informática
Universidad Politécnica de Madrid
E-mail: {farroyo; cluengo}@eui.upm.es
<http://www.lpsi.eui.upm.es>

Juan CASTELLANOS

Dpto. de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
E-mail: jcastellanos@fi.upm.es
<http://www.dia.fi.upm.es>

Luis F. MINGO

Dpto. de Organización y Estructura de la Información
Escuela Unversitaria de Informática
Universidad Politécnica de Madrid
Crta. de Valencia Km. 7
<http://www.oei.eui.upm.es>

Abstract

Membrane Processors have been defined as digital computational devices able to perform the local processes developed at the membrane level in Transition P systems. This paper is related to describing the way of representing multisets of objects and evolution rules in membrane processors. The formal study of evolution in regions is the way to understand the behavior of the membrane processor because it permits to define the basic needed operations in order to design the way in which multisets and evolution rules can be represented in a more accurate way to be implemented in digital processors. From these new representations of object multiset and evolution rules, the set of basic operation over them will be defined. Finally, the machine instructions for membrane processors are presented, considering the evolution rules as the machine instructions for membrane processors.

*Work partially supported by contribution of EU commission under the Fifth Framework Programme, project "MolCoNet" IST-2001-32008.

1 Introduction

Since Gheorghe Păun introduced in 1998 [1] Membrane Computing, this research field has experimented a very important growth.

P systems are models of computation which are inspired by basic features of biological membranes. Membranes define compartments or regions. Multisets of objects and evolution rules are then placed inside regions. Objects evolve by means of evolution rules which are applied in a maximally parallel, nondeterministic manner. Objects can pass through membranes, and membranes can be dissolved. These are the main features used in defining P systems. Many variants of P systems have been investigated and most of them are computationally universal. In the meanwhile, some attempts to implement such computational devices in software and hardware have been reported.

Related to hardware implementations of P systems, membrane processors were introduced in [8] as digital devices that implement processes developed in regions of transition P systems. In this framework, membrane processors trees are an approach to develop a general purpose architecture which will permit to implement transition P systems in hardware. Some problems related to represent membrane structure and to keep it update during evolution in membrane processors trees were presented and solved in that paper.

Next sections are devoted to determine the way in which P systems compute in order to define what will be data and instructions able to be represented in a digital processor like traditional digital computers. Hence, we will start by defining in a classical way computations in P systems and then operational modes for membrane processors are presented; a feasible machine representation for multisets of objects and evolution rules is described. Finally, the machine instruction syntax and the processing algorithm for membrane processors are presented.

2 Transition P systems as Computational Devices

P systems compute starting from an initial configuration and passing from one configuration to another by the application of evolution rules to objects placed inside regions until reaching a halting configuration. This configuration is defined as a configuration in which there is no rule applicable to the objects present in the system.

A configuration is defined as a $(k + 1)$ -tuple $(\mu, \omega_{i_1}, \dots, \omega_{i_k})$, where μ is the membrane structure and ω_{i_j} is the multiset of objects associated to the region defined by the membrane with label i_j in the system [2].

A transition in a P system is got by passing from a configuration to another configuration. More precisely, given two configurations $C_1 = (\mu', \omega'_{i_1}, \dots, \omega'_{i_k})$ and $C_2 = (\mu'', \omega''_{j_1}, \dots, \omega''_{j_l})$ of a P system, Π , it is said that there is a transition from C_1 to C_2 and it is represented by $C_1 \Rightarrow C_2$, if we can pass from C_1 to C_2 by using the evolution rules from R_{i_1}, \dots, R_{i_k} in the regions i_1, \dots, i_k .

From this definition of computation in P systems, two different levels of processing are defined:

1. The global process performed by the P system. This process is got by transi-

tions between two consecutive configurations.

2. The local process performed at the membrane level. This process is responsible of making evolve each region of the system by the application of evolution rules.

The global processing is achieved from the local processing by communication and synchronization among the different membranes of the system.

Local processing is achieved inside regions, which are defined by the membranes of the system. For every region in the system, the following steps are performed in order to achieve local processing [4], [5]:

1. The computation of a complete multiset of evolution rules [3]. This will permit to evolve the region by application of only one evolution rule and will determine the multiset of objects that will be send to the adjacent regions and the resulting multiset of the region.
2. The synchronization and the objects communication among adjacent regions.
3. The rearrangement of the membrane structure. Some of the membranes can be dissolved during evolution. This means that synchronization and communication are needed again in order to compute the new system configuration.

In order to characterize the local processing, let us consider a transition P system Π with labels in L and objects in U , let reg be a region in Π and let (reg, Π') be the sub-tree of Π starting in reg .

Evolution inside regions of Π is characterized by the following equations:

$$LABELS = \{l \in L | ((l, -, -), -) \in \Pi'\} \quad (1)$$

$$ACTIVES = Active(Applicable((Useful R) LABELS)\omega)\rho \quad (2)$$

$$COMPLETES = (Multicomplete ACTIVES)\omega \quad (3)$$

Here, $LABELS$ defines the set of adjacent regions to reg , $ACTIVES$ defines the set of evolution rules that can be applied in the region reg , $COMPLETES$ defines the set of complete multisets of evolution rules for the region reg (the elements of $COMPLETES$ define evolution rules able to make evolve the region to its next configuration). For a complete description of these equations and the whole evolution process we refer to [3, 5].

3 Membrane Processors for Transition P Systems

With the above considerations, a membrane processor can be defined as a digital processor able to implement the membrane local processing performed inside Transition P systems membranes.

Hardware implementation of transition P systems using membrane processors implies to define a membrane processor tree which keeps the same adjacent relations among membranes in the transition P system and in the membrane processor tree.

One problem that is faced in this approach is the dynamic rearrangement of the membrane structure during evolution. This problem does not have an easy solution

in hardware. One possible solution is to identify two different operational modes in membrane processors. These operational modes are:

- the *active mode*, when the membrane is alive in the system.
- the *passive mode*, when the membrane is not alive in the system.

While a membrane processor is operating in its active mode, it is meant to make evolve the system to a new configuration. The following processes have to be performed by the processor:

1. The computation of one complete multiset of evolution rules for each evolution step. At least one complete multiset of evolution rules has to be computed from rules and the multiset of objects in the processor. The computation has to be performed according to equation (3).
2. The synchronization with others membrane processors. The processor cannot send objects to its adjacent processors before finalizing step 1.
3. Objects Communication. Once every processor in the system has finished the computation of the complete multiset of evolution rules, the processors have to send objects to their adjacent processors. This process produces objects passing through membranes and new objects came to the processors.
4. The computation of the new multisets of objects. The communication produces new multisets of objects in processors that have to be computed.
5. If the membrane dissolution is performed, then the processor has to pass to the passive mode. The following processes have to be performed:
 - (a) Synchronization with its children processors. Due to the fact that some of the children processors can dissolve their membranes too, before sending any information to the father processor, the processors must wait for information coming from children.
 - (b) The computation of the resulting object multiset. Once again, a new multiset of objects has to be computed if some of the children processors have changed to the passive mode.
 - (c) The communication with the father membrane processor. At the end of the active mode, the processor has to send to its father processor the needed information in order to rearrange the system membrane structure. This information is related to objects and connectivity as is described below.
 - i. Multiset of objects. If a membrane dissolution is performed, then the rules inside membrane are removed from the system and the contained multiset of objects is embedded into the father membrane.
 - ii. Connectivity array. The information related to membrane structure is stored into bits words. Membrane processors use connectivity array for storing different data used to rearrange the membrane structure during computation [8].

In the passive mode, a membrane processor serves as communication bridge among membrane processors. The main functionality of the processor operating in this mode is to send objects and connectivity arrays from its children processors to its father processor and viceversa. This is due to the fact that in hardware is very difficult to dynamically change physical interconnection among processors.

Until now, it has been described what membrane processors are and their different operating modes. Nevertheless, nothing has been said about how to represent data in a membrane processor. The next section provides a more accurate representation for multisets of objects and evolution rules in membrane processor. This is necessary because all performed processes at the membrane level have been described in terms of the application of evolution rules over multisets of objects, and this description is far from a feasible hardware implementation. Hence, it is needed to give to such data structures a representation closer to the hardware.

4 A Feasible Representation for Objects Multisets and Evolution Rules in Membrane Processor

This section presents data structures for representing multisets of objects and evolution rules in membrane processors. Once a data structure is defined, the set of operation for implementing the behavior of local evolution in membrane processors are also defined.

4.1 Data Structures for Representing Multisets of Objects

Let U be a finite set of objects with $card(U) = m$. Let M be a multiset over U ; for each $a \in M$ we write $M(a) = n_a$. The multiset M can be represented as an m -tuple of natural numbers (the Parikh vector associated to the multiset M with respect to U). The problem is that the Parikh vector representation depends on the order of the elements of U . To avoid this problem, an order over the set U is defined as follows.

Let U be a finite set of objects with $card(M) = m$. The set U can be represented by an ordered succession of objects in the following way:

$$\begin{aligned} \phi : \{1, \dots, m\} &\rightarrow U \\ i &\rightarrow \phi(i) = a_i \end{aligned} \tag{4}$$

Here, ϕ is a one-to-one mapping from $\{1, \dots, m\}$ to U .

Then, a multiset M over U can be represented by a vector of natural numbers of length m as follows:

$$\begin{aligned} M : \{1, \dots, m\} &\rightarrow U \rightarrow N \\ i &\rightarrow M(i) = M(\phi(i)) = M(a_i) = n_i \end{aligned} \tag{5}$$

This fact permits us to represent every multiset of objects over M as an element of N^m in a congruent manner.

With these considerations the needed data structure for representing multisets of objects over a finite set U is an array of length $m = \text{card}(U)$.

This data structure allows us to define the typical operations and relations over multisets of objects in the following easy way:

Let $M_1 = (n_{11}, n_{12}, \dots, n_{1m})$, $M_2 = (n_{21}, n_{22}, \dots, n_{2m}) \in N^m$ be two multisets of objects over U ; then the following operations are defined:

1. Addition:

$$M_1 + M_2 = (n_{11} + n_{21}, n_{12} + n_{22}, \dots, n_{1m} + n_{2m}). \quad (6)$$

2. Inclusion:

$$M_1 \subseteq_U M_2 \Leftrightarrow \forall i \in \{1, \dots, m\}, M_1(i) \leq M_2(i). \quad (7)$$

3. Subtraction: If $M_1 \subseteq_U M_2$ then:

$$M_1 - M_2 = (n_{11} - n_{21}, n_{12} - n_{22}, \dots, n_{1m} - n_{2m}). \quad (8)$$

4.2 Data Structures for Representing Evolution Rules

In this section, the appropriate data structure for representing evolution rules in membrane processors is presented. Moreover, the needed operations over evolution rules are redefined according to this new representation. After that, the machine instruction format for membrane processors is provided.

An evolution rule with labels in $L \subset N$ and objects in $U = \{a_1, a_2, \dots, a_m\}$ has been defined in [3] as a tuple (u, v, δ) where u is a multiset over U , v is a multiset over $U \times (\{out, here\} \cup \{in_j \mid j \in L\})$ and $\delta \in \{dissolve, \neg dissolve\}$. With this representation for evolution rules, several operations and functions were defined in order to characterize the parallel application of evolution rules:

- Addition of evolution rules $(r_1 + r_2)$.
- Product of a natural number with an evolution rule $(n \times r)$.
- Evolution rules inclusion $(r_1 \subseteq_U r_2)$
- Input of an evolution rule $(Input\ r)$
- Rule Dissolving Capability $(Dissolve\ r)$
- Set of membrane's labels where the rule sends a multiset of objects $(Ins\ s)$

On the other hand, a new representation for a multiset of objects closer to a hardware implementation than the original one has been provided. Now, using this representation for multisets of objects, evolution rules are represented and operations over them are also redefined.

Let Π be a transition P system with labels in $L = \{1, 2, \dots, n\} \subset N$ and objects in $U = \{a_1, a_2, \dots, a_m\}$. An evolution rule r can be represented as a natural number vector of length $(n + 2) \times m + 1$ as follows:

Firstly, the rule antecedent u is a multiset over U , therefore $u \in N^m$. Secondly, the rule consequent is a multiset over $U \times (\{out, here\} \cup \{in_j \mid j \in L\})$, however, it can be also considered as a subset of $L \times U$ and in this case the targeting is directly represented in the rule; thus, $v \in N^{m \times n}$. Finally, the rule dissolving capability is represented by $\delta \in \{0, 1\}$. In conclusion, $r \in N^{(n+2) \times m+1}$, and the representation as a vector of natural numbers is the following:

$$\begin{array}{ccccccc} (\underbrace{u_1, \dots, u_m}_{\text{antecedent}}, & \underbrace{v_{0,1}, \dots, v_{0,m}}_{\text{to the environment}}, & \underbrace{v_{1,1}, \dots, v_{1,m}}_{\text{to the processor 1}}, & \dots, & \underbrace{v_{n,1}, \dots, v_{n,m}}_{\text{to the processor n}}, & \delta) \\ \underbrace{\hspace{15em}}_{\text{consequent}} \end{array}$$

As can be seen, this data structure rearranges multisets of objects in order to send objects to each of the membrane processors of the system including the environment. Moreover, with this representation, any rule r for the P system Π belongs to $N^{(n+2) \times m+1}$.

Now, the operations previously defined for evolution rules are presented and defined using this data structure.

Let $r_1 = (u_1^1, \dots, u_m^1, v_{0,1}^1, \dots, v_{0,m}^1, v_{1,1}^1, \dots, v_{1,m}^1, \dots, v_{n,1}^1, \dots, v_{n,m}^1, \delta_1)$ and $r_2 = (u_1^2, \dots, u_m^2, v_{0,1}^2, \dots, v_{0,m}^2, v_{1,1}^2, \dots, v_{1,m}^2, \dots, v_{n,1}^2, \dots, v_{n,m}^2, \delta_2)$ be two evolution rules for the transition P system Π represented in the form of vectors of natural numbers. Then the following operations and functions are defined:

- **Rules addition: +**

$$\begin{aligned} r_1 + r_2 &= (u_1^1 + u_1^2, \dots, u_m^1 + u_m^2, & (9) \\ &v_{0,1}^1 + v_{0,1}^2, \dots, v_{0,m}^1 + v_{0,m}^2, \\ &v_{1,1}^1 + v_{1,1}^2, \dots, v_{1,m}^1 + v_{1,m}^2, \\ &\dots, \\ &v_{n,1}^1 + v_{n,1}^2, \dots, v_{n,m}^1 + v_{n,m}^2, \delta_1 + \delta_2) \\ &= (u_1 + u_2, v_1 + v_2, \delta_1 + \delta_2), \end{aligned}$$

where:

- operation (+) for δ 's is defined by the following table which implements the boolean **or** operator:

+	0	1
0	0	1
1	1	1

- and operator (+) for rule antecedent and consequent is the addition for vectors of natural numbers.

- **Product by a natural number: \times**

$$n \times r_2 = (n \times u_1^1, \dots, n \times u_m^1,$$

$$\begin{aligned}
& n \times v_{1,1}^1, \dots, n \times v_{1,m}^1, \\
& \quad \dots, \\
& n \times v_{n,1}^1, \dots, n \times v_{n,m}^1, \delta_1)
\end{aligned}$$

- **Rules Inclusion:** \subseteq_U

$$r_1 \subseteq_U r_2 \Leftrightarrow u_1 \subset u_2 \Leftrightarrow \forall i \in \{1, \dots, m\}, u_i^1 \leq u_i^2 \quad (10)$$

- **Rule Input: Input**

This function is defined as the rule antecedent multiset, therefore:

$$Input\ r_1 = u_1 = (u_1^1, \dots, u_m^1) \in N^m \quad (11)$$

- **Rule Dissolving Capability: Dissolve**

This function tells us whether or not the rule dissolves the membrane. When a rule that dissolves the membrane is executed in a membrane processor, the processor changes from the active to the passive mode. This change implies other changes in the connectivity arrays of its adjacent membrane processors.

$$Dissolve\ r_1 = \delta_1 \in \{0, 1\} \quad (12)$$

- **Set of Membranes the Rule Sends Objects: Ins**

This function cannot be directly implemented with the vector representation. However, this information is very useful for implementing local processing in membrane processors. The considered functionality will be included in the machine instruction format.

5 Machine Instruction Format for Membrane Processors

Evolution rules are here considered as the machine instructions for membrane processors. Hence, they are not considered as processes that are concurrently executed and the computation of a complete multiset of evolution rules is performed in a sequential manner instead of a parallel way.

The membrane processor machine instructions are constituted by four main fields which are described below:

- **Field of Rule Active:**

1 bit for determining if the rule is or not active. This field expresses if the rule can be chosen in a determined processing step involved in the evolution process of the membrane processor.

- **Field of Rule Applicable:**

1 bit for determining if the rule is or not applicable. This field expresses if the rule can be chosen in a determined processing step involved in the evolution process of the membrane processor.

Active	Applicable	Label	Bit vector	Vector of natural numbers
b	b	l	(b_0, b_1, \dots, b_n)	$((a_1, \dots, a_m), (c_{01}, \dots, c_{0m}), \dots, (c_{n1}, \dots, c_{nm}), \delta)$

Figure 1: Machine Instruction structure for Evolution Rules

- **Rule label:**
It contains the number associated to the rule in the rule set.
- **Labels:**
 n bits determining the processors to which the rule sends objects. It has the same configuration as the connectivity array. It is used to determine whether or not the rule is applicable during the calculation of the complete multiset of evolution rules for the processor.
- **The rule as natural number vector:**
This field contains the expression of the rule as belonging to $N^{(n+1) \times m+1}$. It can be divided into the following sub-fields:
 - **Antecedent:** $u \in N^m$
 - **Consequent:** Divided into n objects multisets, one per membrane processor.
 - * $v_1 \in N^m$
 - * \dots
 - * $v_m \in N^m$
 - **Dissolving Capability of the rule:** δ

The structure of an evolution rule as a machine instruction for a membrane processor is depicted in Fig.(1).

6 Processing Algorithm

This section is devoted to describe the operations performed in active membrane processors when computing one evolution step inside the processor.

When a membrane processor is in the active operating mode, the main function to be performed by the processor is the computation of one complete multiset of evolution rules. After that, the processor has to check whether or not it changes its operating mode.

Let us consider, for the rest of this discussion, the transition P system Π of degree n with labels in $L = \{1, \dots, n\}$ and objects in $U = \{u_1, \dots, u_n\}$ where the membrane structure μ of Π has been labelled with the imposed restriction for defining membrane processors connectivity array.

With the classical representation of P systems, we have

$$\Pi = (U, \mu, \omega_1, \dots, \omega_n, (R_1, \rho_1), \dots, (R_n, \rho_n), n) \quad (13)$$

Every membrane processor i in the active operating mode keeps the following information:

1. $Id_Proc(i)$
2. $\psi(i)(0)$
3. $\psi(i)(t)$
4. ω_i
5. R_i
6. ρ_i
7. $\forall j \in \{p \in L \mid p > i \wedge \Psi(i)(p)(0) = 1\} \sigma_{ij}(t)$ ¹

Where:

- ω_i is the multiset of objects for the membrane processor i ,
- $R_i = \{r_j \in N^{(n+2) \times m+1} \mid j \in L_{R_i} = \{1, \dots, p_i\}\}$ the set of evolution rules for the membrane processor i ,
- $\rho_i = \{(i_k, i_l) \in L \times L \mid i_k \neq i_l\}$ the priority relation among rules in R_i with the following meaning: if $(i_k, i_l) \in \rho_i \Leftrightarrow r_{i_k} < r_{i_l}$ (e.g., the rule r_{i_k} has a lower priority than the rule r_{i_l}).

First of all, it is needed to determine the active rules in the membrane processors, and it is done in two steps:

1. To determine applicable rules following the equations:

$$\begin{aligned}
(Useful\ r_j)\ \omega_i &\Leftrightarrow u_j \subseteq \omega_i & (14) \\
\forall j \in L_{R_i} &= \{1, \dots, p_i\} \\
Applicable\ r_j &= Useful\ r_j \wedge (Labels\ r_j \ \& \ \psi(i)(t) = Labels\ r_j).
\end{aligned}$$

2. To determine among applicable rules the active rules. The priority relation ρ_i defined in R_i is used to do it.

$$\begin{aligned}
\forall j \in L_{R_i} &= \{1, \dots, p_i\} & (15) \\
Active\ r_j &= Applicable\ r_j \wedge \\
(\forall (i_k, i_l) \in \rho_i, & i_k \neq j \vee \neg Applicable\ r_{i_l}).
\end{aligned}$$

Once active rules have been determined, the needed multiset of objects have to be computed in the following manner:

At the beginning of the evolution step, it is considered the multiset of objects in the region ω_i . The membrane processor keeps in memory $(n+1)$ multisets of objects (all of them initially are the empty multiset), and one connectivity array used to accumulate the set of membrane label to which the rules used in the evolution step send objects.

¹See [8] for a detailed description of ψ and σ .

$$\begin{aligned}
\omega_i &= \emptyset \\
\Psi_{i,temp} &= (0, \dots, 0) \\
temp_j &= \emptyset, \forall j \in \{0, 1, \dots, n\}
\end{aligned}$$

The performed process is the following:

1. Take one active rule, $r \in R_i$, in a non deterministic manner.
2. The application of the rule produces the following changes:

$$\omega_i = \omega_i - Input\ r \quad (16)$$

$$\Psi_{i,temp} = \Psi_{i,temp} \vee Labels\ r \quad (17)$$

$$temp_j = temp_j + v_j, \forall j \in \{0, 1, \dots, n\} \quad (18)$$

where *Input r* is the rule antecedent, *Labels r* represents the connectivity array of the rule and v_j is the multiset of objects that the rule sends to the membrane processor j .

3. Re-compute active rules in the membrane processor. Some of them cannot be anymore active due to the modification of the multiset ω_i .
 $\forall r \in R_i$, *Activa r* = *Activa r* & (*Useful r*) ω_i . If there is any active rule in the processor, go to step 1.

Once this internal process is finished, the synchronization and communication phases have to be performed and the transition P system evolution is accomplished.

7 Conclusions

This paper presents data structures for representing multisets of objects and evolution rules in digital devices able to perform the computation of complete multisets of evolution rules. These representations permit to implement, in digital devices, the processes performed inside regions of a transition P system. Moreover, the format of machine instructions for membrane processors and a sketch of the processing algorithm for computing complete multiset of evolution rules are also presented.

As a conclusion, it can be said that the basis for the design of a general purpose and modular architecture for implementing transition P systems in hardware devices have been settled now, so it is needed to go ahead in the development of such devices.

References

- [1] Păun G.: Computing with Membranes. *Journal of Computer and Systems Sciences*, 61, 1 (2000) 108–143
- [2] Gh. Păun, G. Rozenberg, A Guide to Membrane Computing. *Theoretical Computer Science*, 287, 1 (2002), 73–100.

- [3] A.V. Baranda, J. Castellanos, R. Gonzalo, F. Arroyo, L.F. Mingo, Data Structures for Implementing Transition P Systems in Silico. *Romanian J. of Information Science and Technology*, 4, 1-2 (2001), 21–32.
- [4] A. V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo, Towards an Electronic Implementation of Membrane Computing: A Formal Description of Nondeterministic Evolution in Transition P Systems. *Proc. 7th Intern. Meeting on DNA Based Computers*, Tampa, Florida, USA (N. Jonoska, N.C. Seeman, eds.) *Lecture Notes in Computer Science 2340*, Springer Verlag, 2002, 350–359.
- [5] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, A Recursive Algorithm for Describing Evolution in Transition P Systems. *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 19–30.
- [6] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo, Structures and Bio-Language to Simulate Transition P Systems on Digital Computers. *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.) *Lecture Notes in Computer Science 2235*, Springer-Verlag, 2001, 1–16.
- [7] F. Arroyo, C. Luengo, A.V. Baranda, L.F. de Mingo, A software simulation of transition P systems in Haskell. *Pre-Proceedings of Second Workshop on Membrane Computing*, Curtea de Arges, Romania, August 2002 and *Proc. of WMC02*, Curtea de Arges, Romania, (Gh. Păun, G. Rozenberg, A. Saloma, C. Zandron, eds.) *Lecture Notes in Computer Science 2597*, Springer-Verlag, 2003, 19–32.
- [8] F. Arroyo, C. Luengo, J. Castellanos, L.F. de Mingo, A Binary Data Structure for Membrane Processors: Connectivity Arrays. (A. Alhazov, C. Martin-Vide, Gh. Păun, eds.) *Preproceedings of the Workshop on Membrane Computing*, Tarragona, July 17-22 2003, 41–52, and in (C. Marin-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Saloma, eds.), *Lecture Notes in Computer Science, 2933*, Springer Verlag, 2004, 19–30.
- [9] <http://psystems.disco.unimib.it/>