

P systems with active objects: Universality and Efficiency

Madhu Mutyam and Kamala Krithivasan

Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai-36, Tamil Nadu, India
madhu@meena.iitm.ernet.in
kamala@iitm.ernet.in

Abstract. P systems, introduced by Gh. Păun form a new class of distributed computing model. Several variants of P systems were already shown to be computationally universal. In this paper, we propose a new variant of P systems, *P systems with active objects*, in which some objects are active and create membranes. This new variant of P systems is capable of solving Hamiltonian Path Problem in linear time. We show that P systems with active objects with membrane dissolving and without priorities is computationally complete.

Keywords: P systems, membrane computing, active P systems, passive P systems, hamiltonian path problem, computational universality, matrix grammar, strong binary normal form, recursively enumerable languages.

P systems with active objects: Universality and Efficiency

1 Introduction

P systems are a class of distributed parallel computing devices of a biochemical type, introduced in [15], which can be seen as a general computing architecture where various types of objects can be processed by various operations. In the basic model one considers a membrane structure with a main membrane, called *skin membrane*, consisting of several cell-like membranes which delimit regions, where we place *objects*. If a membrane does not contain other membrane, it is called an *elementary membrane*. We formalize a membrane structure by means of well-formed parenthesized expressions, strings of correctly matching parentheses, placed in a unique pair of matching parentheses. Each pair of matching parentheses corresponds to a membrane. Graphically a membrane structure is represented by a Venn diagram without intersection and with a unique superset. Each membrane identifies a region, delimited by it and the membrane inside it (if any). If in the regions delimited by the membranes we place multisets of objects (strings). A *P system* Π [15] is provided with evolution rules for its objects (strings). Here the evolution rules are based either on transition of objects of atomic type (*Transition P System*), rewriting of string of objects (*Rewriting P System*) or splicing of string of objects (*Splicing P System*). Here we are considering *Transition P systems* only.

In this paper, we propose a new variant of P systems, *P systems with active objects*, in which some objects are active and create membranes. This new variant of P systems is capable of solving Hamiltonian Path Problem in linear time. We also show that P systems with active objects with membrane dissolving and without priorities is computationally complete.

2 Passive and Active P systems

P systems defined in [15] has a property that number of membranes can not be increased during the processing. Such systems are called as *passive P systems*. And the degree of passive P systems are the number of initial membranes present in the system.

Some times it may happen that number of membranes may be increased in a membrane system, such as P systems with active membranes defined in [14], [12]. Such systems are called as *active P systems*. Since the number of membranes may increase during the processing, we can not take initial membranes as the degree of membrane system. So the degree of active P systems is defined as the ordered

pair consisting of initial membranes as first component and maximum number of membranes used at any time of processing as the second component. Here second component is always greater than or equal to the first component. Since our variant *P systems with active objects* belongs to active P systems category, we are following later notation for describing the degree of the membrane system.

3 Language Prerequisites

A matrix grammar with appearance checking is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N, M$ is a finite set of sequences of the form $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n), n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices.)

For $w, z \in (N \cup T)^*$ we write $w \Rightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n + 1$, such that $w = w_1, z = z_{n+1}$, and, for all $1 < i < n$, either (1) $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}, A_i$ does not appear in w_i , and some rule $A_i \rightarrow x_i$ appears in F . (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied, so we say that these rules are applied in the *appearance checking* mode.)

The language generated by G is defined by $L(G) = \{w \in T^* | S \Rightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac} . When $F = \phi$ (hence we do not use the appearance checking feature), the generated family is denoted by MAT .

It is known that $CF \subset MAT \subset MAT_{ac} = RE$, the inclusion being proper. All one-letter languages in the family MAT are regular [8].

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the binary normal form if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \rightarrow XA)$, with $X \in N_1, A \in N_2$;
2. $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$;
3. $(X \rightarrow Y, A \rightarrow \dagger)$, with $X, Y \in N_1, A \in N_2$;
4. $(X \rightarrow \lambda, A \rightarrow x)$, with $X \in N_1, A \in N_2, x \in T^*$;

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \rightarrow \dagger$ appearing in matrices of type 3; \dagger is called a trap symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

According to [6], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

For an arbitrary matrix grammar $G = (N, T, S, M, F)$, let us denote by $ac(G)$, the cardinality of the set $\{A \in N | A \rightarrow \alpha \in F\}$. From the construction in the proof of Lemma 1.3.7 in [6] one can see that if we start from a matrix grammar G and we get the grammar G' in the binary normal form, then $ac(G') = ac(G)$.

Improving the result from [13] (six nonterminals, all of them used in the appearance checking mode, suffice in order to characterize RE with matrix grammars), in [7] it was proved that four nonterminals are sufficient in order to characterize RE with matrix grammars and out of them only three are used in appearance checking rules. Of interest here is another result from [7]; if the total number of nonterminals is not restricted, then each recursively enumerable language can be generated by a matrix grammar G such that $ac(G) \leq 2$.

Consequently, to the properties of a grammar G in the binary normal form we can add the fact that $ac(G) \leq 2$. we will say that this is the *strong binary normal form* for matrix grammars.

4 P systems with active objects

One of the primary goals of all living creatures is to survive. To do so, cells must be able to reproduce. Cells [1] can reproduce in two ways, *mitosis* and *meiosis*. Meiosis is a form of sexual reproduction and only occurs in gametes (reproductive cells). Mitosis[2] is the process in which a Eukaryotic cell divides into two equal, genetically identical parts. There are 5 stages in mitosis, those are: 1. Interphase, 2. Prophase, 3. Metaphase, 4. Anaphase, 5. Telophase. In Prophase, the nuclear membrane will dissolve and a new nuclear membrane will reform in Telophase. So in nature, it happens that membranes emerge in certain circumstances just by the organization of certain chemical compounds [16]. Similarly, each membrane (in biological sense) is having its own properties and membranes have recognition proteins[3] with which it can recognize other membranes. With the help of *indexing* the membranes and using the target in_i we are modeling the behavior of membrane. We try to model behavior of *membrane creation* with our new variant, i.e., *P-systems with active objects*. Each membrane in the membrane system is having both active and inactive objects. An active object is an object, which can create new membrane and transforms into other object. An inactive object only transforms into another object without creating new membrane.

Formally a *P systems with active objects* of degree (m, n) , $n \geq m \geq 1$, is a construct

$$\Pi = (V, T, C, \mu, w_0, w_1, \dots, w_{(m-1)}, (R_0, \phi), (R_1, \phi), \dots, (R_{(k-1)}, \phi)),$$

where:

1. V is an alphabet; it consists of both active and inactive objects;
2. $T \subseteq V$, is the output alphabet;
3. $C \cap V \neq \phi$, is the set of catalysts;
4. μ is a membrane structure consisting of m membranes, with the membranes and the regions labeled in a one-to-one manner with elements in a given set; here we always use the labels 0 (for skin membrane), $1, \dots, (m - 1)$;
5. w_i , $0 \leq i \leq (m - 1)$, are multisets of objects over V associated with the regions $0, 1, \dots, (m - 1)$ of μ ;

6. R_i , $0 \leq i \leq (k - 1)$, $m \leq k \leq n$, are finite set of evolution rules over V . An evolution rule is of two types:
- (a) If u is a single inactive object, then the evolution rule is in the form $u \rightarrow v$, where $u \in V$, $v = v'$ or $v = v'\delta$, where v' is a multiset of objects over $(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq (k - 1)\})$, and δ is a special symbol not in V .
 - (b) If u is a single active object, then the evolution rule is in the form $u \rightarrow [{}_i v]_i$, where $u \in V$, v is a multiset of objects over V . The rule of the form $u \rightarrow [{}_i v]_i$ means that the object identified by u is transformed into the objects identified by v , surrounded by a new membrane having the label i . No rule of the form $u \rightarrow [{}_0 v]_0$ can appear in any set R_i . During a computation the number of membranes (hence the degree of the system) can increase or decrease.

The membrane structure and the multisets in Π constitute the *initial configuration* of the system. We can pass from a configuration to another one by using the evolution rules. This is done in parallel: all objects, from all membranes, which can be the subject of local evolution rules, as prescribed by the priority relation, should evolve simultaneously. A rule can be used only if there are objects which are *free* at the moment when we check its applicability, and no rule with a higher priority is applicable, irrespective of the objects to which it is applied.

The application of a rule $u \rightarrow v$ in a region containing a multiset w means to remove a copy of the object u from w and to add the objects specified by v , following the prescriptions given by v : If an object appears in v in the form $(a, here)$, then it remains in the same region; if it appears in the form (a, out) , then a copy of the object a will be introduced in the region of the membrane placed outside the region of the rule $u \rightarrow v$; if it appears in the form (a, in_i) , then a copy of a is introduced in the membrane with index i , if such a membrane exists, otherwise the rule cannot be applied. If the special symbol δ appears, then the membrane which delimits the region where we work is dissolved, and all the objects in this region become elements of the region placed immediately outside, while the rules of the dissolved membrane are removed. When applying a rule $u \rightarrow [{}_i v]_i$ in a region j , a copy of a is removed and a membrane with the label i is created, containing the multiset v , inside the region of membrane j . The skin membrane is never dissolved and we can never create a new membrane with label 0.

A sequence of transitions between configurations of a given P system Π is called a *computation* with respect to Π . A computation is *successful* if and only if it halts, that is, there is no rule applicable to the objects present in the last configuration. The result of a successful computation is $\Psi_T(w)$, where w describes the multiset of objects from T which have left the skin membrane during the computation; we say that this vector is *generated* by Π . (Note that we take into account only the objects from T .) We denote by $N(\Pi)$ the set of all vectors generated by Π and by $N_m(\Pi)$ the set of vectors generated by computations whose configurations contain at most m membranes, for a given $m \geq 1$. For each $(n \geq m \geq 1)$, the family of all sets $N_{(m,n)}(\Pi)$ is denoted by $PA_{(m,n)}(\alpha, \beta)$

and their union over (m, n) is denoted by $PA_{(*,*)}(\alpha, \beta)$, where $\alpha \in \{Cat, nCat\}$ and $\beta \in \{\delta, n\delta\}$. Here Cat means *catalyst* in the rules is allowed and δ means *dissolving* of membranes is allowed.

5 Solving Hamiltonian Path Problem for Undirected Graphs

Given an undirected graph $G = (U, E)$ with $n(\geq 2)$ nodes, where U is the set of nodes and E , the set of edges. The Hamiltonian path problem is to determine whether or not there exists a Hamiltonian path in G , that is, to determine whether or not there exists a path that passes through all the nodes in U exactly once. Hamiltonian Path Problem for undirected graphs is known to be NP-Complete. It is possible to solve this problem, in a time which is linear in the number of nodes of a graph, by using P systems with active objects.

Theorem 1 *Hamiltonian Path Problem for undirected graphs can be solved, in a time which is linear in the number of nodes of a graph, by using P systems with active objects.*

Proof : Let $G = (U, E)$ be an undirected graph with $n(\geq 2)$ nodes. Let $U = \{a_1, a_2, \dots, a_n\}$. We now construct P systems with active objects of degree $(1, m)$, $PA_{(1,m)}(nCat, \delta)$ as

$$\Pi = (V, T, C, \mu, w_0, (R_0, \phi), (R_1, \phi), \dots, (R_n, \phi))$$

where $V = \{a_i, c_i^j \mid 1 \leq i \leq n, 0 \leq j \leq n\} \cup \{Y, f, t_i \mid 0 \leq i \leq 2n\}$;
(Here a_i are active objects and Y, f , and t_i are inactive objects)

$T = \{Y\}$;

$C = \phi$;

$\mu = [0]_0$;

$w_0 = \{t_0, a_1, a_2, \dots, a_n\}$;

The set R_0 contains the following rules:

1. $t_i \rightarrow t_{i+1}, 0 \leq i \leq 2n - 1$;
2. $t_{2n} \rightarrow \delta$;
3. $c_i^j \rightarrow \lambda, 1 \leq j \leq n - 1, \forall i$;
4. $c_i^n \rightarrow (Y, out), \forall i$;
5. $a_i \rightarrow [{}_i c_i^0, a_{j_1}, a_{j_2}, \dots, a_{j_k}]_i, 1 \leq i \leq n$;
(a_i will create a new membrane with label i and transform into a multiset of objects $c_i^0 a_{j_1} a_{j_2} \dots a_{j_k}$, where j_1, \dots, j_k are vertices adjacent to vertex i)

The set R_i contains the following rules:

1. $c_j^k \rightarrow (c_j^{k+1}, out), \forall j \neq i, k \geq 0$;
2. $c_i^k \rightarrow \lambda, k \geq 1$;
3. $a_j \rightarrow [{}_j c_j^0, a_{j_1}, a_{j_2}, \dots, a_{j_k}]_j, 1 \leq j \leq n$;

$$N_{(1,k)}(\Pi) = \begin{cases} Y^m, & \text{if the given graph has 'm' hamiltonian paths.} \\ \phi, & \text{otherwise.} \end{cases}$$

The system works as follows: Initially the skin membrane is having the active objects a_1, \dots, a_n . Each active object a_i , by using the rule $a_i \rightarrow [{}_i c_i^0, a_{j_1}, \dots, a_{j_k}]_i$, will create a membrane with index i and transform into a multiset of objects $c_i^0, a_{j_1}, \dots, a_{j_k}$. For every rewriting step, c_j^k will be rewritten as c_j^{k+1} and sent out of the membrane i , where $i \neq j$. If $i = j$, then c_j^k will be erased. In skin membrane we use a counter t_i so that for every rewriting step it will be incremented and when ever the counter reaches t_{2n} , indicates the end of the computation, it will dissolve the whole membrane system by dissolving the skin membrane. So, if there exists a tour of length n , by traversing all nodes in the graph, then we can get Y from the system.

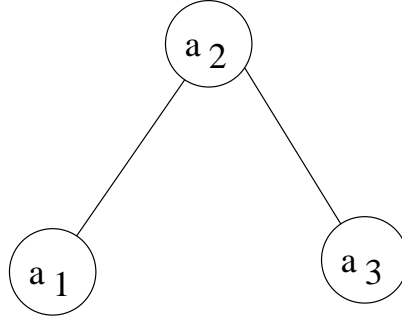
Time Complexity:

- This algorithm takes $2n$ steps for generating the output.

Note:- HPP problem can also be solved using *active membranes* [12] in $3n + 1$ steps but it requires the rules for division of membranes. \square

Example 1 *Checking whether or not there exists Hamiltonian Path in a given undirected graph.*

Sol: Let us consider the graph $G = (U, E)$, with $U = a_1, a_2, a_3$ as shown below. We now construct P systems with active objects of degree $(1, 50)$ as



$$\Pi = (V, T, C, \mu, w_0, (R_0, \phi), \dots, (R_3, \phi)),$$

where

$$V = \{a_i, c_i^j | 1 \leq i \leq 3, 0 \leq j \leq 3\} \cup \{t_i | 0 \leq i \leq 6\} \cup \{Y, f\};$$

$$T = \{Y\};$$

$$C = \phi;$$

form, that is with $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with rules of the four forms mentioned in Section 3, and with $ac(G) \leq 2$. Assume that we are in the worst case, with $ac(G) = 2$, and let $B^{(1)}, B^{(2)}$ be the two symbols in N_2 for which we have rules $B^{(j)} \rightarrow \dagger$ in matrices of M . Let us assume that we have h matrices of the form $m'_i : (X \rightarrow Y, B^{(j)} \rightarrow \dagger), X, Y \in N_1, j = 1, 2, 1 \leq i \leq h$, and k matrices of the form $m_i : (X \rightarrow \alpha, A \rightarrow x), X \in N_1, A \in N_2, \alpha \in N_1 \cup \{\lambda\}$, and $x \in (N_2 \cup T)^*, 1 \leq i \leq k$. Each matrix of the form $(X \rightarrow \lambda, A \rightarrow x), X \in N_1, A \in N_2, x \in T^*$, is replaced by $(X \rightarrow f, A \rightarrow x)$, where f is a new symbol. We continue to label the obtained matrix in the same way as the original one. The matrices of the form $(X \rightarrow Y, B^{(j)} \rightarrow \dagger), X, Y \in N_1$ are labeled by m'_i , with $i \in lab_j$, for $j = 1, 2$, such that lab_1, lab_2 and $lab_0 = \{1, 2, \dots, k\}$ are mutually disjoint sets.

We construct the P systems with active objects as

$$\Pi = (V, T, C, \mu, w_s, w_0, w_1, w_2, R_s, R_0, R_1, R_2),$$

with the following components :

$$\begin{aligned} V = & N_1 \cup N_2 \cup \{X', X'', X_i \mid X \in N_1, 1 \leq i \leq k\} \\ & \cup \{X_i \mid X \in N_1, 1 \leq i \leq h\} \\ & \cup \{A_i, A'_i \mid A \in N_2, 1 \leq i \leq k\} \\ & \cup \{E, E', \dagger\} \end{aligned}$$

$$C = \{c\};$$

$$\mu = [s]_s;$$

$$w_s = \{X, A\}, \text{ for } (S \rightarrow XA) \text{ being the initial matrix of } G;$$

$$w_0 = w_1 = w_2 = \phi;$$

and the sets R_s, R_0, R_1 and R_2 contains the following rules:

- R_s :
 1. $X \rightarrow [{}_0X']_0$;
 2. $X \rightarrow [{}_jX_i]_j$, for $m'_i : (X \rightarrow Y, B^{(j)} \rightarrow \dagger)$;
 3. $cA[{}_0]_0 \rightarrow [{}_0cA']_0$;
 4. $A[{}_j]_j \rightarrow [{}_jA]_j, \forall A \in N_2, j = 1, 2$;
 5. $a \rightarrow (a, out)$;
 6. $X_i \rightarrow \dagger$;
 7. $cA \rightarrow c\dagger, A \in N_2$;
 8. $\dagger \rightarrow \dagger$;
 9. $E' \rightarrow (E, in)$;
 10. $Y'' \rightarrow Y'$;
 11. $Y' \rightarrow Y$;
- R_0 :
 1. $X' \rightarrow X_1$;
 2. $A' \rightarrow A_1$;
 3. $X_i \rightarrow X_{i+1}, 1 \leq i \leq (k-1)$;
 4. $A_i \rightarrow A_{i+1}, 1 \leq i \leq (k-1)$;
 5. $X_i \rightarrow Y$, for $m_i : (X \rightarrow Y, A \rightarrow x)$;
 6. $A_i \rightarrow x\delta$, for $m_i : (X \rightarrow Y, A \rightarrow x)$;
 7. $Y \rightarrow \dagger$;

8. $X_k \rightarrow \dagger$;
 9. $A_k \rightarrow \dagger$;
 10. $\dagger \rightarrow \dagger$;
- R_j , where $j = 1, 2$, such that $m'_i : (X \rightarrow Y, B^{(j)} \rightarrow \dagger)$:
1. $X_i \rightarrow (Y''E', out)$;
 2. $B^{(j)} \rightarrow \dagger$;
 3. $\dagger \rightarrow \dagger$;
 4. $E \rightarrow \delta$;

The system works as follows:

Initially the skin membrane contains the objects X, A . Here X is the active object and creates a membrane. In order to simulate a matrix of type 2 or 4, the object X will create a membrane with index 0. Whenever a membrane with index 0 appears, with the help of catalyst, an inactive object $A \in N_2$ will move into the membrane 0. In membrane 0, X' will introduce X_1 and A' will introduce A_1 and the subscripts increased step by step. At any time we can apply the rules $X_i \rightarrow Y$ and $A_i \rightarrow x\delta$. But if we apply the rule $X_i \rightarrow Y$ first, then a trap symbol will be introduced by the rule $Y \rightarrow \dagger$. Similarly, if we apply the rule $A_i \rightarrow x\delta$ first, then a trap symbol will be introduced in the skin membrane by the rule $X_i \rightarrow \dagger$. So, in order to get the correct simulation, we have to apply these two rules at the same step. In this way we can complete the simulation of a matrix m_i of type 2 or 4. If the symbol A in membrane 0 is not the one in the matrix $m_i : (X \rightarrow Y, A \rightarrow x)$, then we cannot apply the rule $A_i \rightarrow x\delta$ and a trap symbol will be introduced by the rule $Y \rightarrow \dagger$. After simulating a matrix of type 4, if there is any nonterminal $A \in N_2$ present in the skin membrane, then a trap symbol will be introduced by the rule $cA \rightarrow \dagger$.

For simulating a matrix of type 3, X will create a membrane with index $j, j = 1, 2$. Whenever a membrane with index j is present, all the nonterminals $A \in N_2$ should move into it. Otherwise a trap symbol will be introduced by the rule $cA \rightarrow \dagger, A \in N_2$. In membrane j , X_i will be replaced with $Y''E'$ and sent out. If there is any symbol $B^{(j)}$ present in membrane j , then a trap symbol will be introduced. In the next step, Y'' will be replaced with Y' and E' with (E, in) . Finally with the help of $E \rightarrow \delta$, we can dissolve the membrane j and at the same time Y' will be replaced with Y . In this way we can complete the simulation of matrix m'_i . Thus, the equality $\psi_T(L(G)) = N_{(1,4)}(\Pi)$ follows. \square

7 Conclusion

As a variant of P systems, we have introduced a new variant of P systems, *P systems with active objects*, in which some objects are active and create membranes. This new variant of P systems is capable of solving Hamiltonian Path Problem in linear time. We shown that P systems with active objects with membrane dissolving rules and without priorities is computationally complete.

References

1. <http://library.thinkquest.org/C004535/reproduction.html>
2. <http://fermat.stmarys-ca.edu/~jpolos/science/mitosis.html>
3. <http://scidiv.bcc.ctc.edu/rkr/biology101/lectures/membranes.html>
4. <http://bioinformatics.bio.disco.unimib.it/psystems/>
5. Dassow, J., Păun, Gh.: On the Power of Membrane Computing. *Journal of Universal Computer Science*, Vol. 5 (1999) 33-49.
6. Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
7. Freund, R., Păun, Gh.: On the number of nonterminals in graph-controlled, programmed, and matrix grammars, submitted, 2000.
8. Hauschild, D., Jantzen, M.: Petri nets algorithms in the theory of matrix grammars. *Acta Informatica*, vol. 31 (1994) 719-728.
9. Ito, M., Martin-Vide, C., and Păun, Gh.: A characterization of Parikh sets of ETOL languages in terms of P systems. (2000).
10. Krishna, S.N., Rama, R.: On Simple P Systems with External Output. submitted (2000).
11. Krishna, S.N.: Computing with Simple P systems. *Workshop on Multiset Processing* (2000).
12. Krishna, S.N., Rama, R.: A variant of P systems with Active Membranes: Solving NP-Complete Problems. *Romanian Journal of Information Science and Technology*, vol. 2 (1999) 357-367.
13. Păun, Gh.: Six nonterminals are enough for generating each r.e. language by a matrix grammar, *Intern. J. Computer Math.*, Vol. 15 (1984), 23-37.
14. Păun, Gh.: P systems with Active Membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*.
15. Păun, Gh.: Computing with Membranes. *Journal of Computer and System Sciences*. Vol. 61 (2000) 108-143.
16. Păun, Gh.: *Computing with Membranes (P systems): Twenty six Research topics*. Auckland University (2000).
17. Rozenberg, G., Salomaa, A.: *Handbook of Formal Languages*. Springer-Verlag, Berlin (1997).