Variations on the Theme: P Colonies

Lucie Ciencialová *
ciecilka@gmail.com

Luděk Cienciala*
ludek.cienciala@fpf.slu.cz

Abstract: In this paper we present the results achieved in research of P colonies, a biochemically inspired formal model of a computing device. We show that P colonies with one or two objects inside each agent are computationally complete.

Keywords: P colonies, membrane systems, P systems.

1 Introduction

P colonies were introduced in the paper [3] as a formal model of a computing device inspired by membrane systems and formal grammars called colonies. This model is inspirated by structure and functioning of a community of living organisms in a shared enviroment.

The independent organisms living in a P colony are called agents. Each agent consists of some objects embedded in a membrane. The number of objects inside the agent is the same for each of them. The enviroment contains several copies of the basic enviromental object denoted by e. The number of the copies of e is unlimited.

The set of programs is associated with each agent. The program determines the activity of the agent. Each program consists of the same number of rules as the number of the objects inside the agent. So in every moment all the objects inside of the agent are being evolved or transported.

The rules are as simple as they can be. There are two kinds of rules in programs. The first type called rewriting is in the form $a \rightarrow b$. It means that object a inside of agent is rewritten to object b. The second type of rules can be called communication and they are in the form $c \leftrightarrow d$. When this rule is performed, the object c inside and the object d outside of the agent change their places, so d is now inside and c is outside of the agent.

In the specific model introduced in [3] only two objects are allowed to be inside of each agent. Each program consists of one evolution rule and one communication rule $\langle a \rightarrow b, c \leftrightarrow d \rangle$.

In [3] the ability of agents is extended by checking programs. We give the agents the posibility to opt between two possibilities. They have form $\langle a \rightarrow b, c \leftrightarrow d/c' \leftrightarrow d' \rangle$. If this program is performing, the communication rule $c \leftrightarrow d$ has higher priority to be executed than the rule $c' \leftrightarrow d'$. It means that the agent checks the possibility to use the rule $c \leftrightarrow d$ (it tries to find object c inside of it and the object d in the enviroment). If this rule can be executed the agent must use it. If the first rule cannot be applied, the agent uses the second one $c' \leftrightarrow d'$. In the case of two objects inside of each agent objects c and c' are identical, we can rewrite the checking rule to the form $c \leftrightarrow d/c \leftrightarrow d'$.

* Institute of Computer Science, Silesian University in Opava, Czech Republic

At the beginning of computation the enviroment and all agents contain only copies of object e. By using their programs the agents change their content and by enviroment can affect behavior of other agents. In each step of computation each agent nondeterministically chooses one of its applicable programs and executes it. The computation halts when no agent can apply any of its programs. The result of the computation is the number of some specific objects present at the enviroment at the end of the computation.

In this paper we study the computational power of P colonies. The restricted P colonies with two objects in an agent are studied in works [2, 3, 4, 5]. The posibility of another number of objects inside of agent is is mentioned too. Some results for P colonies with three objects in each agent were presented in [5]. We present that P colonies with one object in a agent are computationaly complete too.

2 Definitions

2.1 P colony

Definition 1. The P colony of the capacity c is a construct

$$\Pi = (A, e, f, B_1, \ldots, B_n), \ where$$

- A is an alphabet of the colony, its elements are called objects,
- $e \in A$ is the basic object of the colony,
- $f \in A$ is the final object of the colony,
- B_i, $1 \leq i \leq n$, are agents, B is construct $B_i = (O_i, P_i)$, where
  - O_i is a multiset of c copies of the basic object e, this multiset determines the initial state of the agent,
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where the programs contain c rules, which are in one of the following forms:
    - $a \rightarrow b$, these rules are called evolution rules,
    - $c \leftrightarrow d$, these rules are called communication rules,
    - $c \leftrightarrow d/c' \leftrightarrow d'$, which are called checking rules.

At the beginning of the computation the enviroment contains arbitrary number of copies of the object e, each agent contains c copies of e. The P colony is in its initial configuration. A configuration is $(n+1)-$tuple of strings of objects. Formally, the configuration of P colony Π is $(w_1, \ldots, w_n, w_E e^\omega)$, where $|w_i| = c$, $1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent and $w_E \in (A - \{e\})^*$ represents all the objects in the enviroment different from e.

The computation can be done in two different ways. At each step of the parallel computation each agent tries to find one program to use. If the number of applicable programs is higher than one, the agent nondeterministically chooses one of them. At one step of computation the maximal number of agents works. At each step of sequential computation only one agent use its program.

The computation ends by halting, it means that no agent can use any of its programs. With a halting computation we can associate a result of the computation. It is the number of copies of the special symbol f present in the enviroment.

Because of nondeterminism in a computation of the P colony, we can obtain more than one halting computation. Hence what we associate with P colony Π is a set of natural numbers denoted by $N(\Pi)$ computed by all possible halting computations of Π.

Given a P colony $\Pi = (A,\ e,\ f,\ B_1,\ \ldots,\ B_n)$ the maximal number of programs associated with the agents in P colony Π is called the height of P colony Π. The degree of P colony Π is the number of agents in P colony Π. The third parameter characterizing a P colony is the capacity of P colony Π describing the number of the objects inside each agent.

The family of all sets of numbers $N(\Pi)$ computed by P colonies of the capacity c, the degree at most n and the height at most h without using checking rules in their programs is denoted by $NPCOL_{par}(c, n, h)$ for P colonies work in parallel way and $NPCOL_{seq}(c, n, h)$ for P colonies work in sequential way. If we allow checking rules, the family of all sets of numbers computed by P colonies is denoted by $NPCOL_mK$. If the P colonies are restricted too, we change notation to $NPCOL_mR$ or $NPCOL_mKR$ for m being seq or par.

2.2 Register machine

In this work we want to characterize the generative power of the families $NPCOL_m(c, n, h)$ comparing them with the family of sets of numbers computed by Turing machines. To achieve this aim we need the notion of a register machine.

Definition 2. [6] A register machine is the construct $M = (m, H, l_0, l_h, P)$ where:

- m is the number of registers,
- H is the set of instruction labels,
- l_0 is the intial/start label,
- l_h is the final label,
- P is a finite set of instructions injectively labelled with the elements from the given set H.

The instruction of the register machine are of the following forms:

$l_1 : (ADD(r), l_2, l_3)$ Add 1 to the contents of the register r and proceed to the instruction (labelled with) l_2 or l_3.

$l_1 : (SUB(r), l_2, l_3)$ If the register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3.

$l_h : HALT$ Stop the machine. The final label l_h is only assigned to this instruction.

Without loss of generality, we can assume that in each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ and in each conditional ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct.

The register machine M computes a set $N(M)$ of numbers in the folloving way: it starts with all registers empty (hence storing the number zero) with the instruction with label l_0 and we proceed to apply the instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number stored at that time in the register 1 is said to be computed by M and hence it is introduced in $N(M)$. (Because of the nondeterminism in choosing the continuation of the computation in the case of ADD-instructions, $N(M)$ can be an infinite set.) It is known (see e.g.[4]) that in this way we can compute all sets of numbers which are Turing computable.

3 P colonies with one object inside the agent

In this part we analyze behaviour of a P colony with only one object inside each agent "living" in this P colony. It means that every program is formed by only one rule. This rule is rewriting, communication or checking. If all the agents have their programs from rewriting rules, the agents "live only for themselves" and do not communicate with the enviroment.

Theorem 3. $NPCOL_{par}(1, *, 7) = NRE.$

Proof. Let us consider a register machine $M = (m, H, l_0, l_h, P)$. All the labels from H will be objects from P colony. The contents of a register i will be represented by the number of copies of a specific object a_i in the enviroment. We will construct a P colony $\Pi = (A, f, e, B_1, \ldots, B_n)$ with:

- the alphabet $A = H \cup \{a_i \mid 1 \leq i \leq m\} \cup \{F_i \mid 1 \leq i \leq |H|\} \cup \{e, d, D\}$
- final object $f = a_1$
- agent $B_i = (e, P_i)$, $1 \leq i \leq |H| + 3 = n$, and its programs are following:

1. We consider the starting agents B_1, B_2 with a set of programs:

   $P_1 = \{ \langle e \rightarrow l_0 \rangle, \langle l_0 \leftrightarrow D / l_0 \leftrightarrow e \rangle \}.$ $\qquad$ $P_2 = \{ \langle e \rightarrow D \rangle, \langle D \leftrightarrow l_0 \rangle \}.$

   The agent B_1 generates and sends copies of the initial label l_0 of the register machine M and stops by consuming one copy of the object D. The second agent B_2 generates one copy of D and waits for the object l_0. After having transported it inside the agent finishes its work. A simulation of computation of M can start.

2. We need one more agent to generate some special object d. In every second step the agent B_3 places one copy of d to the enviroment.

   $P_3 = \{ \langle e \rightarrow d \rangle, \langle d \leftrightarrow H / d \leftrightarrow e \rangle \}.$

3. For each instruction $l_1 : (ADD(r), l_2, l_3)$ there is one agent in P colony Π. This agent has to add one copy of the object a_r and the object l_2 or l_3 to the enviroment.

   $P_{l_1} = \{ \langle e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow a_r \rangle, \langle a_r \leftrightarrow d \rangle, \langle d \rightarrow l_2 \rangle,$
   $\langle d \rightarrow l_3 \rangle, \langle l_2 \leftrightarrow e \rangle, \langle l_3 \leftrightarrow e \rangle \}.$

   If the object l_1 is present in the enviroment, the agent B_{l_1} can start to be active, it can consume the object l_1, generate the object a_r, place it to the enviroment and finally exchange the object l_2 or l_3 by e. At the end of this part of the computation the object with the label of the next instruction of M is placed in the enviroment and another agent can start to work.

4. For each instruction $l_1 : (SUB(r), l_2, l_3)$ from P we consider the agent B_{l_1} with the set of programs:

   $P_{l_1} = \{ \langle e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow F_1 \rangle, \langle F_1 \leftrightarrow a_r \ / \ F_1 \leftrightarrow d \rangle, \langle a_r \rightarrow l_2 \rangle,$
   $\langle d \rightarrow l_3 \rangle, \langle l_2 \leftrightarrow e \rangle, \langle l_3 \leftrightarrow e \rangle \}.$

   The agent again brings inside the object l_1 and changes it to another object F_1. In the next step the agent checks whether at least one copy of a_r is present in the enviroment. In the positive case the agent consumes a_r inside itself and rewrites it to the object l_2.
   In the negative case the agent consumes the object d and rewrite it to the object l_3. In the last step the agent again exchanges the object l_2 or l_3 by e.

5. For the halting instruction labelled l_h we consider the agent B_{l_h} with the following set of programs:

   $P_{l_h} = \{ \langle e \leftrightarrow l_h \rangle , \langle l_h \rightarrow H \rangle , \langle H \leftrightarrow d \rangle \}.$

   The agent consume the object l_h and in the enviroment there is no other object l_m. This agent places one copy of the object H to the enviroment and stops working. The object H is on the next step consumed by the agent B_3. No agent can start its work and computation halts.

From the previous explanations, it is easy to see that P colony Π correctly simulates computation in the register machine M.

The computation of Π starts with no object a_r placed in the enviroment in the same way as the computation in M starts with zeroes in all the registers. The computation of Π stops if the symbol l_h is placed inside the corresponding agent in the same way as M stops by executing the halting instruction labelled l_h.

Consequently, $N(M) = N(\Pi)$, and because each agent contains at most seven programs, the proof is complete. □

Another question is how many agents are necessary to simulate any register machine. To do this we need do add one type of checking rules. At the definitions we said that checking rule has the form $\langle c \leftrightarrow d \ / \ c' \leftrightarrow d' \rangle$. Because each agent contains only one object, we change it to $\langle c \leftrightarrow d \ / \ c' \rightarrow d' \rangle$. It means that the agent tries to exchange the object c for object d. If its demand is not satisfied the agent rewrites the object c' to d'.

Theorem 4. $NPCOL_{par}(1, 5, *) = NRE$

Proof. Let us consider a register machine $M = (m, H, l_0, l_h, P)$ and present the content of the register i by the number of copies of a specific object a_i in the enviroment. We construct a P colony $\Pi = (A, f, e, B_1, \ldots, B_5)$ with:

- alphabet $A = \{l_i, l_i' | l_i \in H\} \cup \{L_i, E_i, E_i', F_i, F_i', F_i'' \mid \textit{for each } l_i \in H\} \cup$
  $\cup \{a_i | 1 \leq i \leq m\} \cup \{e, d, m, D, H\},$
- $f = a_1,$
- $B_i = (e, P_i),\ 1 \leq i \leq 5.$

1. To initialize simulation of computation of M we take two agents $B_1 = (e, P_1)$ and $B_2 = (e, P_2)$ with two sets of programs:

   $P_1 = \{ \langle e \rightarrow l_0 \rangle , \langle l_0 \leftrightarrow D / l_0 \leftrightarrow e \rangle \}.$ $\qquad$ $P_2 = \{ \langle e \rightarrow D \rangle , \langle D \leftrightarrow l_0 \rangle \}.$

2. We need one more agent to generate some special object d. In every two steps the agent B_3 places one copy of d to the enviroment.

   $P_3 = \{ \langle e \rightarrow d \rangle , \langle d \leftrightarrow H / d \leftrightarrow e \rangle \}.$

3. To simulate the ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ there are two agents B_4 and B_5 in P colony Π. These agents help themselves to add one copy of object a_r and object l_2 or l_3 to the enviroment.

Programs in P_4	Programs in P_5	Programs in P_5
$\langle e \leftrightarrow l_1 \rangle$,		$\langle e \leftrightarrow l'_2 \rangle$,
$\langle l_1 \rightarrow E_1 \rangle$,		$\langle e \leftrightarrow l'_3 \rangle$,
$\langle E_1 \leftrightarrow d \rangle$,		$\langle l'_2 \rightarrow l_2 \rangle$,
$\langle d \rightarrow L_1 \rangle$,	$\langle e \leftrightarrow E_1 \rangle$,	$\langle l'_3 \rightarrow l_3 \rangle$,
	$\langle E_1 \rightarrow E'_1 \rangle$,	$\langle l_2 \leftrightarrow e \rangle$,
	$\langle E'_1 \leftrightarrow e \rangle$,	$\langle l_3 \leftrightarrow e \rangle$.
$\langle L_1 \leftrightarrow E'_1 / L_1 \rightarrow m \rangle$,		
$\langle m \rightarrow d \rangle$,		
$\langle E'_1 \rightarrow l'_2 \rangle$,	$\langle e \leftrightarrow L_1 \rangle$,	
$\langle E'_1 \rightarrow l'_3 \rangle$,	$\langle L_1 \rightarrow a_r \rangle$,	
$\langle l'_2 \leftrightarrow e \rangle$,	$\langle a_r \leftrightarrow e \rangle$,	
$\langle l'_3 \leftrightarrow e \rangle$.		

The agent B_4 consumes the object l_1, changes it to E_1 and places it to the enviroment. The agent B_5 lends E_1 from the enviroment and a little altered (to E'_1) gives back. B_4 rewrites the object d to some L_i. If this L_i has the same index as E'_i placed in the enviroment, the computation can go to the next phase. If indexes of L_i and E_i are different the agent B_4 tries to generate another L_i. If the computation gets over this checking step, B_4 generates the helpful object l'_2 or l'_3 and places them to the enviroment. The agent B_5 exchanges it to "valid label" l_2 or l_3.

4. For each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ there are subsets of P_4 and P_5:

Programs in P_4	Programs in P_5	Programs in P_5
$\langle e \leftrightarrow l_1 \rangle$,		$\langle e \leftrightarrow l'_2 \rangle$,
$\langle l_1 \rightarrow F_1 \rangle$,		$\langle e \leftrightarrow l'_3 \rangle$,
$\langle F_1 \leftrightarrow d \rangle$,		$\langle l'_2 \rightarrow l_2 \rangle$,
	$\langle e \leftrightarrow F_1 \rangle$,	$\langle l'_3 \rightarrow l_3 \rangle$,
	$\langle F_1 \rightarrow F'_1 \rangle$,	$\langle l_2 \leftrightarrow e \rangle$,
	$\langle F'_1 \leftrightarrow a_r / F'_1 \rightarrow F''_1 \rangle$,	$\langle l_3 \leftrightarrow e \rangle$
$\langle d \leftrightarrow F'_1 \rangle$,	$\langle a_r \rightarrow e \rangle$,	
$\langle F'_1 \rightarrow l'_2 \rangle$,	$\langle F''_1 \leftrightarrow e \rangle$,	
$\langle d \leftrightarrow F''_1 \rangle$,		
$\langle F''_1 \rightarrow l'_3 \rangle$,		
$\langle l'_2 \leftrightarrow e \rangle$,		
$\langle l'_3 \leftrightarrow e \rangle$		

The agent B_4 starts simulation of executing of SUB-instruction l_1, the agent B_5 checks whether there is a copy of the object a_r in the enviroment or not and gives this information (F'_1 = there is some a_r, F''_1 = there is no object a_r in the enviroment) to agent B_4 by placing the object F'_1 or F''_1 to the enviroment.

5. The halting instruction l_h is simulated by the agent B_4 with subset of programs:

   $\{ \langle e \leftrightarrow l_h \rangle, \langle l_h \rightarrow H \rangle, \langle H \leftrightarrow e \rangle \}$.

   The agent consumes the object l_h and in the enviroment there is no other object l_m. This agent places one copy of the object H to the enviroment and stops working. At the next step the object H is consumed by the agent B_3. No agent can start its work and computation halts.

From the previous explanations, it is easy to see that P colony Π correctly simulates compu-

tation in register machine M. The computation of Π starts with no object a_r placed in the enviroment in the same way as the computation in M starts with zeroes in all the registers. The computation of Π stops if the symbol l_h is placed inside the corresponding agent in the same way as M stops by executing the halting instruction labelled l_h. Consequently, $N(M) = N(\Pi)$ and because the number of agents is five, the proof is complete. □

4 P colonies with two objects inside the agent

In the case of agents with two objects each program must consists of two rules. If the former of these rules is evolving and the latter is communication or checking, we talk about restricted P colonies. If we allow also another combination of these three types of the rules, we obtain non-restricted P colonies. The restricted P colonies with the checking rules are computationally complete. The next results were proved to be true:

- $NPCOL_{par}KR(2,*,5) = NRE$ in [1, 4],
- $NPCOL_{par}R(2,*,5)$ and $NPCOL_{seq/par}KR(2,1,*) = NRE$ in [2] ,
- $NPCOL_{seq}(2,1,*) = NPCOL_{seq}(2,*,*) = NMAT$ in [2], where $NMAT$ is the family of sets of vectors of non-negative integers generated by partially blind register machines.

In the case of two rules in the same form in the program, we can say that the program is rewriting, communication or checking one. The rewriting program can be modified to the form $\langle ab \rightarrow cd \rangle$. In the same way we can modify communication (and checking) programs to the form $\langle ab \leftrightarrow cd \rangle$ (and $\langle ab \leftrightarrow cd / ab' \leftrightarrow cd' \rangle$).

Theorem 5. $NPCOL_{par/seq}K(2,1,*) = NRE$.

Proof. Let us consider a register machine M with m registers. We construct a P colony $\Pi = (A, f, e, B)$ simulating a computation of register machine M with:

- $A = \{d, a, s, f, h, v\} \cup \{l, l' \mid l \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$,
- $f = a_1$,
- $B = (O, P),\ O = \{d, d\}$

At the beginning of computation the agent generates the object l_0 (the label of starting instruction of M) and two copies of the object a. Then agent starts to simulate instruction labelled l_0 and generates the label of the next instruction. The set of programs is as follows:

1. For initializing of the simulation there are programs in P:

   $\langle dd \rightarrow sl_0 \rangle , \langle sl_0 \leftrightarrow ee \rangle , \langle ee \rightarrow ah \rangle , \langle ah \leftrightarrow se \rangle , \langle se \rightarrow af \rangle , \langle af \leftrightarrow l_0 h \rangle , \langle l_0 h \leftrightarrow aa \rangle ,$

2. For every ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ we add programs to P:

   $\langle aa \leftrightarrow l_1 e \rangle , \langle el_1 \rightarrow l_2' a_r \rangle , \langle el_1 \rightarrow l_3' a_r \rangle , \langle l_2' a_r \leftrightarrow ef \rangle , \langle l_3' a_r \leftrightarrow ef \rangle ,$
   $\langle ef \leftrightarrow el_2' \rangle , \langle ef \leftrightarrow el_3' \rangle , \ \langle el_2' \rightarrow l_2 v \rangle , \ \langle el_3' \rightarrow l_3 v \rangle , \ \langle l_2 v \leftrightarrow aa \rangle , \ \langle l_3 v \leftrightarrow aa \rangle ,$

   When the agent takes objects l_1 and e inside, it rewrites them to one copy of a_r and the object l_2' or l_3'. The next sequence of steps finishes by generating of l_2 or l_3. This object must be sent out to the enviroment with object v.
3. For every SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ there is a subset of programs in P:

   $\langle aa \leftrightarrow l_1 a_r \ / \ aa \leftrightarrow l_1 e \rangle , \langle l_1 a_r \rightarrow l_2 v \rangle , \langle l_1 e \rightarrow l_3 v \rangle , \langle l_2 v \leftrightarrow aa \rangle , \langle l_3 v \leftrightarrow aa \rangle ,$

At the first step the agent checks if there is any copy of a_r on the enviroment (if register r is noempty). In the positive case it brings l_1 with a_r inside, in the negative case l_1 enters the agent with symbol e. In dependence on the content of the agent, it generates the object l_2 or l_3.

4. For the halting instruction l_h there is a program in the set P:

   $\langle aa \leftrightarrow hl_h \rangle$

   By using this program, the P colony finishes computation as well as the register machine halts its computation.

It is easy to see that the P colony Π correctly simulates any computation of the register machine M and the number contained on the first register of M corresponds to the number of copies of the object a_1 presented in the enviroment of Π. □

5 Conclusions

We have shown that the P colonies with capacity $c = 1$ and with checking programs with height at most seven are computationally complete and with checking/rewriting programs with degree 5 they have the same computational power. In the last part of this study we have shown that P colonies with two objects inside the agent and with homogeneous programs can have only one agent (degree $d = 1$) and we obtain computational completeness too.

Activities carried out in the field of membrane computing are currently numerous and available at [9].

This work has been supported by the Grant Agency of Czech Republic grants No. 201/04/0528 "Výpočetní aspekty emergence - teorie a experimenty" and Grant Agency of Czech Republic grants No. 201/06/0567 "Bioinformatika a biovýpočty: souvislosti, modely a aplikace".

Bibliography

1. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P Colonies.* Journal of Multi-Valued Logic, 2005(accepted) 13 pp.
2. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode.* Pre/Proceedings of the 1st International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds), Timisoara, Romania,2005, pp. 49-56.
3. Kelemen, J., Kelemenová, A., Păun, Gh.: *On the power of a biochemically inspired simple computing model: P colonies.* Downloadable version at [9].
4. Kelemen, J., Kelemenová, A., Păun, Gh.: *The power of cooperation in a biochemically inspired simple computing model: P colonies.* Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX (M. Bedau at al., eds) Boston, Masss, 2004, pp. 82–86.
5. Kelemen, J., Kelemenová, A.: *On P colonies, a Simple Bio-Chemically Inspired Model of Computation.* Manuscript.
6. M. L. Minsky: *Computation Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967
7. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108-143 and TUCS Research Report No 208, Turku, 1998.
8. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
9. P systems web page: http://psystems.disco.unimib.it