# Eco-P Colonies*

Luděk Cienciala, Lucie Ciencialová,

Institute of Computer Science, Silesian University in Opava, Czech Republic
{ludek.cienciala, lucie.ciencialova}@fpf.slu.cz

**Summary.** Eco-P colonies are constructed as natural extension of P colonies with dynamical evolution of environment. P colonies are one of the kind of computational models based on independent autonomous agents represented by membrane systems working and evolving in a shared environment. These acts are based on the set of program associated with every agent. There are two types of agents in eco-P colonies - senders and consumers. They differ by the type of programs. The eco-P colonies have a mechanism ($0L$ scheme) which can change the objects in the environment. We show that eco-P colonies with "active" environment and with two agents consumers can generate every recursive enumerable set of natural numbers and the family of sets of natural numbers computed by partially blind register machine is subset of the family of sets of natural numbers computed by eco-P colonies with "static" environment and one agent sender and one agent consumer.

## 1 Introduction

The computation model of eco-P colonies is based on two models of theoretical computer science: membrane systems and eco-colonies. Membrane systems (P systems) were introduced by Gheorghe Păun in [9] in 1998. From this time there are lots of types of P systems differing by type of rules, objects or by structure. More information about P systems can reader find in [10, 11], about eco-colonies in [4].

There are three types of entities in eco-P colonies. (1) The objects are symbols, they can be evolved or moved. (2) The agents (very simple one membrane systems) work according to their programs. In one step the agent can consume one object (transport it inside) or produce one object (transport it outside). Every agent contain two objects and this number of objects inside of agent stay constant during all the computation. (3) The environment of eco-P colony is used as communication channel for agents. Through the environment the agents are able to affect the behaviour of another agent. In the environment special objects occur, we call

* This research is partially supported by projects GAČR 201/09/P075, IGS 37/2009 (L. Ciencialová) and by research plan MSM 4781305903 (L. Cienciala).

them environmental and we denote them by $e$. There are sufficient number of copies of the object $e$. The environment can change independently to the agents. The evolution of the environment is independent from the states of agents and it is done by parallel using context free rules of $0L$ scheme to all possible objects placed in the environment.

The computation is parallel, in every step every agent nondeterministically chooses one of its applicable programs, if it has any, and executes it. Each object in the environment which is unused by agent is changed by $0L$ scheme. The computation ends by halting when no agent has applicable program. With every halting computation we associate the result of computation. It is the number of copies of specific object placed in the environment at the moment of halting of computation. We have to note, that at the end of computation some rules of $0L$ scheme are still applicable.

## 2 Definitions

Throughout the paper we assume the reader is familiar with basic of formal automata and language theory. We introduce notation used in the paper.

We use *NRE* to denote the family of recursively enumerable set of natural numbers and $N$ to denote the set of natural numbers.

$\Sigma$ is a notation for the alphabet. Let $\Sigma^*$ be set of all words over alphabet $\Sigma$ (except empty word $\varepsilon$). For the length of the word $w \in \Sigma^*$ we use notation $|w|$ and for the number of occurrences of symbol $a \in \Sigma$ in $w$ $|w|_a$.

A multiset of objects $M$ is a pair $M(V, f)$, where $V$ is an arbitrary (not necessarily finite) set of objects and $f$ is a mapping $f : V \to N$; $f$ assigns to each object in $V$ its multiplicity in $M$. The set of all multisets over the set of objects $V$ is denoted by $V^\circ$. The set $V'$ is called the support of $M$ and denoted by $supp(M)$ if for all $x \in V'$ $f(x) \neq 0$. The cardinality of $M$, denoted by $card(M)$, is defined by $card(M) = \sum_{a \in V} f(a)$. Any multiset of objects $M$ with the set of objects $V = \{a_i, \dots a_n\}$ can be represented as a string $w$ over alphabet $V$ with $|w|_{a_i} = f(a_i)$; $1 \leq i \leq n$. Obviously, all words obtained from $w$ by permuting the letters can also represent $M$, and $\varepsilon$ represents the empty multiset.

The mechanism of evolution in the environment is based on $0L$ schemes. It is a pair $(\Sigma, P)$, where $\Sigma$ is the alphabet of $0L$ scheme and $P$ is the set of context free rules, it fulfilled following condition $\forall a \in \Sigma \; \exists \alpha \in \Sigma^*$ such that $(a \to \alpha) \in P$. For $w_1, w_2 \in \Sigma^*$ we write $w_1 \Rightarrow w_2$ if $w_1 = a_1 a_1 \dots a_n, w_2 = \alpha_2 \alpha_2 \dots \alpha_n$, for $a_i \to \alpha_i \in P, 1 \leq i \leq n$.

A **register machine**[8] is the construct $M = (m, H, l_0, l_h, P)$ where:

- $m$ is a number of registers, $H$ is a set of instruction labels,
- $l_0$ is an initial/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 instructions 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, one can assume that in each $ADD$-instruction $l_1 : (ADD(r), l_2, l_3)$ and in each conditional $SUB$-instruction $l_1 : (SUB(r), l_2, l_3)$ the labels $l_1, l_2, l_3$ are mutually distinct. The register machine $M$ computes a set $N(M)$ of numbers in the following way: we start 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.) The family of sets of numbers computed by register machines is denoted by $NRM$. It is known (see e.g.[7]) that in this way we can compute all sets of numbers which are Turing computable $NRE$.

**Theorem 1.** *[8]* $NRM = NRE$.

Moreover, we call a register machine partially blind, if we interpret a subtract instruction in the following way: $l_1 : (S(r); l_2; l_3)$ - if register $r$ is not empty, then subtract one from its contents and go to instruction $l_2$ or to instruction $l_3$; if register $r$ is empty when attempting to decrement register $r$, then the program ends without yielding a result. When the register machine reaches the final state, the result obtained in the first register is only taken into account if the remaining registers are empty. The family of sets of non-negative integers generated by partially blind register machines is denoted by $NRM_{pb}$. The partially blind register machine accepts a proper subset of $NRE$.

**Theorem 2.** $NRM_{pb} \subset NRM$.

## 3 Eco-P colony

In this part we define the eco-P colony, the step and the result of the computation of eco-P colony.

**Definition 1.** *The eco-P colony is structure*
$\Pi = (A, e, f, V_E, D_E, B_1, \ldots, B_n)$, *where*

- $A$ *is the alphabet of the colony, its elements are called objects,*
- $e$ *is the basic (environmental) object of the colony,* $e \in A$,
- $f$ *is final object of the colony,* $f \in A$,
- $V_E$ *is the initial content of the environment,* $V_E \in (A - \{e\})^\circ$,
- $D_E$ *is* $0L$ *scheme* $(A, P_E)$, *where* $P_E$ *is the set of context free rules,*

- *$B_i$, $1 \leq i \leq n$, are the agents, every agent is the structure $B_i = (O_i, P_i)$, where $O_i$ is the multiset over $A$, it defines the initial state (content) of the agent $B_i$ and $|O_i| = 2$ and $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is the finite set of programs of two types: (1) generating $\langle a \rightarrow bc, d\ out \rangle$ - the program is applicable if agent contents objects $a$ and $d$. Object $a$ is used for generation of new content of the agent and object $d$ agent sends to the environment.*
  *(2) consuming $\langle ab \rightarrow c, d\ in \rangle$ - the program is applicable if the agent contents objects $a$ and $b$. These objects are evolved to one new object $c$ and object $d$ the agent imports from the environment.*

*Every agent has only one type of programs. The agent with generating programs is called **sender** and the agent with consuming programs is called **consumer**.*

**An initial configuration** of eco-P colony is $(n+1)$-tuple $(O_1, \ldots, O_n, V_E)$ of the multisets of objects placed in eco-P colony at the beginning of the computation, where $O_i$ ( $1 \leq i \leq n$ ) is content of the agent $B_i$ and $V_E$ is the multiset of object in the environment different from $e$. In general, the configuration of the eco-P colony $\Pi$ is defined as $(n + 1)$-tuple $(w_1, \ldots, w_n, w_E)$, where $w_i$ represents all objects inside of $i$-th agent, $|w_i| = 2$, $1 \leq i \leq n$, $w_E \in (A - \{e\})^\circ$ is composed by objects different from $e$ placed in the environment.

**The computation** of eco-P colonies is maximally parallel. It means that in every step the maximum number of agents works. Each agent who can use one or more of its program must be active. If the agent has more applicable programs, it nondeterministically chooses one program and executes it.

Let the programs of each $P_i$ be labelled in a one-to-one manner by labels in a set $lab\,(P_i)$ in such a way that $lab\,(P_i) \cap lab\,(P_j) = \emptyset$ for $i \neq j$, $1 \leq i, j \leq n$.

To define **the step of computation** we have to introduce following four functions: *whatsin*, *demand*, *putout*, *newin*. The first two functions assign the multisets of objects needed to execution of the program. The last two functions assign the multisets of objects placed inside and outside of agent after execution of given program. Formally, let $\langle ab \rightarrow c, d\ in \rangle$ be consuming program and $\langle a \rightarrow bc, d\ out \rangle$ be generating program, than we define functions:

$$whatsin(p_k) = \begin{cases} \{ab\} \text{ for consuming program } p_k \\ \{ad\} \text{ for generating program } p_k \end{cases}$$

$$demand(p_k) = \begin{cases} \{d\} \text{ for consuming program } p_k \\ \emptyset \text{ for generating program } p_k \end{cases}$$

$$newin(p_k) = \begin{cases} \{cd\} \text{ for consuming program } p_k \\ \{bc\} \text{ for generating program } p_k \end{cases}$$

$$putout(p_k) = \begin{cases} \emptyset \text{ for consuming program } p_k \\ \{d\} \text{ for generating program } p_k \end{cases}$$

Passing from the configuration to another one is defined as

$$(w_1, \ldots, w_n, w_E) \Rightarrow (w'_1, \ldots, w'_n, w'_E)\,,$$

where the following conditions are fulfilled:

1. The set of the labels of programs $P$ with $|P| \leq n$ is constructed in the way that

- $p, p' \in P$, $p \neq p'$, $p \in lab(P_j)$, $p' \in lab(P_i)$, $i \neq j$,
- for every $p \in P$, $p \in lab(P_j)$,
$$whatsin(p) = w_j \text{ and } \bigcup_{p \in P} demand(p) \subseteq w_E.$$

2. The set $P$ of selected labels of programs is maximal, it means that there is no other program with label $r \in \bigcup_{1 \leq i \leq n} lab(P_i)$, $r \notin P$, which can be add to the set $P$ such that previous conditions will be fulfilled.
Generally for every $j$, $1 \leq j \leq n$, for which there exists $p \in P$, such that $p \in lab(P_j)$, let $w'_j = newin(p)$. If there is no $p \in P$, $p \in lab(P_j)$ for some $j$, $1 \leq j \leq n$, let $w'_j = w_j$. Let $w_E - \bigcup_{p \in P} demand(p) \Rightarrow_{D_E} w''_E$ be the step of derivation in $0L$ scheme $(A, P_E)$ and then $w'_E = w''_E \cup \bigcup_{p \in P} putout(p)$.

The union and "-" are operations over multisets.

The configuration is final if the set $P$ cannot be chosen to be other than the empty set. The set of the final configurations we denote by $H$. If the computation halts, we can obtain a result. The result of computation is given by the number of objects $f$ placed in the environment at the end of the computation. The set of the numbers computed by eco-P colony $\Pi$ is defined as

$N(\Pi) = \{|w_E|_f \mid (O_1, \ldots, O_n, V_E) \Rightarrow^* (w_1, \ldots, w_n, w_E) \in H\}$,
where $(O_1, \ldots, O_n, V_E)$ is the initial configuration, $(w_1, \ldots, w_n, w_E)$ is the final configuration, and $\Rightarrow^*$ denotes reflexive and transitive closure of $\Rightarrow$.

Let $\Pi = (A, e, f, V_E, D_E, B_1, \ldots, B_n)$ be eco-P colony. The maximal number of programs associated with one agent we call the height and the degree of eco-P colony $\Pi$ is the number of agents in $\Pi$.

We denote $NEPCOL_{x,y,z}(n,h)$ the family of the sets computing by eco-P colonies such that:

- $x$ can be formed by two symbols: $s$, $c$. $s$ - if there is agent sender in eco-P colony, $c$ - if there is agent consumer in eco-P colony,
- $y = passive$ if the rules of $0L$ scheme are of type $a \to a$ only,
- $y = active$ if the set of rules of $0L$ scheme disposes of at least one rule of another type than $a \to a$,
- $z = ini$ if the environment or agents contains objects different from $e$, otherwise we eliminate this notation,
- the degree of eco-P colony is at most $n$ and
- the height is at most $h$.

We compare eco-P colonies with above-mentioned computation models. In [2] the author shows that:

**Theorem 3.** [2] $NEPCOL_{sc,passive}(3,*) = NRE$.

We prove that eco-P colony with active environment and with two agents consumers can generate every recursive enumerable set of natural numbers.

**Theorem 4.** $NEPCOL_{c,active,ini}(2,*) = NRE$.

*Proof.* Consider register machine $M = (m, H, l_0, l_h, P)$. All labels from the set $H$ are objects in eco-P colony. The content of register $i$ is represented by the number of copies of objects $a_i$ placed in the environment.

At the beginning of computation there are object $l_0$ and auxiliary object $D$ in the environment. Object $l_0$ corresponds with initial label of instruction of $M$.

The instruction $l_i = (ADD(r), l_j, l_k)$ will be realized by rules:

| $ENV$ : | $B_1$ : |
|---|---|
| $1 : l_i \to a_r l_i' D;$ | $5 : \langle Pe \to P; \overline{l_j}\ in \rangle;$ |
| $2 : l_i' \to \overline{l_j} \overline{l_k} D;$ | $6 : \langle Pe \to P; \overline{l_k}\ in \rangle;$ |
| $3 : \overline{l_j} \to l_j D;$ | $7 : \langle P\overline{l_j} \to P; e\ in \rangle;$ |
| $4 : \overline{l_k} \to l_k D;$ | $8 : \langle P\overline{l_k} \to P; e\ in \rangle;$ |

The computation is done in such a way that $0L$ scheme works in the environment, it executes adding one to the content of register $r$ (generate one copy of object $a_r$ - the rule number 1) and generating of the objects $l_j$ and $l_k$, labels of all instructions which will be possibly executed in the next steps of computation of the register machine $M$ (the rule 2). In the next step agent consumer $B_1$ takes one of these objects inside the agent - the rule 5 or 6. In the next step instruction $l_j$ or $l_k$ will be simulated.

In the eco-P colony the instruction $l_i : (SUB(r), l_j, l_k)$ is realized by following rules and programs:

| $ENV$ : | $B_1$ : |
|---|---|
| $9 : l_i \to (l_i) l_i^R D;$ | $16 : \langle Pe \to n_r; (l_i)\ in \rangle;$ |
| $10 : l_i^R \to l_i^P D;$ | $17 : \langle (l_i) n_r \to L_k; a_r\ in \rangle;$ |
| $11 : l_i^P \to \boxed{l_j}\ \boxed{l_k} D;$ | $18 : \left\langle (l_i) n_r \to R; \boxed{l_j}\ in \right\rangle;$ |
| $12 : \boxed{l_j} \to l_j'' D;$ | $19 : \left\langle a_r L_k \to R; \boxed{l_k}\ in \right\rangle;$ |
| $13 : \boxed{l_k} \to l_k'' D;$ | $20 : \left\langle \boxed{l_j} R \to P; e\ in \right\rangle;$ |
| $14 : l_j'' \to l_j D;$ | $21 : \left\langle \boxed{l_k} R \to P; e\ in \right\rangle;$ |
| $15 : l_k'' \to l_k D;$ | |

If there is the object $l_i$ (the label of SUB-instruction) in the environment, $0L$ scheme generates (using the rule no. 9) the object $(l_i)$. This is the message for the agent $B_1$, that the agent has to try to consume one copy of object $a_r$ from the environment (try to subtract one from the content of register $r$.)

If the agent is successful (agent used program 17) in the next step agent consume object $\boxed{l_k}$ and computation will follow with instruction labelled $l_j$ because object $\boxed{l_j}$ is present in the environment.

If the agent do not consume object $a_r$ (register $r$ contents 0, there was no one object $a_r$ in the environment), in the next step the agent take object $\boxed{l_j}$ from the environment and computation will proceed with instruction labelled $l_k$.

For the halting instruction there is rule $l_h \to l_h$ in $0L$ scheme, this rule is of the same type as rules for other objects, which are not changed by environment during all the computation ( for example $e$, $a_r, \ldots$).

During the computation there are moments when the agent $B_1$ has no applicable program. It means that computation will be terminated in such moment. To solve this problem we add one more agent to the eco-P colony (agent $B_2$), it has to work during all computation. The agent $B_2$ has only one program $\langle PD \to P; D\ in \rangle$.

We construct eco-P colony $\Pi = (A, e, f, V_E, D_E, B_1, B_2)$ with:

- alphabet $A = \{l_i, l_i', l_i'', (l_i), \boxed{l_i}, L_i \mid \text{for each } l_i \in H\} \cup \{a_i \mid 1 \leq i \leq m\} \cup \{e, R, P, D\}$
- final object $f = a_1$,
- initial content of the environment $V_E = l_0 D$, $0L$ scheme $D_E = (A, P_E)$
- set of rules of the environment $P_E = \{a_i \to a_i \ \ 1 \leq i \leq m\} \cup \{e \to e\} \cup$ $\cup$ {the rules already mentioned above}
- and the agents $B_1 = (Pe, P_1)$, $B_2 = (PD, P_2)$, the sets of programs are described above.

Eco-P colony starts its computation with object $l_0$ in the environment and simulation of instruction labelled $l_0$. By the rules and programs it places and deletes from the environment the objects $a_r$ and halts its computation when object $l_h$ appears in the environment. The result of computation is the number of copies of object $a_1$ placed in the environment at the end of computation. No other computation can be executed in eco-P colony. So the computation in the eco-P colony $\Pi$ correctly simulates computation in register machine.

Eco-P colonies with "active" environment and with two agents - consumers can generate every recursively enumerable set of natural numbers. The question is if an eco-P colony with "static" environment ($0L$ scheme contains the rule of type $a \to a$ only) can generate it too.

**Theorem 5.** $NEPCOL_{sc,passive}(2, *) \supseteq NRM_{pb}$.

*Proof.* (Draft) Let us consider partially blind register machine $M = (m, H, l_0, l_h, P)$. For all labels from the set $H$ we construct corresponding objects in eco-P colony $\Pi$. The content of register $i$ will be represented by the number of copies of objects $a_i$ placed in the environment.

At the beginning of computation there are only copies environmental object $e$ in the environment. In eco-P colony $\Pi$ there are two agents: agent $B_1$, which is sender, and agent $B_2$, it is consumer.

**$B_1$**

$0 : \langle e \to l_0 e; e\ out \rangle$ ;

The object $l_0$ corresponds to the label of the first instruction realized by register machine and it is own by agent $B_1$.

The instruction $l_i = (ADD(r), l_j, l_k)$ is realized by following programs:

**$B_1$**

$1 : \langle l_i \to a_r l_i'; e\ out \rangle$ ; $2 : \langle l_i' \to l_j e; a_r\ out \rangle$ ; $3 : \langle l_i' \to l_k e; a_r\ out \rangle$ ;

The agent $B_1$ places to the environment step by step objects: $a_r$ - adding one to the content of register $r$ ( program 1) and object $l_j$ (program 2) or object $l_k$ (program 3).

The instruction $l_i : (SUB(r), l_j, l_k)$ is in eco-P colony realized by following programs:

**$B_1$**

$4 : \langle l_i \to l_i^R (l_i); e\ out \rangle$; $5 : \langle l_i^R \to l_i^{R1} e; (l_i)\ out \rangle$; $6 : \langle l_i^{R1} \to l_i^{R2} l_i''; e\ out \rangle$;
$7 : \langle l_i^{R2} \to \overline{l_j} e; l_i''\ out \rangle$; $8 : \langle l_i^{R2} \to \overline{l_k} e; l_i''\ out \rangle$; $9 : \left\langle \overline{l_j} \to \overline{l_j'} e; e\ out \right\rangle$;
$10 : \left\langle \overline{l_k} \to \overline{l_i'} e; e\ out \right\rangle$; $11 : \left\langle \overline{l_j'} \to l_j e; e\ out \right\rangle$; $12 : \left\langle \overline{l_k'} \to l_k e; e\ out \right\rangle$;

**$B_2$**

$13 : \langle ee \to n_r; (l_i)\ in \rangle$; $14 : \langle (l_i) n_r \to L_i; a_r\ in \rangle$; $15 : \langle (l_i) n_r \to R; l_i''\ in \rangle$;
$16 : \langle R l_i'' \to R; e\ in \rangle$; $17 : \langle Re \to R; e\ in \rangle$; $18 : \langle L_i a_r \to e; l_i''\ in \rangle$;
$19 : \langle l_i'' e \to e; e\ in \rangle$;

The subtracting is done in four phases: (1) It starts with object $l_i$ inside agent $B_1$ corresponding with a label of some SUB-instruction. The agent places object $(l_i)$ to the environment (programs 4 and 5). It is a message for agent $B_2$ to try to consume object $a_r$ from the environment. (2) The agent $B_2$ consume object $(l_i)$ (program 13) and then object $a_r$ (program 14). (3) In last step we describe at the same time there are two applicable programs 14 and 15. The execution of the program 15 make the computation endless. Program 17 is the only one applicable program and it is circling. Program 15 must be used if there is no object $a_r$ in the environment (this is the case of unsuccessful subtracting- register $r$ stores value zero). Because of nondeterminism there exists computation correctly decreasing the number of copies of $a_r$ in the environment if this is possible. (4) Agent $B_1$ work independently from agent $B_2$. It continues the computation, it means that it generates labels of instructions and objects $a_r$ regardless of success or failure of removing demanded object by agent $B_2$.

For instruction $l_h$ there is no program applicable in agent $B_1$. If every subtracting instruction was successfully executed agent $B_2$ has no applicable program too. Then computation halts. The result is the number of object $a_1$ placed in the environment and it corresponds to the result of successful computation in the partially blind register machine.

We construct eco-P colony $\Pi = (A, e, f, V_E, D_E, B_1, B_2)$ with alphabet $A = \{l_i, l_i', l_i'', (l_i), L_i, l_i^R, , l_i^{R1}, l_i^{R2} |$ for each $l_i \in H\} \cup \{a_i \mid 1 \le i \le m\} \cup \{e, R\}$, final object $f = a_1$, initial state of the environment $V_E = \varepsilon$, $0L$ scheme $D_E = (A, P_E)$, the set of rules of environment $P_E = \{x \to x \mid \forall x \in A\}$ and with agents $B_1 = (ee, P_1)$, $B_2 = (ee, P_2)$. The sets of programs we describe in previous paragraphs.

We prove that $NRM_{pb}$ is the subset of the family the sets of natural numbers generated by eco-P colonies with one agent sender and one agent consumer.

## 4 Conclusions

In this paper we presented the results obtained during research of eco-P colonies - the extended model of P colonies. We show that eco-P colonies with active environment and with two agents consumers can compute every recursively enumerable

set of natural numbers. We show that the family of the sets of natural numbers computed by partially blind register machine is subset of the family of sets of natural numbers computed by eco-P colonies with static environment and one agent consumer and one agent sender.

## References


1. L. Cienciala, L. Ciencialová, A. Kelemenová. On the number of agents in P colonies. In: *Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers.* Edited by G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa. Volume 4860 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin-Heidelberg, 2007, 193-208.
2. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, G. Vaszil. On Very Simple P Colonies, Proceeding of The seventh Brainstorming Week on Membrane Computing,Sevilla 2009.
3. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun. *Grammar Systems – A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach, London, 1994.
4. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun: Eco-grammar Systems. *Grammatical Framework for Studying Lifelike Interactions.* Artificial Life 3, 1997, l-28.
5. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil. Computing with cells in environment: P colonies. *Journal of Multi-Valued Logic and Soft Computing* 12:201-215, 2006.
6. J. Kelemen, A. Kelemenová. On P colonies, a biochemically inspired model of computation. Proc. of the $6^{th}$ International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, 40-56.
7. J. Kelemen, A. Kelemenová, Gh. Păun. Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX).* Edited by M. Bedau et al. Boston Mass., 2004, 82-86.
8. M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967.
9. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences* 61, 2000, 108-143.
10. Gh. Păun. *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
11. P systems web page. January 15 2001. April 23 2009 <http://ppage.psystems.eu>