On the Number of Agents in P Colonies

Luděk Cienciala[1], Lucie Ciencialová[1], and Alica Kelemenová[1,2]

[1] Institute of Computer Science, Silesian University in Opava, Czech Republic
[2] Department of Computer Science, Catholic University Ružomberok, Slovakia
{ludek.cienciala,lucie.ciencialova,alica.kelemenova}@fpf.slu.cz

Abstract. We continue the investigation of P colonies introduced in [8], a class of abstract computing devices composed of independent membrane agents, acting and evolving in a shared environment.

We decrease the number of agents sufficient to guarantee computational completeness of P colonies with one and with two objects inside each agent, respectively, owing some special restrictions to the type of programs. We characterize the generative power of the partially blind machine by the generative power of special P colonies.

1 Introduction

P colonies were introduced in [8] as formal models of a computing device inspired by membrane systems ([10]) and by grammar systems called colonies ([6]). This model intends to structure and functioning of a community of living organisms in a shared environment.

The independent organisms living in a P colony are called agents. Each agent is given by a collection of objects embedded in a membrane. The number of objects inside the agent is the same for each one of them. The environment contains several copies of a basic environmental object denoted by e. The number of the copies of e is unlimited.

A set of programs is associated with each agent. The program determines the activity of the agent by rules. In every moment of computation all the objects inside of the agent are being either evolved (by an evolution rule) or transported (by a communication rule). Two such rules can also be combined into checking rule which specifies two possible actions: if the first rule is not applicable then the second one should be applied. So it sets the priority between two rules.

The computation starts in the initial configuration. Using their programs the agents can change their objects and possibly objects in the environment. This gives possibility to affect the behavior of the other agents in next steps of computation. In each step of the computation, each agent with at least one applicable program nondeterministically chooses one of them and executes it. The computation halts when no agent can apply any of its programs. The result of the computation is given by the number of some specific objects present at the environment at the end of the computation.

There are several different ways used how to define the beginning of the computation. (1) At the beginning of computation the environment and all agents

G. Eleftherakis et al. (Eds.): WMC8 2007, LNCS 4860, pp. 193–208, 2007.

contain only copies of object e. (2) All the agents can contain various objects at the beginning of computation - the agents are in different initial states. The environment contains only copies of object e. (3) The initial state of the environment is nonempty (there are some object different from the object e) - the environment contains initial "parameters" for future computation, while the agents start with e-s.

In [4,7,8] the authors study P colonies with two objects inside the agents. In this case programs consist of two rules, one for each object. If the former of these rules is an evolution and the latter is a communication or checking, we speak about restricted P colonies. If also another combination of the types of the rules is used, we obtain non-restricted P colonies. The restricted P colonies with checking rules are computationally complete [3,4].

In the present paper we study properties of restricted P colonies without checking rules and computational power of P colonies with one object and the minimal number of agents.

We start with definitions in Section 2.

In Section 3 we will deal with P colonies with one object inside each agent. In [1] there was shown that at most seven programs for each agent as well as five agents guarantee the computational completeness of these P colonies. In the preset paper we look for the generative power of P colonies with less than five agents. Two results are achieved in this direction. First, we show, that four agents are enough for computational completeness of P colonies. The second result gives a lower bound for the generative power the P colonies with two agents. Even a restricted variant of these P colonies is at least as powerful as the partially blind register machines.

Restricted P colonies are studied in Section 4. It is known that one agent is sufficient to obtain computational completeness of restricted P systems with checking rules ([4]). For the restricted P colonies that do not use checking rules we will prove that two agents are sufficient to obtain the universal computational power.

2 Definitions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory. For more information on membrane computing, we recommend [11]. We briefly summarize the notation used in the present paper.

We use NRE to denote the family of the recursively enumerable sets of non-negative integers and N to denote the set of non-negative integers.

Let Σ be an alphabet. Let Σ^* be the set of all words over Σ (including the empty word ε). We denote the length of the word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$.

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 finite multisets over the finite set V is denoted by V°. The support of M is the set $supp(M) = \{a \in V \mid f_M(a) \neq 0\}$.

The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f_M(a)$. Any finite multiset M over V can be represented as a string w over alphabet V with $|w|_a = f_M(a)$ for all $a \in V$. Obviously, all words obtained from w by permuting the letters can also represent the same M, and ε represents the empty multiset. For multiset M represented by word w we use the notation ${}_\star w$.

2.1 P Colonies

We briefly recall the notion of P colonies introduced in [8]. A P colony consists of agents and environment. Both the agents and the environment contain objects. With every agent a set of programs is associated. There are two types of rules in the programs. The first type, called evolution rules, are of the form $a \rightarrow b$. It means that object a inside of the agent is rewritten (evolved) to the object b. The second type of rules, called communication rules, are of the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and the object d outside of the agent change their positions, so, after execution of the rule object d appears inside the agent and c is placed outside in the environment.

In [7] the ability of agents was extended by checking rule. Such a rule gives to the agents an opportunity to choose between two possibilities. It has the form r_1/r_2. If the checking rule is performed, the rule r_1 has higher priority to be executed than the rule r_2. It means that the agent checks the possibility to use rule r_1. If it can be executed, the agent has to use it. If the rule r_1 cannot be applied, the agent uses the rule r_2.

Definition 1. *A P colony of the capacity c is a construct*
$\Pi = (A, e, f, {}_\star v_E, B_1, \ldots, B_n)$, *where:*

- *A is an alphabet whose elements are called objects,*
- *e is the basic object of the colony, $e \in A$,*
- *f is the final object of the colony, $f \in A$,*
- *${}_\star v_E$ is a multiset over $A - \{e\}$,*
- *B_i, $1 \le i \le n$, are agents; each agent is a construct $B_i = ({}_\star o_i, P_i)$, where*
  - *${}_\star o_i$ is a multiset over A which determines the initial state (content) of agent B_i and $|{}_\star o_i| = c$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where each program contains exactly c rules, which are in one of the following forms each:*
    - ◇ *$a \rightarrow b$, called an evolution rule,*
    - ◇ *$c \leftrightarrow d$, called a communication rule,*
    - ◇ *r_1/r_2, called a checking rule; r_1, r_2 are evolution or communication rules.*

The initial configuration of the P colony is the $(n+1)$-tuple of multisets of objects present in the P colony at the beginning of the computation, i.e., $({}_\star o_1, \ldots, {}_\star o_n, {}_\star v_E)$. Formally, a configuration of P colony Π is given by $({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E)$, where $|{}_\star w_i| = c$, $1 \le i \le n$, ${}_\star w_i$ represents all the objects placed inside the i-th agent and ${}_\star w_E \in (A - \{e\})^\circ$ represents all the objects in the environment different from the object e.

In this paper, the parallel model of P colonies will be studied. At each step of a parallel computation, each agent which can use one of its programs should use one. If the number of applicable programs is higher than one, the agent nondeterministically chooses one of them.

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

To express derivation step formally we introduce the following four functions. For a rule r being $a \rightarrow b, c \leftrightarrow d$, and $c \leftrightarrow d/c' \leftrightarrow d'$, respectively, and for multiset ${}_\star w \in A^\circ$ we define:

$$\begin{array}{ll}
left\,(a \rightarrow b, {}_\star w) = {}_\star a & left\,(c \leftrightarrow d, {}_\star w) = {}_\star \varepsilon \\
right\,(a \rightarrow b, {}_\star w) = {}_\star b & right\,(c \leftrightarrow d, {}_\star w) = {}_\star \varepsilon \\
export\,(a \rightarrow b, {}_\star w) = {}_\star \varepsilon & export\,(c \leftrightarrow d, {}_\star w) = {}_\star c \\
import\,(a \rightarrow b, {}_\star w) = {}_\star \varepsilon & import\,(c \leftrightarrow d, {}_\star w) = {}_\star d
\end{array}$$

$$\begin{array}{l}
left\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star \varepsilon \\
right\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star \varepsilon \\
\left.\begin{array}{l} export\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star c \\ import\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star d \end{array}\right\} \text{for } |{}_\star w|_d \geq 1 \\
\left.\begin{array}{l} export\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star c' \\ import\,(c \leftrightarrow d/c' \leftrightarrow d', {}_\star w) = {}_\star d' \end{array}\right\} \text{for } |{}_\star w|_d = 0 \text{ and } |{}_\star w|_{d'} \geq 1
\end{array}$$

For a program p and any $\alpha \in \{left, right, export, import\}$, let

$$\alpha\,(p, {}_\star w) = \cup_{r \in p} \alpha\,(r, {}_\star w).$$

A transition from a configuration to another one is denoted as

$({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E) \Rightarrow ({}_\star w'_1, \ldots, {}_\star w'_n, {}_\star w'_E)$, where the following conditions are satisfied:

- There is a set of program labels P with $|P| \leq n$ such that
  - $p, p' \in P$, $p \neq p'$, $p \in lab\,(P_j)$, $p' \in lab\,(P_i)$, $i \neq j$,
  - for each $p \in P$, $p \in lab\,(P_j)$, $left\,(p, {}_\star w_E) \cup export\,(p, {}_\star w_E) = {}_\star w_j$, and $\bigcup_{p \in P} import\,(p, {}_\star w_E) \subseteq {}_\star w_E$.
- Furthermore, the chosen set P is maximal, that is, if any other program $r \in \bigcup_{1 \leq i \leq n} lab\,(P_i)$, $r \notin P$, is added to P, then the conditions above are not satisfied.

Finally, for each j, $1 \leq j \leq n$, for which there exists a $p \in P$ with $p \in lab\,(P_j)$, let $w'_j = right\,(p, {}_\star w_E) \cup import\,(p, {}_\star w_E)$. If there is no $p \in P$ with $p \in lab\,(P_j)$ for some j, $1 \leq j \leq n$, then let ${}_\star w'_j = {}_\star w_j$ and moreover, let

$${}_\star w'_E = {}_\star w_E - \bigcup_{p \in P} import\,(p, {}_\star w_E) \cup \bigcup_{p \in P} export\,(p, {}_\star w_E).$$

Union and "–" here are multiset operations.

A configuration is halting if the set of program labels P satisfying the conditions above cannot be chosen to be other than the empty set. A set of all possible halting configurations is denoted by H. With a halting computation we can associate a result of the computation, given by the number of copies of the special symbol f present in the environment. The set of numbers computed by a P colony Π is defined as

$$N(\Pi) = \left\{ |_\star w_E|_f \;\mid\; ({}_\star o_1, \ldots, {}_\star o_n, {}_\star v_E) \Rightarrow^* ({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E) \in H \right\},$$

where $({}_\star o_1, \ldots, {}_\star o_n, {}_\star v_E)$ is the initial configuration, $({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

Given a P colony $\Pi = (A,\ e,\ f,\ {}_\star v_E,\ B_1,\ \ldots,\ B_n)$ 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 of the agents.

Let us use the following notations: $NPCOL_{par}(c, n, h)$ is the family of all sets of numbers computed by P colonies working in parallel, using no checking rules, and with the capacity at most c, the degree at most n, and the height at most h. If the checking rules are allowed, the family of all sets of numbers computed by P colonies is denoted by $NPCOL_{par}K$. If the P colonies are restricted, we use notation $NPCOL_{par}R$ and $NPCOL_{par}KR$, respectively.

2.2 Register Machines

In this paper we characterize the size of the families $NPCOL_{par}(c, n, h)$ comparing them with the recursively enumerable sets of numbers. To achieve this aim we use the notion of a register machine.

Definition 2. [9] *A register machine is a 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 start label, l_h is the final label, P is a finite set of instructions injectively labeled with the elements from the 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 content of the register r and proceed to the instruction (labeled with) l_2 or l_3.
$l_1 : (SUB(r), l_2, l_3)$	If the register r stores a value different from zero, then subtract 1 from its content and go to instruction l_2, otherwise proceed to instruction l_3.
$l_h : HALT$	Halt 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 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: it starts with all registers empty (hence storing the number zero) with the instruction labeled l_0 and it proceeds to apply the instructions as indicated by the labels (and made possible by the contents of registers). If it reaches 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., [9]) that in this way we compute all Turing computable sets.

Moreover, we call a register machine partially blind [5], if we interpret a subtract instruction $l_1 : (SUB(r); l_2; l_3)$ in the following way: if the value of register r is different from zero, then subtract one from its contents and go to instruction l_2 or to instruction l_3; if in register r is stored zero, then the program ends without yielding a result.

When the partially blind register machine reaches the final state, the result obtained in the first register is taken into account if the remaining registers store value zero. The family of sets of non-negative integers generated by partially blind register machines is denoted by NRM_{pb}. The partially blind register machines accept a proper subset of NRE.

3 P Colonies with One Object Inside the Agent

In this section we analyze the behavior of P colonies with only one object inside each agent. Each program in this case is formed by only one rule, either an evolution or a communication.

If all the agents have their programs with evolution rules, the agents "live only for themselves" and do not communicate with the environment.

In [1] the following results were proved:

$NPCOL_{par}K(1, *, 7) = NRE$,
$NPCOL_{par}K(1, 5, *) = NRE$.

The number of agents in the second result can be decreased. This is demonstrated by the following theorem.

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

Proof. We construct a P colony simulating the computation of a register machine. Because there are only copies of e in the environment and inside the agents in the initial configuration, we will initialize a computation by generating the initial label l_0. After generating the symbol l_0 this agent stops and it can start its activity only by using a program with a communicating rule.

Two agents will cooperate in order to simulate the ADD and SUB instructions.

Let us consider a register machine $M = (m, H, l_0, l_h, P)$. We can represent the content m_i of the register i by m_i copies of the specific object a_i in the environment. We construct the P colony $\Pi = (A, e, f, {}_\star\varepsilon, B_1, \ldots, B_4)$ with:

– alphabet $A = \{l, l' | l \in H\}$
$\cup \{E_i, E_i', F_i, F_i', F_i'' \mid \text{ for each } l_i \in H\}$
$\cup \{a_i \mid 1 \leq i \leq m\} \cup \{e, d, m, C\}$,

– $f = a_1$,

– $B_i = ({}_\star e, P_i),\ 1 \leq i \leq 4$, where P_i will be specified in the next steps of the proof.

The programs in P_1 serve for the initialization of the computation and in the simulation of SUB instructions, programs in P_2 have an auxiliary character. The programs in P_3 and in P_4 realize ADD and SUB instructions.

(1) To initialize the simulation of a computation of M we take an agent $B_1 = ({}_\star e, P_1)$ with the set of programs:

P_1 :
$1 : \langle e \to l_0 \rangle$, $2 : \langle l_0 \leftrightarrow d \rangle$;

(2) We need one more agent to generate a special object d. While object C is not in the environment the agent B_2 places a further copy of d to the environment.

P_2 :
$3 : \langle e \to d \rangle$, $4 : \langle d \leftrightarrow C / d \leftrightarrow e \rangle$;

The P colony Π starts its computation in the initial configuration $({}_\star e, {}_\star e, {}_\star e, {}_\star e, {}_\star \varepsilon)$. In the first subsequence of steps of P colony Π only agents B_1 and B_2 can apply their programs.

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star e$		1	3		
2.	${}_\star l_0$	${}_\star d$	${}_\star e$	${}_\star e$			4		
3.	${}_\star l_0$	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star d$	2	3		
4.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_0$				

(3) To simulate the ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ two agents B_3 and B_4 are used in Π. These agents help each other to add one copy of object a_r and object l_2 or l_3 to the environment using the following programs:

P_3	P_3	P_4	P_4
$5 : \langle e \leftrightarrow l_1 \rangle$,	$11 : \langle E_1' \to l_2' \rangle$,	$15 : \langle e \leftrightarrow E_1 \rangle$,	$21 : \langle e \leftrightarrow l_2' \rangle$,
$6 : \langle l_1 \to E_1 \rangle$,	$12 : \langle E_1' \to l_3' \rangle$,	$16 : \langle E_1 \to E_1' \rangle$,	$22 : \langle e \leftrightarrow l_3' \rangle$,
$7 : \langle E_1 \leftrightarrow d \rangle$,	$13 : \langle l_2' \leftrightarrow e \rangle$,	$17 : \langle E_1' \leftrightarrow e \rangle$,	$23 : \langle l_2' \to l_2 \rangle$,
$8 : \langle d \to L_1 \rangle$,	$14 : \langle l_3' \leftrightarrow e \rangle$,	$18 : \langle e \leftrightarrow L_1 \rangle$,	$24 : \langle l_3' \to l_3 \rangle$,
$9 : \langle L_1 \leftrightarrow E_1' / L_1 \to m \rangle$,		$19 : \langle L_1 \to a_r \rangle$,	$25 : \langle l_2 \leftrightarrow e \rangle$,
$10 : \langle m \to d \rangle$,		$20 : \langle a_r \leftrightarrow e \rangle$,	$26 : \langle l_3 \leftrightarrow e \rangle$.

The agent B_3 consumes the object l_1, changes it to E_1 and places it to the environment. The agent B_4 borrows E_1 from the environment and returns E_1'. B_3 rewrites the object d to some L_i. If this L_i has the same index as E_i' placed in the environment, the computation can go to the next phase. If indices of L_i and E_i are different, the agent B_3 tries to generate another L_i. If the computation gets over this checking step, agent B_4 generates one copy of object a_r and places it into the environment (adding 1 to the content of register r). Then agent B_3 generates the helpful object l_2' or l_3' and places it into the environment. The agent B_4 exchanges it for the "valid label" l_2 or l_3.

An instruction $l_i : (ADD(r), l_j, l_k)$ is simulated by the following sequence of steps. Let the content of the agent B_2 be d.

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_i a_r^u d^v$		4	5	
2.	${}_\star d$	${}_\star e$	${}_\star l_i$	${}_\star e$	${}_\star a_r^u d^{v+1}$		3	6	
3.	${}_\star d$	${}_\star d$	${}_\star E_i$	${}_\star e$	${}_\star a_r^u d^{v+1} d$		4	7	
4.	${}_\star d$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star E_i a_r^u d^{v+1}$		3	8	15
5.	${}_\star d$	${}_\star d$	${}_\star L_i$	${}_\star E_i$	${}_\star a_r^u d^{v+1}$		4		16
6.	${}_\star d$	${}_\star e$	${}_\star L_i$	${}_\star E'_i$	${}_\star a_r^u d^{v+2}$		3		17
7.	${}_\star d$	${}_\star d$	${}_\star L_i$	${}_\star e$	${}_\star E'_i a_r^u d^{v+2}$		4	9	
8.	${}_\star d$	${}_\star e$	${}_\star E'_i$	${}_\star e$	${}_\star L_i a_r^u d^{v+3}$		3	**11** or 12	18
9.	${}_\star d$	${}_\star d$	${}_\star l'_j$	${}_\star L_i$	${}_\star a_r^u d^{v+3}$		4	13	19
10.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star a_r$	${}_\star l'_j a_r^u d^{v+4}$		3		20
11.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l'_j a_r^{u+1} d^{v+4}$		4		21
12.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l'_j$	${}_\star a_r^{u+1} d^{v+5}$		3		23
13.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star l_j$	${}_\star a_r^{u+1} d^{v+5}$		4		25
14.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star l_j a_r^{u+1} d^{v+6}$				

(4) For each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the next programs are introduced in the sets P_1, P_3, and in the set P_4:

P_3	P_3	P_1	P_4
$27 : \langle e \leftrightarrow l_1 \rangle$,	$33 : \langle F''_1 \rightarrow l'_3 \rangle$,	$36 : \langle d \leftrightarrow F_1 \rangle$,	$41 : \langle e \leftrightarrow l'_2 \rangle$,
$28 : \langle l_1 \rightarrow F_1 \rangle$,	$34 : \langle l'_2 \leftrightarrow e \rangle$,	$37 : \langle F_1 \rightarrow F'_1 \rangle$,	$42 : \langle e \leftrightarrow l'_3 \rangle$,
$29 : \langle F_1 \leftrightarrow d \rangle$,	$35 : \langle l'_3 \leftrightarrow e \rangle$;	$38 : \langle F'_1 \leftrightarrow a_r / F'_1 \rightarrow F''_1 \rangle$,	$43 : \langle l'_2 \rightarrow l_2 \rangle$,
$30 : \langle d \leftrightarrow F'_1 \rangle$,		$39 : \langle a_r \rightarrow d \rangle$,	$44 : \langle l'_3 \rightarrow l_3 \rangle$,
$31 : \langle F'_1 \rightarrow l'_2 \rangle$,		$40 : \langle F''_1 \leftrightarrow d \rangle$,	$45 : \langle l_2 \leftrightarrow e \rangle$,
$32 : \langle d \leftrightarrow F''_1 \rangle$,			$46 : \langle l_3 \leftrightarrow e \rangle$.

Agent B_3 starts the simulation of executing SUB-instruction l_1, the agent B_1 checks whether there is a copy of the object a_r in the environment or not and gives this information (F'_1 – if there is some a_r; F''_1 – if there is no object a_r in the environment) to the environment.

An instruction $l_i : (SUB(r), l_j, l_k)$ is simulated by the following sequence of steps. The computation for 0 in the register r is given below.

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_i d^v$		4	27	
2.	${}_\star d$	${}_\star e$	${}_\star l_i$	${}_\star e$	${}_\star d^{v+1}$		3	28	
3.	${}_\star d$	${}_\star d$	${}_\star F_i$	${}_\star e$	${}_\star d^{v+1} d$		4	29	
4.	${}_\star d$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star F_i d^{v+1}$	36	3		
5.	${}_\star F_i$	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star d^{v+2}$	37	4		
6.	${}_\star F'_i$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star d^{v+3}$	38	3		

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
7.	${}_\star F_i''$	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star d^{v+3}$	40	4		
8.	${}_\star d$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star F_i'' d^{v+3}$		3	32	
9.	${}_\star d$	${}_\star d$	${}_\star F_i''$	${}_\star e$	${}_\star d^{v+4}$		4	33	
10.	${}_\star d$	${}_\star e$	${}_\star l_k'$	${}_\star e$	${}_\star d^{v+5}$		3	35	
11.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_k' d^{v+5}$		4		42
12.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_k'$	${}_\star d^{v+6}$		3		44
13.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star l_k$	${}_\star d^{v+6}$		4		46
14.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star l_k d^{v+7}$				

The computation for a value different from 0 in the register r:

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_i a_r^u d^v$		4	27	
2.	${}_\star d$	${}_\star e$	${}_\star l_i$	${}_\star e$	${}_\star a_r^u d^{v+1}$		3	28	
3.	${}_\star d$	${}_\star d$	${}_\star F_i$	${}_\star e$	${}_\star a_r^u d^{v+1} d$		4	29	
4.	${}_\star d$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star F_i a_r^u d^{v+1}$	36	3		
5.	${}_\star F_i$	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star a_r^u d^{v+2}$	37	4		
6.	${}_\star F_i'$	${}_\star e$	${}_\star d$	${}_\star e$	${}_\star a_r^u d^{v+3}$	38	3		
7.	${}_\star a_r$	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star F_i a_r^{u-1} d^{v+3}$	39	4	30	
8.	${}_\star d$	${}_\star e$	${}_\star F_i'$	${}_\star e$	${}_\star a_r^{u-1} d^{v+5}$		3	31	
9.	${}_\star d$	${}_\star d$	${}_\star l_j'$	${}_\star e$	${}_\star a_r^{u-1} d^{v+5}$		4	34	
10.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star l_j' a_r^{u-1} d^{v+6}$		3		41
11.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star l_j'$	${}_\star a_r^{u-1} d^{v+6}$		4		43
12.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_j$	${}_\star a_r^{u-1} d^{v+7}$		3		45
13.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_j a_r^{u-1} d^{v+7}$				

(5) The halting instruction l_h is simulated by agent B_3 with subset of programs:

$$\frac{P_3}{47: \langle e \leftrightarrow l_h \rangle,\ 48: \langle l_h \rightarrow C \rangle,\ 49: \langle C \leftrightarrow e \rangle.}$$

The agent consumes the object l_h and in the environment there is no other object l_m. This agent places one copy of the object C to the environment and stops working. In the next step the object C is consumed by the agent B_3. No agent can start its work and the computation halts. The execution of the halting instruction l_h stops all agents in colony Π:

	configuration of Π					labels of applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star l_h d^v$		4	47	
2.	${}_\star d$	${}_\star e$	${}_\star l_h$	${}_\star e$	${}_\star d^{v+1}$		3	48	
3.	${}_\star d$	${}_\star d$	${}_\star C$	${}_\star e$	${}_\star d^{v+1} d$		4	49	
4.	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star e$	${}_\star C d^{v+1}$		3		
5.	${}_\star d$	${}_\star d$	${}_\star e$	${}_\star e$	${}_\star C d^{v+2}$		4		
6.	${}_\star d$	${}_\star C$	${}_\star e$	${}_\star e$	${}_\star d^{v+3}$	- - - - -	- - - - -	- - - - -	- - - - -

The P colony Π correctly simulates the computation in the register machine M. The computation of Π starts with no object a_r placed in the environment in the same way as the computation in M starts with zeros in all registers. The computation of Π stops if the symbol l_h and consequently object C is placed inside the corresponding agent in the same way as M stops by executing the halting instruction labeled l_h. Consequently, $N(M) = N(\Pi)$ and because the number of agents equals four, the proof is complete. □

Theorem 2. $NRM_{pb} \subseteq NPCOL_{par}(1, 2, *)$.

Proof. Let us consider a partially blind register machine M with m registers. We construct a P colony $\Pi = (A, e, f, {}_\star v_E, B_1, B_2)$ simulating a computation of the register machine M with:

- $A = \{J, J', V, Q\} \cup \{l_i, l'_i, l''_i, L_i, L'_i, L''_i, E_i \mid l_i \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$,
- $f = a_1$,
- $B_i = ({}_\star e, P_i), i = 1, 2$.

The sets of programs are as follows:

(1) For initializing the simulation:

P_1 :	P_1 :	P_2 :
$1:\ \langle e \to J \rangle$,	$3:\ \langle J \to l_0 \rangle$,	$5:\ \langle e \leftrightarrow J \rangle$,
$2:\ \langle J \leftrightarrow e \rangle$,	$4:\ \langle Q \to Q \rangle$,	$6:\ \langle J \to J' \rangle$,
		$7:\ \langle J' \leftrightarrow e \rangle$.

At the beginning of the computation the first agent generates the object l_0 (the label of the starting instruction of M). It generates some copies of object J. The agent B_2 exchange them by J'.

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star e$	${}_\star e$		1	–
2.	${}_\star J$	${}_\star e$		**2** *or* 3	–
3.	${}_\star e$	${}_\star e$	${}_\star J$	1	5
4.	${}_\star J$	${}_\star J$		**2** *or* 3	6
5.	${}_\star l_0$	${}_\star J'$		8 *or* 24 *or* 34	7
6.	?	${}_\star e$	${}_\star J'$		

(2) For every ADD-instruction $l_1 : (ADD(r), l_2, l_3)$, P_1 and P_2 contain:

P_1 :	P_1 :	P_2 :
$8:\ \langle l_1 \to l'_1 \rangle$,	$14:\ \langle L_1 \leftrightarrow E_1 \rangle$,	$18:\ \langle e \leftrightarrow l'_1 \rangle$,
$9:\ \langle l'_1 \leftrightarrow J' \rangle$,	$15:\ \langle L_1 \to Q \rangle$,	$19:\ \langle l'_1 \to E_1 \rangle$,
$10:\ \langle l'_1 \to Q \rangle$,	$16:\ \langle E_1 \to l_2 \rangle$,	$20:\ \langle E_1 \leftrightarrow e \rangle$,
$11:\ \langle J' \to L''_1 \rangle$,	$17:\ \langle E_1 \to l_3 \rangle$,	$21:\ \langle e \leftrightarrow L_1 \rangle$,
$12:\ \langle L''_1 \to L'_1 \rangle$,		$22:\ \langle L_1 \to a_r \rangle$,
$13:\ \langle L'_1 \to L_1 \rangle$,		$23:\ \langle a_r \leftrightarrow e \rangle$.

When there is an object l_1 inside agent B_1, the agent rewrites it to a copy of l_1' and the agent sends it to the environment. The agent B_2 borrows E_1 from the environment and returns E_1' back.

The agent B_1 rewrites the object J' to some L_i. The first agent has to generate it in three steps to wait until the second agent generates the symbol E_i' and places it into the environment. If this L_i has the same index as E_i' placed in the environment, the computation can go to the next phase. If the indices of L_i and E_i are different, the agent B_1 generates Q and the computation never stops. If the computation gets over this checking step, B_1 generates object l_2 or object l_3.

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	$_\star l_1$	$_\star e$	$_\star J'$	8	–
2.	$_\star l_1'$	$_\star e$	$_\star J'$	**9** *or* 10	–
3.	$_\star J'$	$_\star e$	$_\star l_1'$	11	18
4.	$_\star L_1''$	$_\star l_1'$		12	19
5.	$_\star L_1'$	$_\star E_1$		13	20
6.	$_\star L_1$	$_\star e$	$_\star E_1$	**14** *or* 15	–
7.	$_\star E_1$	$_\star e$	$_\star L_1$	**16** *or* 17	21
8.	$_\star l_2$	$_\star L_1$		8 *or* 24 *or* 34	22
9.	?	$_\star a_r$		9 *or* 25 *or* 35	23
10.	?	$_\star e$	$_\star a_r$		

(3) For every SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ the following subsets of programs are in P_1 and P_2:

P_1 :	P_1 :	P_2 :
$24 : \langle l_1 \to l_1'' \rangle$,	$28 : \langle V \leftrightarrow l_1''' \rangle$,	$31 : \langle e \leftrightarrow l_1'' \rangle$,
$25 : \langle l_1'' \leftrightarrow a_r \rangle$,	$29 : \langle l_1''' \to l_2 \rangle$,	$32 : \langle l_1'' \to l_1''' \rangle$,
$26 : \langle l_1'' \to Q \rangle$,	$30 : \langle l_1''' \to l_3 \rangle$	$33 : \langle l_1''' \leftrightarrow e \rangle$,
$27 : \langle a_r \to V \rangle$.		

In the first step the agent checks if there is any copy of a_r in the environment (for zero in register r). Because of the nondeterminism of the computation in

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	$_\star l_1$	$_\star e$	$_\star a_r$	24	–
2.	$_\star l_1''$	$_\star e$	$_\star a_r$	**25** *or* 26	–
3.	$_\star a_r$	$_\star e$	$_\star l_1''$	27	31
4.	$_\star V$	$_\star l_1''$		–	32
5.	$_\star V$	$_\star l_1'''$		–	33
6.	$_\star V$	$_\star e$	$_\star l_1'''$	28	–
7.	$_\star l_1'''$	$_\star e$		**29** *or* 30	–
8.	$_\star l_2$	$_\star e$			

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	$_\star l_1$	$_\star e$		24	–
2.	$_\star l_1''$	$_\star e$		26	–
3.	$_\star Q$	$_\star e$		4	
4.	$_\star Q$	$_\star e$			

the positive case it can rewrite a_r to V, in the other case l_1'' is rewritten to Q and the computation will never halt. At the end of this simulation the agent B_1 generates one of the objects l_2, l_3.

(4) For the halting instruction l_h the following programs are in sets P_1 and P_2:

P_1 :	P_2 :	P_2 :
$34: \langle l_h \leftrightarrow J' \rangle,$	$39: \langle e \leftrightarrow l_h \rangle,$	$43: \langle L_h \leftrightarrow a_r \rangle, 1 < r \leq m$
$35: \langle J' \rightarrow L_h \rangle,$	$40: \langle l_h \rightarrow \overline{l_h} \rangle,$	$44: \langle a_r \leftrightarrow e \rangle.$
$36: \langle l_h \rightarrow Q \rangle,$	$41: \langle \overline{l_h} \leftrightarrow e \rangle,$	
$37: \langle L_h \rightarrow L_h \rangle,$	$42: \langle e \leftrightarrow L_h \rangle,$	
$38: \langle L_h \leftrightarrow \overline{l_h} \rangle,$		

By using these programs, the P colony finishes the computation in the same way as the partially blind register machine halts its computation. Programs with labels 43 and 44 in P_2 check value zero stored in all except the first register. If there is some copy of object a_r, programs 43 and 44 are applied in a cycle and the computation never ends. Some copies of object J' (for the the program with label 34) are present in the environment from the initialization of computation.

all counters $r, 1 < r \leq m$ store zero					
	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star l_h$	${}_\star e$	${}_\star J'$	**34** *or* 36	–
2.	${}_\star J'$	${}_\star e$	${}_\star l_h$	35	39
3.	${}_\star L_h$	${}_\star l_h$		37	40
4.	${}_\star L_h$	${}_\star \overline{l_h}$		37	41
5.	${}_\star L_H$	${}_\star e$	${}_\star \overline{l_h}$	38	–
6.	${}_\star \overline{l_h}$	${}_\star e$	${}_\star L_h$	–	42
7.	${}_\star \overline{l_h}$	${}_\star L_h$		–	–

content of some counter $r, 1 < r \leq m$ is different from zero					
	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star l_h$	${}_\star e$	${}_\star J' a_r$	**34** *or* 36	–
2.	${}_\star J'$	${}_\star e$	${}_\star l_h a_r$	35	39
3.	${}_\star L_h$	${}_\star l_h$	${}_\star a_r$	37	40
4.	${}_\star L_h$	${}_\star \overline{l_h}$	${}_\star a_r$	37	41
5.	${}_\star L_H$	${}_\star e$	${}_\star \overline{l_h} a_r$	38	–
6.	${}_\star \overline{l_h}$	${}_\star e$	${}_\star L_h a_r$	–	42
7.	${}_\star \overline{l_h}$	${}_\star L_h$	${}_\star a_r$	–	43
8.	${}_\star \overline{l_h}$	${}_\star a_r$	${}_\star L_h$	–	44
9.	${}_\star \overline{l_h}$	${}_\star L_h$	${}_\star a_r$	–	43

The P colony Π correctly simulates any computation of the partially blind register machine M. □

4 On the Computational Power of Restricted P Colonies Without Checking

For restricted P colonies the following results are known:

– $NPCOL_{par}KR(2, *, 5) = NRE$ in [2,8],
– $NPCOL_{par}R(2, *, 5) = NPCOL_{par}KR(2, 1, *) = NRE$ in [4].

The next theorem determines the computational power of restricted P colonies working without checking rules.

Theorem 3. *$NPCOL_{par}R(2, 2, *) = NRE$.*

Proof. Let us consider a register machine M with m registers. We construct a P colony $\Pi = (A, e, f, {}_\star v_E, B_1, B_2)$ simulating the computations of register machine M with:

- $A \ = \{G\} \cup \{l_i, l_i', l_i'', l_i''', l_i'''', \overline{l_i}, \overline{\overline{l_i}}, \underline{l_i}, \underline{\underline{l_i}}, L_i, L_i', L_i'', F_i \mid l_i \in H\} \cup$
  $\cup \ \{a_r \mid 1 \leq r \leq m\}$,
- $f \ = a_1$,
- $B_j = ({}_\star ee, P_j), j = 1, 2$.

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

(1) For initializing the simulation there is one program in P_1:

P_1
$1 : \langle e \to l_0; e \leftrightarrow e \rangle$

The initial configuration of Π is $({}_\star ee, {}_\star ee, {}_\star \varepsilon)$. After the first step of the computation (only program 1 is applicable) the system enters configuration $({}_\star l_0 e, {}_\star ee, {}_\star \varepsilon)$.

(2) For every ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ we add to P_1 the programs:

P_1	
$2 : \langle e \to a_r; l_1 \leftrightarrow e \rangle$,	$3 : \langle e \to G; a_r \leftrightarrow l_1 \rangle$,
$4 : \langle l_1 \to l_2; G \leftrightarrow e \rangle$,	$5 : \langle l_1 \to l_3; G \leftrightarrow e \rangle$.

When there is an object l_1 inside the agent, it generates one copy of a_r, puts it into the environment and generates the label of the next instruction (it nondeterministically chooses one of the last two programs 4 and 5).

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star l_1 e$	${}_\star ee$	${}_\star a_r^x$	2	–
2.	${}_\star a_r e$	${}_\star ee$	${}_\star l_1 a_r^x$	3	–
3.	${}_\star G l_1$	${}_\star ee$	${}_\star a_r^{x+1}$	4 *or* 5	–
4.	${}_\star l_2 e$	${}_\star ee$	${}_\star a_r^{x+1} G$		

(3) For every SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the next programs are added to sets P_1 and P_2:

P_1	P_1	P_2
$6 : \langle l_1 \to l_1'; e \leftrightarrow e \rangle$,	$12 : \left\langle \overline{\overline{l_1}} \to \underline{l_2}; e \leftrightarrow L_1'' \right\rangle$,	$18 : \langle e \to L_1; e \leftrightarrow l_1' \rangle$,
$7 : \langle e \to l_1''; l_1' \leftrightarrow e \rangle$,	$13 : \left\langle \overline{\overline{l_1}} \to \underline{l_3}; e \leftrightarrow L_1 \right\rangle$,	$19 : \langle l_1' \to L_1'; L_1 \leftrightarrow l_1'' \rangle$,
$8 : \langle e \to l_1'''; l_1'' \leftrightarrow e \rangle$,	$14 : \left\langle L_1'' \to l_2; \underline{l_2} \leftrightarrow e \right\rangle$,	$20 : \langle l_1'' \to L_1''; L_1' \leftrightarrow a_r \rangle$,
$9 : \langle l_1''' \to l_1''''; e \leftrightarrow e \rangle$,	$15 : \left\langle L_1 \to F_3; \underline{l_3} \leftrightarrow e \right\rangle$,	$21 : \langle a_r \to e; L_1'' \leftrightarrow L_1 \rangle$,
$10 : \left\langle l_1'''' \to \overline{l_1}; e \leftrightarrow e \right\rangle$,	$16 : \left\langle e \to \underline{\underline{l_3}}; F_3 \leftrightarrow \underline{l_3} \right\rangle$,	$22 : \langle L_1 \to e; e \leftrightarrow e \rangle$,
$11 : \left\langle \overline{l_1} \to \overline{\overline{l_1}}; e \leftrightarrow e \right\rangle$,	$17 : \left\langle \underline{l_3} \to l_3; \underline{\underline{l_3}} \leftrightarrow e \right\rangle$,	$23 : \langle l_1'' \to e; L_1' \leftrightarrow F_3 \rangle$,
		$24 : \langle F_3 \to e; e \leftrightarrow e \rangle$.

At the first phase of the simulation of the SUB instruction the first agent generates object l_1', which is consumed by the second agent. The agent B_2 generates symbol L_1 and tries to consume one copy of symbol a_r. If there is any a_r, the agent sends to the environment object L_1'' and consumes L_1. After this step the first agent consumes L_1'' or L_1 and rewrites it to l_2 or l_3. The objects $\underline{x}$, $\overline{x}$ and $\overline{\overline{x}}$ are used for a synchronization of the computation in both agents and for storing information about the state of the computation.

Instruction $l_1 : (SUB(r), l_2, l_3)$ is simulated by the following sequence of steps.

If the register r stores a nonzero value:

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star l_1 e$	${}_\star ee$	${}_\star a_r^x$	6	–
2.	${}_\star l_1' e$	${}_\star ee$	${}_\star a_r^x$	7	–
3.	${}_\star l_1'' e$	${}_\star ee$	${}_\star l_1' a_r^x$	8	18
4.	${}_\star l_1''' e$	${}_\star L_1 l_1'$	${}_\star l_1'' a_r^x$	9	19
5.	${}_\star l_1'''' e$	${}_\star L_1' l_1''$	${}_\star L_1 a_r^x$	10	20
6.	${}_\star \overline{l_1} e$	${}_\star L_1'' a_r$	${}_\star L_1 L_1' a_r^{x-1}$	11	21
7.	${}_\star \overline{\overline{l_1}} e$	${}_\star e L_1$	${}_\star L_1'' a_r^{x-1}$	12	22
8.	${}_\star \underline{l_2} L_1''$	${}_\star ee$	${}_\star a_r^{x-1}$	14	–
9.	${}_\star l_2 e$	${}_\star ee$	${}_\star a_r^{x-1} \underline{l_2}$		

If the register r stores value zero :

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
1.	${}_\star l_1 e$	${}_\star ee$		6	–
2.	${}_\star l_1' e$	${}_\star ee$		7	–
3.	${}_\star l_1'' e$	${}_\star ee$	${}_\star l_1'$	8	18
4.	${}_\star l_1''' e$	${}_\star L_1 l_1'$	${}_\star l_1''$	9	19
5.	${}_\star l_1'''' e$	${}_\star L_1' l_1''$	${}_\star L_1$	10	

	configuration of Π			labels of applicable programs	
	B_1	B_2	Env	P_1	P_2
6.	$_\star\overline{l_1}e$	$_\star L_1'l_1''$	$_\star L_1$	11	
7.	$_\star\overline{\overline{\overline{l_1}}}e$	$_\star L_1'l_1''$	$_\star L_1$	13	
8.	$_\star\underline{l_3}L_1$	$_\star L_1'l_1''$		15	–
9.	$_\star F_3e$	$_\star L_1'l_1''$	$_\star\underline{l_3}$	16	–
10.	$_\star\underline{l_3l_3}$	$_\star L_1'l_1''$	$_\star F_3$	17	23
11.	$_\star\overline{l_3}e$	$_\star F_3e$	$_\star\underline{\underline{l_3}}L_1'$	2 or 6 or none	24
12.	?	$_\star ee$	$_\star\underline{\underline{l_3}}L_1'$		

(4) For halting instruction l_h no program is added to the sets P_1 and P_2.

The P colony Π correctly simulates all computations of the register machine M and the number contained in the first register of M corresponds to the number of copies of the object a_1 present in the environment of Π. □

5 Conclusions

We have shown that the P colonies with capacity $c = 2$ and without checking programs, with height at most 2, are computationally complete. In Section 3 we have shown that the P colonies with capacity $c = 1$ and with checking/evolution programs and 4 agents are computationally complete.

We have verified also that partially blind register machines can be simulated by P colonies with capacity $c = 1$ without checking programs with two agents. The generative power of $NPCOL_{par}K(1, n, *)$ for $n = 2, 3$ remains open.

In Section 4 we have studied P colonies with capacity $c = 2$ without checking programs. Two agents guarantee the computational completeness in this case.

For more information on membrane computing, see [11], for more on computational machines and colonies in particular, see [9] and [6,7,8], respectively. Activities carried out in the field of membrane computing are currently numerous and they are available also at [12].

Acknowledgements

This work has been supported by the Grant Agency of Czech Republic grants No. 201/06/0567 and by IGS SU 32/2007.

References

1. Ciencialová, L., Cienciala, L.: Variations on the theme: P colonies. In: Kolář, D., Meduna, A. (eds.) Proceedings of the 1st International workshop WFM 2006, Ostrava, pp. 27–34 (2006)

2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, G., Vaszil, G.: Cells in environment: P colonies. Journal of Multiple-valued Logic and Soft Computing 12(3-4), 201–215 (2006)
3. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: P colonies with a bounded number of cells and programs. In: Hoogeboom, H.J., Păun, G., Rozenberg, G. (eds.) Pre-Proceedings of the 7th Workshop on Membrane Computing, Leiden, the Netherlands, pp. 311–322 (2006)
4. Freund, R., Oswald, M.: P colonies working in the maximally parallel and in the sequential mode. In: Ciobanu, G., Păun, G. (eds.) Pre-Proceedings of the 1st International Workshop on Theory and Application of P Systems, Timisoara, Romania, pp. 49–56 (2005)
5. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science 7(1), 311–324 (1978)
6. Kelemen, J., Kelemenová, A.: A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems 23, 621–633 (1992)
7. Kelemen, J., Kelemenová, A.: On P colonies, a biochemically inspired model of computation. In: Proc. of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, pp. 40–56 (2005)
8. Kelemen, J., Kelemenová, A., Păun, G.: Preview of P colonies: A biochemically inspired computing model. In: Bedau, M., et al. (eds.) ALIFE IX. Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, Boston, Mass., pp. 82–86 (2004)
9. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ (1967)
10. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61, 108–143 (2000)
11. Păun, G.: Membrane Computing: An Introduction. Springer, Berlin (2002)
12. P systems web page. `http://psystems.disco.unimib.it`