On the Number of Agents in P Colonies

Luděk Cienciala[1], Lucie Ciencialová[1], 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

Summary. We continue the investigation of P colonies introduced in [7], a class of abstract computing devices composed of independent agents, acting and evolving in a shared environment.

We decrease the number of agents needed to computational completeness of P colonies with one and 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 paper [7] as formal models of a computing device inspired by membrane systems and by grammar systems called colonies. This model is inspired by 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 represented 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 all the objects inside of the agent are being evolved (by an evolution rule) or transported (by a communication rule). The third type of the rules used is a checking rule. This type of the rules sets the priority between two rules.

The computation starts in the initial configuration specified in the definition. Using their programs the agents change themselves and by the environment they can affect the behavior of the other agents. 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 the number of some specific objects present at the environment at the end of the computation.

There are several different ways how to define the beginning of the computation. (1) At the beginning of computation the environment and all agents 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.
(3) The initial state of the environment is nonempty - the environment contains initial "parameters" for future computation, the agents start with e-s.

In [4, 6, 7] 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 talk about restricted P colonies. If we allow also another combination of the types of the rules, we obtain non-restricted P colonies. The restricted P colonies with the checking rules are computationally complete [3, 4]. Activities carried out in the field of membrane computing are currently numerous and they are available also at [11].

In the present paper we start with definitions in Section 2.

In Section 3 we will deal with P colonies with one object inside each agent. In recent paper [1] there was shown, that at most seven programs for each agent as well as five agents guarantees 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. If no checking rules are used in the restricted P colonies then we need two agents to prove the universal computational power of those P colonies.

2 Definitions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory.

We use NRE to denote the family of the recursively enumerable sets of natural numbers. Let Σ be the 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 multisets with the set of objects V is denoted by V°. The set V' is called the support of M and denoted by $supp(M)$ if for all $x \in V'$ $f(x) \neq 0$ holds. The cardinality of M, denoted by

$|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V' = \{a_1, \ldots a_n\}$ 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 the same M, and ε represents the empty multiset.

2.1 P colonies

We briefly recall the notion of P colonies. A P colony consists of agents and environment. Both the agents and the environment contain objects. With every agent the set of program is associated. There are two types of rules in the programs. The first type, called the evolution, is 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 a communication, is in 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 places, so, after execution of the rule d appears inside the agent and c is placed outside of the agent.

In [6] the ability of agents is extended by checking rules. These rules give to the agents an opportunity to opt between two possibilities. They have form r_1/r_2. If the checking rule is performed, the rule r_1 has higher priority to be executed as the rule r_2 has. 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 first rule cannot be applied, the agent uses the second one.

Definition 1. *The P colony of the capacity k is a construct*
$$\Pi = (A, e, f, V_E, B_1, \ldots, B_n), \text{ where}$$

- *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,*
- *V_E is a multiset over $A - \{e\}$,*
- *$B_i,\ 1 \leq i \leq n$, are agents, each agent is a construct $B_i = (O_i, P_i)$, where*
  - *O_i is a multiset over A, it determines the initial state (content) of the agent, $|O_i| = k$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite multiset of programs, where each program contains exactly k 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 an evolution or a communication rules.*

An initial configuration of the P colony is an $(n+1)$-tuple of strings of objects present in the P colony at the beginning of the computation, it is given by O_i for $1 \leq i \leq n$ and by V_E. Formally, the configuration of P colony Π is given by $(w_1, \ldots, w_n, w_E)$, where $|w_i| = k,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent and $w_E \in (A - \{e\})^*$ represents all the objects in the environment different from the object e.

In the paper parallel model of P colonies will be studied. 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 possible number of agents are active.

Let the programs of each P_i be labeled 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 express derivation step formally we introduce following four functions for the agent using the rule r of program $p \in P$ with objects w in the environment:

For rule r being $a \to b, c \leftrightarrow d$ and $c \leftrightarrow d/c' \leftrightarrow d'$, respectively, and for multiset $w \in V^\circ$ we define:

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

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

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

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

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

A transition from a configuration to another is denoted as

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

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

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

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

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. It is 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

$$N(\Pi) = \left\{ |v_E|_f \mid (w_1, \ldots, w_n, V_E) \Rightarrow^* (v_1, \ldots, v_n, v_E) \in H \right\},$$

where $(w_1, \ldots, w_n, V_E)$ is the initial configuration, $(v_1, \ldots, v_n, v_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

Given a P colony $\Pi = (A,\ e,\ f,\ V_E,\ B_1,\ \ldots,\ B_n)$ 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 agent.

Let us use the following notations:
$NPCOL_{par}(k, n, h)$ for the family of all sets of numbers computed by P colonies working in parallel, using no checking rules and with:

- the capacity at most k,
- the degree at most n and
- the height at most h.

If we allow checking rules the family of all sets of numbers computed by P colonies is denoted by $NPCOL_{par}K$. If the P colonies are restricted, we replace the denotation to $NPCOL_{par}R$ and $NPCOL_{par}KR$, respectively.

2.2 Register machines

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

Definition 2. *[8] 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 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 instruction 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 the value different from zero, then subtract 1 from its content and go to instruction l_2, otherwise proceed to instruction l_3.

$l_h : HALT$ 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: 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.[8]) that in this way we can compute all sets of numbers which are Turing computable.

Moreover, we call a register machine partially blind [5], if we interpret a subtract instruction in the following way: $l_1 : (SUB(r); l_2; l_3)$ - if in register r there is value different from zero, then subtract one from its contents and go to instruction l_2 or to instruction l_3; if in register r there is stored zero 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 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 machine accepts 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 of P colonies. This gives that every program 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] following results was proved:

– $NPCOL_{par}K(1, *, 7) = NRE$.

– $NPCOL_{par}K(1, 5, *) = NRE$

The number of agents in the second result can be decreased:

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

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

Let us consider an m-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 environment. We construct the P colony $\Pi = (A, e, f, \emptyset, B_1, \ldots, B_4)$ with:

– alphabet $A = \{l, l' | l \in H\} \cup$
$\cup \{E_i, E'_i, F_i, F'_i, F''_i \mid \textit{for each } l_i \in H\} \cup$
$\cup \{a_i | 1 \leq i \leq m\} \cup \{e, d, m, C\}$,
– $f = a_1$,
– $B_i = (e, P_i),\ 1 \leq i \leq 4$.

(1) To initialize simulation of computation of M we take agent $B_1 = (e, P_1)$ with a set of programs:

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

(2) We need one more agent to generate some special object d. In every pair of steps the agent B_2 places one copy of d to the environment.

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

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

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	e	e	e	e		1	3		
2.	l_0	d	e	e			4		
3.	l_0	e	e	e	d	2	3		
4.	d	d	e	e	l_0				

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

P_3	P_3	P_4	P_4
$5 : \langle e \leftrightarrow l_1 \rangle ,$	$11 : \langle E'_1 \rightarrow l'_2 \rangle ,$	$15 : \langle e \leftrightarrow E_1 \rangle ,$	$21 : \langle e \leftrightarrow l'_2 \rangle ,$
$6 : \langle l_1 \rightarrow E_1 \rangle ,$	$12 : \langle E'_1 \rightarrow l'_3 \rangle ,$	$16 : \langle E_1 \rightarrow E'_1 \rangle ,$	$22 : \langle e \leftrightarrow l'_3 \rangle ,$
$7 : \langle E_1 \leftrightarrow d \rangle ,$	$13 : \langle l'_2 \leftrightarrow e \rangle ,$	$17 : \langle E'_1 \leftrightarrow e \rangle ,$	$23 : \langle l'_2 \rightarrow l_2 \rangle ,$
$8 : \langle d \rightarrow L_1 \rangle ,$	$14 : \langle l'_3 \leftrightarrow e \rangle ,$	$18 : \langle e \leftrightarrow L_1 \rangle ,$	$24 : \langle l'_3 \rightarrow l_3 \rangle ,$
$9 : \langle L_1 \leftrightarrow E'_1 / L_1 \rightarrow m \rangle ,$		$19 : \langle L_1 \leftarrow a_r \rangle ,$	$25 : \langle l_2 \leftrightarrow e \rangle ,$
$10 : \langle m \rightarrow d \rangle ,$		$20 : \langle a_r \leftrightarrow e \rangle ,$	$26 : \langle l_3 \leftrightarrow e \rangle ;$

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 gives a little altered (to E'_1) back. 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, B_3 generates the helpful object l'_2 or l'_3 and places it to the environment. The agent B_4 exchanges it for "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								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	d	d	e	e	$l_i a_r^u d^v$		4	5	
2.	d	e	l_i	e	$a_r^u d^{v+1}$		3	6	
3.	d	d	E_i	e	$a_r^u d^{v+1} d$		4	7	
4.	d	e	d	e	$E_i a_r^u d^{v+1}$		3	8	15

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

(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_4 starts 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' - there is some a_r; F_1'' - 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. When the value in counter r is zero:

	configuration of Π					applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	d	d	e	e	$l_i d^v$		4	27	
2.	d	e	l_i	e	d^{v+1}		3	28	
3.	d	d	F_i	e	$d^{v+1}d$		4	29	
4.	d	e	d	e	$F_i d^{v+1}$	36	3		
5.	F_i	d	d	e	d^{v+2}	37	4		
6.	F_i'	e	d	e	d^{v+3}	38	3		
7.	F_i''	d	d	e	d^{v+3}	40	4		
8.	d	e	d	e	$F_i'' d^{v+3}$		3	32	
9.	d	d	F_i''	e	d^{v+4}		4	33	
10.	d	e	l_k'	e	d^{v+5}		3	35	
11.	d	d	e	e	$l_k' d^{v+5}$		4		42
12.	d	e	e	l_k'	d^{v+6}		3		44
13.	d	d	e	l_k	d^{v+6}		4		46
14.	d	e	e	e	$l_k d^{v+7}$				

When register r stores value different from zero:

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

(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 computation halts. The execution of halting instruction l_h stops all agents in P colony Π:

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

P colony Π correctly simulates 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 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 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 $\Pi = (A, e, f, V_E, B_1, B_2)$ simulating a computation of the register machine M with:

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

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 starting instruction of M). It generates some copies of object J. The agent B_2 exchange them by J'.

	configuration of Π			P_1	P_2
	B_1	B_2	Env		
1.	e	e		1	–
2.	J	e		**2** *or* 3	–
3.	e	e	J	1	5
4.	J	J		**2** *or* 3	6
5.	l_0	J'		8 *or* 24 *or* 34	7
6.	?	e	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 object l_1 inside agent B_1, the agent rewrites it to one 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 till the second agent generates the symbol E_i' and places it to 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 l_3.

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

(3) For every SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ there are subsets of programs 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 : $\langle l_1'' \leftrightarrow e \rangle$,
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). In the positive case it rewrites 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 object l_2 or l_3.

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

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

(4) For halting instruction l_h there are programs 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 one registers.

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

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

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

4 On computational power of restricted P colonies without checking

For restricted P colonies Following results are known from the literature:

- $NPCOL_{par}KR(2,*,5) = NRE$ in [2, 7],
- $NPCOL_{par}R(2,*,5) = NPCOL_{par}KR(2,1,*) = NRE$ in [4].

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

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

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

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 instruction labeled l_0 and it generates the label of the next instruction. The sets of programs are as follows:

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

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

The initial configuration of Π is (ee, ee, ε). After the first step of computation (only the program 1 is applicable) the system enters configuration $(l_0 e, ee, \varepsilon)$.
(2) For every ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ we add to P_1 the programs:

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

When there is object l_1 inside the agent, it generates one copy of a_r, puts it to 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				
	B_1	B_2	Env	P_1	P_2
1.	$l_1 e$	ee	a_r^x	2	–
2.	$a_r e$	ee	$l_1 a_r^x$	3	–
3.	$G l_1$	ee	a_r^{x+1}	**4** *or* 5	–
4.	$l_2 e$	ee	$a_r^{x+1} G$		

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

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 value zero :

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

If the register r stores nonzero value:

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	$l_1 e$	ee		6	–
2.	$l_1' e$	ee		7	–
3.	$l_1'' e$	ee	l_1'	8	18
4.	$l_1''' e$	$L_1 l_1'$	l_1''	9	19
5.	$l_1'''' e$	$L_1' l_1''$	L_1	10	
6.	$\overline{l_1} e$	$L_1' l_1''$	L_1	11	
7.	$\overline{\overline{l_1}} e$	$L_1' l_1''$	L_1	13	
8.	$\underline{l_3} L_1$	$L_1' l_1''$		15	–
9.	$F_3 e$	$L_1' l_1''$	$\underline{l_3}$	16	–
10.	$\underline{l_3 l_3}$	$L_1' l_1''$	F_3	17	23
11.	$\overline{\overline{l_3}} e$	$F_3 e$	$\underline{\underline{l_3}} L_1'$	2 or 6 or none	24
12.	??	ee	$\underline{\underline{l_3}} L_1'$		

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

P colony Π correctly simulates all computations 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 present in the environment of Π. □

5 Conclusions

We have shown that the P colonies with capacity $k = 2$ and without checking programs with height at most 2 are computationally complete. In Section 3 we have shown that the P colonies with capacity $k = 1$ 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 $k = 1$ 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 $k = 2$ without checking programs. Two agents guarantee the computational completeness in this case.

Remark 1. 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*, Proceedings of the 1^{st} International workshop WFM'06 (Kolář, D., Meduna, A., eds.), Ostrava, 2006, pp. 27-34.
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, G.: *Cells in environment: P colonies*, Multiple-valued Logic and Soft Computing, 12, 3-4, 2006, pp. 201-215.
3. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: *P Colonies with a bounded number of cells and programs.* Pre-Proceedings of the 7^{th} Workshop on Membrane Computing (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311-322.
4. Freund, R., Oswald, M.: *P colonies working in the maximally parallel and in the sequential mode.* Pre-Proceedings of the 1^{st} International Workshop on Theory and Application of P Systems (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, pp. 49-56.
5. Greibach, S. A.: *Remarks on blind and partially blind one-way multicounter machines.* Theoretical Computer Science, 7(1), 1978, pp. 311-324.
6. Kelemen, J., Kelemenová, A.: *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, pp. 40-56.
7. Kelemen, J., Kelemenová, A., Păun, Gh.: *Preview of P colonies: A biochemically inspired computing model.* Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX (M. Bedau at al., eds.) Boston, Mass., 2004, pp. 82–86.
8. Minsky, M. L.: *Computation: Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967.
9. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108-143.
10. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
11. P systems web page: http://psystems.disco.unimib.it