Notes on Restricted P Colonies

Lucie Ciencialová[1] and Luděk Cienciala[1]

Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz

Abstract. 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 determine the generative power of P colonies with one resp. two objects inside each agent owing some special restrictions to the number of agents and type of programs.

Keywords: P colony, membrane systems, generative power.

1 Introduction

P colonies were introduced in the paper [6] as formal models of a computing device inspired by membrane systems and formal grammars 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 and rules for processing these objects. The number of objects inside the agent is the same for each of them. The environment contains several copies of the basic environmental object denoted by e. The number of the copies of e is unlimited.

With each agent a set of programs is associated. The program determines the activity of the agent. Each program consists of the same number of rules as the number of the objects inside the agent. 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 is checking rule. This type of the rules sets the priority between two communication rules.

The computation starts in the initial configuration when all agents and environment contain copies of the environmental object e. By 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 nondeterministically chooses one of its applicable programs and executes it. The computation halts when no agent can apply any of its programs. The result of the computation is the number of some specific objects present at the environment at the end of the computation.

In [4, 6, 7] the authors study P colonies with two objects inside agents. In this case programs consist of two rules. If the former of these rules is evolution and

the latter is 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].

In the present paper we show that restricted P colonies without checking rules and two agents can also generate any recursively enumerable set of natural numbers working in maximally parallel way. We also show that computational power of P colonies with one object inside the agent can simulate the partially blind register machine.

2 Definitions

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

We use NRE to denote the family of 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 $f : V \rightarrow N$; 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 $x \in V'$ $f(x) \neq 0$. 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 $V = \{a_i, \ldots a_n\}$ 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 M, and ε represents the empty multiset.

2.1 P Colony

We briefly recall the notion of P colonies.

The P colony consists of agents and environment. Both agents and environment contain objects. With every agent the set of program is associated. There are two kinds of rules in programs. The first type called evolution is in the form $a \rightarrow b$. It means that object a inside of agent is rewritten (evolved) to object b. The second type of rules can be called communication and they are in the form $c \leftrightarrow d$. When this rule is performed, the object c inside and the object d outside of the agent change their places, so d is now inside and c is outside of the agent.

In [6] the ability of agents is extended by checking programs. They give to the agents the opportunity to opt between two possibilities. These rules have form $c \leftrightarrow d/c' \leftrightarrow d'$. If the checking rule is performed, the communication rule $c \leftrightarrow d$ has higher priority to be executed as the rule $c' \leftrightarrow d'$. It means that the agent checks the possibility of using the rule $c \leftrightarrow d$ (it tries to find object c inside of itself and the object d in the environment). If this rule can be executed, the agent must use it. If the first rule cannot be applied, the agent uses the second one $c' \leftrightarrow d'$.

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

- A *is an alphabet of the colony, its elements are called objects,*
- $e \in A$ *is the basic object of the colony,*
- $f \in A$ *is the final object of the colony,*
- 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 set of programs, where each program contains exactly* k *rules, which are in one of the following forms:*
    - $a \rightarrow b$, *these rules are called evolution rules,*
    - $c \leftrightarrow d$, *these rules are called communication rules,*
    - $c \leftrightarrow d/c' \leftrightarrow d'$, *which are called checking rules.*

An initial configuration of the P colony is $(n+1)-$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 V_E. Formally, the configuration of P colony Π is $(w_1, \ldots, w_n, w_E)$, 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 e.

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

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 $i \neq j$, $1 \leq i, j \leq n$.

For a rule r and a multiset $w \in V^\circ$ we can define:

$$r = (a \rightarrow b) \Rightarrow \begin{cases} left(r, w) = a \\ right(r, w) = b \\ export(r, w) = \varepsilon \\ import(r, w) = \varepsilon \end{cases} \qquad r = (c \leftrightarrow d) \Rightarrow \begin{cases} left(r, w) = \varepsilon \\ right(r, w) = \varepsilon \\ export(r, w) = c \\ import(r, w) = d \end{cases}$$

$$r = (c \leftrightarrow d/c' \leftrightarrow d') \Rightarrow \begin{cases} left(r, w) = right(r, w) = \varepsilon \\ \left.\begin{array}{l} export(r, w) = c \\ import(r, w) = d \end{array}\right\} \text{if } d \in w \\ \left.\begin{array}{l} export(r, w) = c' \\ import(r, w) = d' \end{array}\right\} \text{if } d \notin w \text{ and } d' \in w \end{cases}$$

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. With a halting computation we can associate a result of the computation. It is the number of copies of the special symbol f present in the environment. The set of numbers computed by a P colony Π is defined as

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

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 $\Rightarrow$.

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

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

$NPCOL_{par}(k, n, h)$ - the family of all sets of numbers $N(\Pi)$ computed by P colonies working in parallel way with: - the capacity k,
- the degree at most n and
- the height at most h
- without using checking rules.

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 too, we change notation to $NPCOL_{par}R$ or $NPCOL_{par}KR$.

2.2 Register Machines

In this work 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 need 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 initial/start label, l_h is the final label,*
- *P is a finite set of instructions injectively labeled with the elements from the given set H.*

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

$l_1 : (ADD(r), l_2, l_3)$ Add 1 to the contents of the register r and proceed to the instruction (labeled 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 $l_1 : (A(r), l_2, l_3)$ and in each SUB-instruction $l_1 : (S(r), l_2, l_3)$ 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.) It is known (see e.g.[7]) 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: $l_1 : (S(r); l_2; l_3)$ - 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.

3 On computational power of restricted P colonies Without Checking

The next results were proved to be true:

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

3.1 P colonies With Two Objects

Theorem 1. $NPCOL_{par}R(2,2,*) = NRE$.

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 a computation 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 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:

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

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 programs to P_1:

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

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 $l_1 : (SUB(r), l_2, l_3)$ there are subsets of programs in P_1 and P_2:

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

At the first phase of 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. The first agent after

this step consumes L_1'' or L_1 and rewrites it to l_2 or l_3. The objects $\underline{l_i}$, $\overline{l_i}$ and $\overline{\overline{l_i}}$ are for a synchronization of the computation in both agents and for storing informations about the state of the computation.

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

If the register r is empty:

	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.	$\overline{\overline{l_1}} e$	eL_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 is not empty:

	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.	$\underline{\underline{l_3 l_3}}$	$L_1' l_1''$	F_3	17	23
11.	$l_3 e$	$F_3 e$	$\underline{\underline{l_3}} L_1'$	2 or 6 or none	24
12.	??	ee	$\underline{\underline{l_3}} L_1'$		

(4) For the halting instruction l_h there is no program neither in the set P_1 either in the set P_2.

It is easy to see that the P colony Π correctly simulates any computation of the register machine M and the number contained on the first register of M corresponds to the number of copies of the object a_1 presented in the environment of Π. The rest of the proof follows by the Church-Turing thesis and by the computational universality of the register machine. □

3.2 P Colonies With One Object

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

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 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 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				
	B_1	B_2	Env	P_1	P_2
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 : $\langle l_1 \rightarrow l_1' \rangle$,	14 : $\langle L_1 \leftrightarrow E_1 \rangle$,	18 : $\langle e \leftrightarrow l_1' \rangle$,
9 : $\langle l_1' \leftrightarrow J' \rangle$,	15 : $\langle L_1 \rightarrow Q \rangle$,	19 : $\langle l_1' \rightarrow E_1 \rangle$,
10 : $\langle l_1' \rightarrow Q \rangle$,	16 : $\langle E_1 \rightarrow l_2 \rangle$,	20 : $\langle E_1 \leftrightarrow e \rangle$,
11 : $\langle J' \rightarrow L_1'' \rangle$,	17 : $\langle E_1 \rightarrow l_3 \rangle$,	21 : $\langle e \leftrightarrow L_1 \rangle$,
12 : $\langle L_1'' \rightarrow L_1' \rangle$,		22 : $\langle L_1 \rightarrow a_r \rangle$,
13 : $\langle L_1' \rightarrow L_1 \rangle$,		23 : $\langle a_r \leftrightarrow e \rangle$

When there is object l_1 inside the 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 gives a little altered (to E_1') back.

The agent B_1 rewrites the object J' to some L_i. We have 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 computation will never stop. 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 : $\langle l_1 \rightarrow l_1'' \rangle$,	28 : $\langle V \leftrightarrow l_1''' \rangle$,	31 : $\langle l_1'' \leftrightarrow e \rangle$,
25 : $\langle l_1'' \leftrightarrow a_r \rangle$,	29 : $\langle l_1''' \rightarrow l_2 \rangle$,	32 : $\langle l_1'' \rightarrow l_1''' \rangle$,
26 : $\langle l_1'' \rightarrow Q \rangle$,	30 : $\langle l_1''' \rightarrow l_3 \rangle$,	33 : $\langle l_1''' \leftrightarrow e \rangle$
27 : $\langle a_r \rightarrow V \rangle$,		

In the first step the first agent checks if there is any copy of a_r in the environment (whether register r is not empty). 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 the 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 the halting instruction l_h there are programs in the 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 this program, the P colony finishes computation as well as the partially blind register machine halts its computation. The last two programs in the sequence of P_2 are to control if the registers except the first one are empty.

if all counters $_{r,1<r\leq m}$ are empty					
	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		–	–

if some counter $_{r,1<r\leq m}$ is nonempty					
	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

It is easy to see that the P colony Π correctly simulates any computation of the partially blind register machine M. □

4 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 the next part of this study we have verified that P colonies with one object inside the agent and without checking programs can simulate partially blind register machine.

Activities carried out in the field of membrane computing are currently numerous and available at [11].

This work has been supported by the Grant Agency of Czech Republic grants No. 201/06/0567 "Bioinformatika a biovýpočty: souvislosti, modely, aplikace" and by IGS SU č.32/2007.

References

1. Ciencialová, L., Cienciala. L.: *Variations on the theme: P Colonies*, Proceedings of 1^st International workshop WFM06 (Kol, D., Meduna, A., eds.),Ostrava, 2006, pp. 27-34.
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenova, A., Paun, 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 7th Workshop on Membrane Computing (Hendrik Jan Hoogeboom, H. J., Păun, Gh., Rozenberg, G. 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 1st 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 6th 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, Masss, 2004, pp. 82–86.
8. Minsky, M. L.: *Computation Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967
9. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108-143 and TUCS Research Report No 208, Turku, 1998.
10. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
11. P systems web page: http://psystems.disco.unimib.it