P Colonies of Capacity One and Modularity

Luděk Cienciala, Lucie Ciencialová, and Miroslav Langer

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

Summary. We continue the investigation of P colonies introduced in [8], a class of abstract computing devices composed of independent agents, acting and evolving in a shared environment. The first part is devoted to the P colonies of the capacity one. We present improved allready presented results concerning the computional power of the P colonies. The second part is devoted to the modularity of the P colonies. We deal with dividing the agents into modules.

1 Introduction

P colonies were introduced in the paper [7] 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 or cells. Each agent is represented by a collection of objects embedded in a membrane.

The number of objects inside each agent is the same and constant during computation.

The environment contains several copies of the basic environmental object denoted by e. The number of the copies of e in the environment is sufficient.

With each agent a set of programs is associated. The program, which determines the activity of the agent, is very simple and depends on content of the agent and on multiset of objects placed in the environment. Agent can change content of the environment by programs and through the environment it can affect the behavior of other agents.

This influence between agents is a key factor in functioning of the P colony. In each moment each object inside the agent is affected by executing the program.

For more information about P systems see [12] or [13].

2 Definitions

Throughout the paper we assume that the reader is 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 is 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 $|M| = \sum_{a \in V} f(a)$. Each 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', where $|w|_{a_i} = f(a_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters represent the same multiset M. The ε represents the empty multiset.

2.1 P colonies

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

In [7] the set of programs was extended by the checking rules. These rules give an opportunity to the agents to opt between two possibilities. The rules are in the form r_1/r_2. If the checking rule is performed, then the rule r_1 has higher priority to be executed over the rule r_2. It means that the agent checks whether the rule r_1 is applicable. If the rule can be executed, then the agent is compulsory to use it. If the rule r_1 cannot be applied, then the agent uses the rule r_2.

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,* $|O_i| = k$,
- $P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ *is a finite multiset of programs, where each program contains exactly* k *rules, which are in one of the following forms each:*
  - $a \to b$, *called the evolution rule,*
  - $c \leftrightarrow d$, *called the communication rule,*
  - r_1/r_2, *called the checking rule;* r_1, r_2 *are the evolution rules or the 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 the multiset O_i for $1 \le i \le n$ and by the set V_E. Formally, the configuration of the P colony Π is given by $(w_1, \dots, w_n, w_E)$, where $|w_i| = k$, $1 \le i \le n$, w_i represents all the objects placed inside the i-th agent, and $w_E \in (A - \{e\})^\circ$ 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 usable program. If the number of applicable programs are higher than one, then the agent chooses one of the rule nondeterministically. At one step of the 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 $i \ne j$, $1 \le i, j \le n$.

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

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

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

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

For a program p and any $\alpha \in \{left, right, export, import\}$, let be
$$\alpha\,(p, w) = \cup_{r \in p} \alpha\,(r, w).$$

A transition from a configuration to another is denoted as
$(w_1, \dots, w_n; w_E) \Rightarrow (w'_1, \dots, w'_n; w'_E)$, 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 $\bigcup\limits_{p \in P} import\,(p, w_E) \subseteq w_E$.
- Furthermore, the chosen set P is maximal, i.e. 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 listed 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 be $w'_j = right\,(p, w_E) \cup import\,(p, w_E)$. If there is no $p \in P$ with $p \in lab\,(P_j)$ for some j, $1 \leq j \leq n$, then let be $w'_j = w_j$ and moreover, let be

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

A configuration is halting if the set of program labels P satisfying the conditions above cannot vary from the empty set. A set of all possible halting configurations is denoted by H. A halting computation can be associated with the 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 $\Rightarrow$.

Consider a P colony $\Pi = (A,\ e,\ f,\ V_E,\ B_1,\ \ldots,\ B_n)$. The maximal number of programs associated with the agents in the P colony Π are called the height of the P colony Π. The degree of the P colony Π is the number of agents in it. The third parameter characterizing the P colony is the capacity of the 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 the P colonies working in a 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 the checking rules, then the family of all sets of numbers computed by the P colonies is denoted by $NPCOL_{par}K$. If the P colonies are restricted, we use the notation $NPCOL_{par}R$, respectively $NPCOL_{par}KR$.

2.2 Register machines

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

Definition 2. *[9] 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 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 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, it can be assumed 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: the computation starts with all registers empty (hence storing the number zero) and with the instruction labeled l_0. The computation proceeds by applying the instructions indicated by the labels (and the content of registers allows its application). 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 can compute all sets of numbers which are Turing computable.

Moreover, we call a register machine partially blind [6] if we interpret a subtract instruction in the following way: $l_1 : (SUB(r); l_2; l_3)$ - if there is a value different from zero in the register r, then subtract one from its contents and go to instruction l_2 or to instruction l_3; if there is stored zero in the register r when attempting to decrement the 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. It means that each program is formed by only one rule, either the evolution rule or the communication rule. If all agents have their programs with the evolution rules, the agents "live only for themselves" and do not communicate with the environment.

Following results were proved:
– $NPCOL_{par}K(1, *, 7) = NRE$ in [1],
– $NPCOL_{par}K(1, 4, *) = NRE$ in [2],
– $NPCOL_{par}(1, 2, *) = NPBRM$ in [2].

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

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 the initial label l_0. After generating the symbol l_0, the agent stops. It can continue its activity only by using a program with the communication 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 $A = \{l_i, L_i, \textcircled{l_i}, \boxed{l_i}, \boxed{L_i}, m_i, m_i', \textcircled{m_i}, \boxed{m_i}, y_i, n_i, \mid 0 \leq i \leq |H|\} \cup$
$\cup \{a_i | 1 \leq i \leq m\} \cup \{A_r^i \mid \text{for every } l_i : (SUB(r), l_j, l_k) \in H\} \cup$
$\cup \{e, d, C\}$,

– $f = a_1$,

– $B_i = (e, P_i),\ 1 \leq i \leq 4$.

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

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

(2) We need an additional agent to generate a special object d. This agent will be working during whole computation. In each pair of steps the agent B_2 places a copy of d to the environment. This agent stops working when it consumes the symbol which is generated by the simulation of the instruction l_h from the environment.

P_2 :
$2 : \langle e \to d \rangle$, $3 : \langle d \leftrightarrow e \rangle$, $4 : \langle d \leftrightarrow l_h \rangle$;

The P colony Π starts its computation in the initial configuration $(e, e, e, e, \varepsilon)$. In the first subsequence of steps of the P colony Π only agents B_1, B_2 can apply their 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	2		
2.	l_0	d	e	e			3		
3.	l_0	e	e	e	d		2		

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

P_1	P_1	P_3	P_3
$5 : \langle l_1 \leftrightarrow D_1 \rangle$,	$9 : \left\langle \boxed{l_1} \rightarrow \boxed{L_1} \right\rangle$,	$13 : \langle e \leftrightarrow D_1 \rangle$,	$16 : \langle e \leftrightarrow \enclose{circle}{l_1} \rangle$,
$6 : \langle D_1 \leftrightarrow d \rangle$,	$10 : \left\langle \boxed{L_1} \rightarrow L_1 \right\rangle$,	$14 : \left\langle D_1 \rightarrow \boxed{l_1} \right\rangle$,	$17 : \langle \enclose{circle}{l_1} \rightarrow a_r \rangle$,
$7 : \langle d \rightarrow \enclose{circle}{l_1} \rangle$,	$11 : \langle L_1 \rightarrow l_2 \rangle$,	$15 : \left\langle \boxed{l_1} \leftrightarrow e \right\rangle$,	$18 : \langle a_r \leftrightarrow e \rangle$,
$8 : \left\langle \enclose{circle}{l_1} \leftrightarrow \boxed{l_1} \right\rangle$,	$12 : \langle L_1 \rightarrow l_3 \rangle$,		

This pair of agents generate two objects. One object increments value of the particular register and the second one defines of which instruction will simulation continue. One agent is not able to generate both objects corresponding to the simulation of one instruction, because at the moment of placing all of its content into the environment via the communication rules, it does not know which instruction it simulates. It nondeterministically chooses one of the possible instructions. Now it is necessary to check whether the agent has chosen the right instruction. For this purpose the second agent slightly changes first generated object. The first agent swaps this changed object for the new one generated only if it belongs to the same instruction. If this is not done succesfully, the computation never stops because of absence of the halting object for the agent B_2.

An instruction $l_1 : (ADD(r), l_2, l_3)$ 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.	l_1	d	e	e	$a_r^u d^v$	5	3		
2.	D_1	e	e	e	$a_r^u d^{v+1}$	6	2		
3.	d	d	e	e	$D_1 a_r^u d^v$	7	3	13	
4.	$\enclose{circle}{l_1}$	e	D_1	e	$a_r^u d^{v+1}$		2	14	
5.	$\enclose{circle}{l_1}$	d	$\boxed{l_1}$	e	$a_r^u d^{v+1}$		3	15	
6.	$\enclose{circle}{l_1}$	e	e	e	$\boxed{l_1} a_r^u d^{v+2}$	8	2		
7.	$\boxed{l_1}$	d	e	e	$\enclose{circle}{l_1} a_r^u d^{v+2}$	9	3	16	
8.	$\boxed{L_1}$	e	$\enclose{circle}{l_1}$	e	$a_r^u d^{v+3}$	10	2	17	
9.	L_1	d	a_r	e	$a_r^u d^{v+3}$	**11** or 12	3	18	
10.	l_2	e	e	e	$a_r^{u+1} d^{v+4}$				

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

P_1	P_1	P_3	P_3
$19 : \langle l_1 \to D_1 \rangle$,	$28 : \langle l_1' \to n^1 \rangle$,	$36 : \langle e \leftrightarrow D_1 \rangle$,	$45 : \langle e \leftrightarrow y^1 \rangle$,
$20 : \langle D_1 \leftrightarrow d \rangle$,	$29 : \langle n^1 \leftrightarrow m_1 \rangle$,	$37 : \langle D_1 \to \boxed{l_1} \rangle$,	$46 : \langle y^1 \to l_2 \rangle$,
$21 : \langle d \to \textcircled{l_1} \rangle$,	$30 : \langle m_1 \to m_1' \rangle$,	$38 : \langle \boxed{l_1} \leftrightarrow e \rangle$,	$47 : \langle l_2 \leftrightarrow e \rangle$,
$22 : \langle \textcircled{l_1} \leftrightarrow \boxed{l_1} \rangle$,	$31 : \langle m_1' \to m_1'' \rangle$,	$39 : \langle e \leftrightarrow \textcircled{l_1} \rangle$,	$48 : \langle e \leftrightarrow n^1 \rangle$,
$23 : \langle \boxed{l_1} \to A_r^1 \rangle$,	$32 : \langle m_1'' \to \textcircled{m_1} \rangle$,	$40 : \langle \textcircled{l_1} \to l_1' \rangle$,	$49 : \langle n_1 \to l_3 \rangle$,
$24 : \langle A_r^1 \leftrightarrow a_r \rangle$,	$33 : \langle \textcircled{m_1} \leftrightarrow l_2 \rangle$,	$41 : \langle l_1' \leftrightarrow e \rangle$,	$50 : \langle l_3 \leftrightarrow e \rangle$,
$25 : \langle A_r^1 \leftrightarrow l_1' \rangle$,	$34 : \langle \textcircled{m_1} \to \boxed{m_1} \rangle$,	$42 : \langle e \leftrightarrow A_r^1 \rangle$,	
$26 : \langle a_r \to y^1 \rangle$,	$35 : \langle \boxed{m_1} \leftrightarrow l_3 \rangle$,	$43 : \langle A_r^1 \to m_1 \rangle$,	
$27 : \langle y^1 \leftrightarrow m_1 \rangle$,		$44 : \langle m_1 \leftrightarrow e \rangle$,	

Agents B_1, B_3 and B_4 collectively check the state of particular register and generate label of following instruction. Part of the simulation is devoted to purify the environment from redundant objects.

P_4	
$51 : \langle e \leftrightarrow \textcircled{m_1} \rangle$,	$54 : \langle l_1' \to e \rangle$,
$52 : \langle \textcircled{m_1} \to d \rangle$,	$55 : \langle e \leftrightarrow \boxed{m_1} \rangle$,
$53 : \langle d \leftrightarrow l_1' \rangle$,	$56 : \langle \boxed{m_1} \to e \rangle$,

The instruction $l_1 : (SUB(r), l_2, l_3)$ is simulated by the following sequence of steps. If 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.	l_1	d	e	e	d^v	19	3		
2.	D_1	e	e	e	d^{v+1}	20	2		
3.	d	d	e	e	D_1d^v	21	3	36	
4.	$\textcircled{$l_1$}$	e	D_1	e	d^{v+1}		2	37	
5.	$\textcircled{$l_1$}$	d	$\boxed{l_1}$	e	d^{v+1}		3	38	
6.	$\textcircled{$l_1$}$	e	e	e	$\boxed{l_1}d^{v+2}$	22	2		
7.	$\boxed{l_1}$	d	e	e	$\textcircled{$l_1$}d^{v+2}$	23	3	39	
8.	A_r^1	e	$\textcircled{$l_1$}$	e	d^{v+3}		2	40	
9.	A_r^1	d	l_1'	e	d^{v+3}		3	41	
10.	A_r^1	e	e	e	$l_1'd^{v+4}$	25	2		
11.	l_1'	d	e	e	$A_r^1d^{v+4}$	28	3	42	
12.	n_1	e	A_r^1	e	d^{v+5}		2	43	
13.	n_1	d	m_1	e	d^{v+5}		3	44	
14.	n_1	e	e	e	m_1d^{v+6}	29	2		
15.	m_1	d	e	e	n_1d^{v+6}	30	3	48	
16.	m_1'	e	n_1	e	d^{v+7}	31	2	49	
17.	m_1''	d	l_3	e	d^{v+7}	32	3	50	
18.	$\textcircled{$m_1$}$	e	e	e	l_3d^{v+8}	34	2		
19.	$\boxed{m_1}$	d	e	e	l_3d^{v+8}	35	3		
20.	l_3	e	e	e	$\boxed{m_1}d^{v+9}$		2		55
21.	l_3	d	e	$\boxed{m_1}$	d^{v+9}		3		56
22.	l_3	e	e	e	d^{v+10}		2		

If the register r stores a value different from 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.	l_1	d	e	e	$a_r^ud^v$	19	3		
2.	D_1	e	e	e	$a_r^ud^{v+1}$	20	2		
3.	d	d	e	e	$D_1a_r^ud^v$	21	3	36	
4.	$\textcircled{$l_1$}$	e	D_1	e	$a_r^ud^{v+1}$		2	37	
5.	$\textcircled{$l_1$}$	d	$\boxed{l_1}$	e	$a_r^ud^{v+1}$		3	38	
6.	$\textcircled{$l_1$}$	e	e	e	$\boxed{l_1}a_r^ud^{v+2}$	22	2		
7.	$\boxed{l_1}$	d	e	e	$\textcircled{$l_1$}a_r^ud^{v+2}$	23	3	39	

	configuration of Π					applicable programs			
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
8.	A_r^1	e	$\textcircled{1}$	e	$a_r^u d^{v+3}$	24	2	40	
9.	a^r	d	l_1'	e	$A_r^1 a_r^{u-1} d^{v+3}$	26	3	41	
10.	y_1	e	e	e	$l_1' A_r^1 a_r^{u-1} d^{v+4}$		2	42	
11.	y_1	d	A_r^1	e	$l_1' a_r^{u-1} d^{v+4}$	28	3	43	
12.	y_1	e	m_1	e	$l_1' a_r^{u-1} d^{v+5}$		2	44	
13.	y_1	d	e	e	$m_1 l_1' a_r^{u-1} d^{v+5}$	27	3		
14.	m_1	e	e	e	$y_1 l_1' a_r^{u-1} d^{v+6}$	30	2	45	
15.	m_1'	d	y_1	e	$l_1' a_r^{u-1} d^{v+6}$	31	3	46	
16.	m_1''	e	l_2	e	$l_1' a_r^{u-1} d^{v+7}$	32	2	47	
17.	$\textcircled{m_1}$	d	e	e	$l_2 l_1' a_r^{u-1} d^{v+7}$	33	3		
18.	l_2	e	e	e	$\textcircled{m_1} l_1' a_r^{u-1} d^{v+8}$		2		51
19.	l_2	d	e	$\textcircled{m_1}$	$l_1' a_r^{u-1} d^{v+8}$		3		52
20.	l_2	e	e	d	$l_1' a_r^{u-1} d^{v+9}$		2		53
21.	l_2	d	e	l_1'	$a_r^{u-1} d^{v+10}$		3		54
22.	l_2	e	e	e	$a_r^{u-1} d^{v+11}$		2		

(5) The halting instruction l_h is simulated by the agent B_1 with a subset of programs:

P_1
$57 : \langle l_h \leftrightarrow d \rangle$.

The agent places the object l_h into the environment, from where it can be consumed by the agent B_2 and by this the agent B_2 stops its activity.

	configuration of Π								
step	B_1	B_2	B_3	B_4	Env	P_1	P_2	P_3	P_4
1.	l_h	e	e	e	d^v	57	2		
2.	d	e	d	e	$l_h d^v$		4		
3.	d	l_h	e	e	d^{v+1}				

The P colony Π correctly simulates computation of 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 of M starts with zeros in all registers. The computation of Π stops if the symbol l_h is placed inside the agent B_2 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. $NPCOL_{par}(1, *, 8) = NRE$

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 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 the communication 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

$\Pi = (A, e, f, \emptyset, B_1, \ldots, B_n),\ n = |H| + 2$ where:

– alphabet $A = \{l_i, l_i', i', i'', \textcircled{i}, \boxed{i}, D_i, \overline{l_i} | 0 \leq i \leq |H|\} \cup$
$\cup\ \{a_i | 1 \leq i \leq m\} \cup \{e, d\}$,

– $f = a_1$,

– $B_i = (e, P_i),\ 1 \leq i \leq 4$.

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

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

(2) We need an additional agent to generate a special object d. This agent will be working during whole computation. In each pair of steps the agent B_2 places a copy of d to the environment..

P_2 :
$3 : \langle e \to d \rangle$, $4 : \langle d \leftrightarrow e \rangle$, $5 : \langle d \leftrightarrow l_h \rangle$;

The P colony Π starts its computation in the initial configuration $(e, e, e, e, \varepsilon)$. In the first subsequence of steps of the P colony Π, only the agents B_1 and B_2 can apply their 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	e	e	d		4		

(3) To simulate the ADD-instruction $l_1 : (ADD(r), l_2, l_3)$, there are two agents $B_{l_1^1}$ and $B_{l_1^2}$ in the P colony Π. These agents help each other to add one copy of the object a_r and the object l_2 or l_3 to the environment.

$P_{l_1^1}$	$P_{l_1^1}$	$P_{l_1^2}$
$6 : \langle e \leftrightarrow l_1 \rangle$,	$10 : \langle d \to l_3 \rangle$,	$13 : \langle e \leftrightarrow D_1 \rangle$,
$7 : \langle l_1 \to D_1 \rangle$,	$11 : \langle l_2 \leftrightarrow e \rangle$,	$14 : \langle D_1 \to a_r \rangle$,
$8 : \langle D_1 \leftrightarrow d \rangle$,	$12 : \langle l_3 \leftrightarrow e \rangle$,	$15 : \langle a_r \leftrightarrow e \rangle$,
$9 : \langle d \to l_2 \rangle$,		

The instruction $l_1 : (ADD(r), l_2, l_3)$ is simulated by the following sequence of steps. Let the content of the agent B_2 be d.

	configuration of Π						
step	B_2	$B_{l_1^1}$	$B_{l_1^2}$	Env	P_2	P_3	P_4
1.	d	e	e	$l_1 a_r^u d^v$	3	6	
2.	e	l_1	e	$a_r^u d^{v+1}$	2	7	
3.	d	D_1	e	$a_r^u d^{v+1}$	3	8	
4.	e	d	e	$D_1 a_r^u d^{v+1}$	2	**9** or 10	13
5.	d	l_2	D_1	$a_r^u d^{v+1}$	3	11	14
6.	e	e	a_r	$l_2 a_r^u d^{v+2}$	2		15
7.	d	e	e	$l_2 a_r^{u+1} d^{v+2}$	3		

(4) For each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, the below mentioned programs are introduced in the sets $P_{l_1^1}$, $P_{l_1^2}$, $P_{l_1^3}$, $P_{l_1^4}$ and in the set $P_{l_1^5}$:

$P_{l_1^1}$	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^5}$
$16 : \langle e \leftrightarrow l_1 \rangle$,	$19 : \langle d \rightarrow \boxed{1} \rangle$,	$21 : \langle e \leftrightarrow D_1 \rangle$,	$32 : \langle e \leftrightarrow \textcircled{1} \rangle$,
$17 : \langle l_1 \rightarrow D_1 \rangle$,	$20 : \langle \boxed{1} \leftrightarrow e \rangle$,	$22 : \langle D_1 \rightarrow \textcircled{1} \rangle$,	$33 : \langle \textcircled{1} \rightarrow 1'' \rangle$,
$18 : \langle D_1 \leftrightarrow d \rangle$,		$23 : \langle \textcircled{1} \leftrightarrow e \rangle$,	$34 : \langle 1'' \leftrightarrow e \rangle$,

$P_{l_1^3}$	$P_{l_1^3}$	$P_{l_1^5}$	$P_{l_1^5}$
$24 : \langle e \leftrightarrow \boxed{1} \rangle$,	$28 : \langle l_2 \leftrightarrow e \rangle$,	$35 : \langle e \leftrightarrow 1' \rangle$,	$39 : \langle 1'' \rightarrow e \rangle$,
$25 : \langle \boxed{1} \rightarrow 1' \rangle$,	$29 : \langle 1' \leftrightarrow 1'' \rangle$,	$36 : \langle 1' \rightarrow d \rangle$,	$40 : \langle \overline{l_3} \rightarrow l_3 \rangle$,
$26 : \langle 1' \leftrightarrow a_r \rangle$,	$30 : \langle 1'' \rightarrow \overline{l_3} \rangle$,	$37 : \langle d \leftrightarrow 1'' \rangle$,	$41 : \langle l_3 \leftrightarrow e \rangle$,
$27 : \langle a_r \rightarrow l_2 \rangle$,	$31 : \langle \overline{l_3} \leftrightarrow e \rangle$,	$38 : \langle d \leftrightarrow \overline{l_3} \rangle$,	

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

If the register r stores a value different from zero, then the computation proceeds as follows (we do not consider the number of copies of the object d in the environment):

	configuration of Π						applicable programs				
step	$B_{l_1^1}$	$B_{l_1^2}$	$B_{l_1^3}$	$B_{l_1^4}$	$B_{l_1^5}$	Env	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^3}$	$P_{l_1^4}$	$P_{l_1^5}$
1.	e	e	e	e	e	$l_1 a_r^u d^v$	16				
2.	l_1	e	e	e	e	$a_r^u d^v$	17				
3.	D_1	e	e	e	e	$a_r^u d^v$	18				
4.	d	e	e	e	e	$D_1 a_r^u d^{v-1}$	19	21			
5.	$\boxed{1}$	D_1	e	e	e	$a_r^u d^{v-1}$	20	22			
6.	e	$\textcircled{1}$	e	e	e	$\boxed{1} a_r^u d^{v-1}$		23	24		
7.	e	e	$\boxed{1}$	e	e	$\textcircled{1} a_r^u d^{v-1}$			25	32	
8.	e	e	$1'$	$\textcircled{1}$	e	$a_r^u d^{v-1}$			26	33	
9.	e	e	a_r	$1''$	e	$1' a_r^{u-1} d^{v-1}$			27	34	35
10.	e	e	l_2	e	$1'$	$1'' a_r^{u-1} d^{v-1}$			28		36
11.	e	e	e	e	d	$l_2 1'' a_r^{u-1} d^{v-1}$					37
12.	e	e	e	e	$1''$	$l_2 a_r^{u-1} d^v$					39
13.	e	e	e	e	e	$l_2 a_r^{u-1} d^v$					

From the 12th step the agent $B_{l_2^1}$ starts to work and consumes the object l_2. We do not notice this fact in the table.

When the value in the counter r is zero:

	configuration of Π						applicable programs				
step	$B_{l_1^1}$	$B_{l_1^2}$	$B_{l_1^3}$	$B_{l_1^4}$	$B_{l_1^5}$	Env	$P_{l_1^1}$	$P_{l_1^2}$	$P_{l_1^3}$	$P_{l_1^4}$	$P_{l_1^4}$
1.	e	e	e	e	e	l_1d^v	16				
2.	l_1	e	e	e	e	d^v	17				
3.	D_1	e	e	e	e	d^v	18				
4.	d	e	e	e	e	D_1d^{v-1}	19	21			
5.	$\boxed{1}$	D_1	e	e	e	d^{v-1}	20	22			
6.	e	$\textcircled{1}$	e	e	e	$\boxed{1}d^{v-1}$		23	24		
7.	e	e	$\boxed{1}$	e	e	$\textcircled{1}d^{v-1}$			25	32	
8.	e	e	$1'$	$\textcircled{1}$	e	d^{v-1}				33	
9.	e	e	$1'$	$1''$	e	d^{v-1}				34	
10.	e	e	$1'$	e	e	$1''d^{v-1}$			29		
11.	e	e	$1''$	e	e	$1'd^{v-1}$			30		35
12.	e	e	$\overline{l_3}$	e	$1'$	d^{v-1}			31		36
13.	e	e	e	e	d	$\overline{l_3}d^{v-1}$					38
14.	e	e	e	e	$\overline{l_3}$	d^v					40
15.	e	e	e	e	l_3	d^v					41
16.	e	e	e	e	e	l_3d^v					

(5) The halting instruction l_h is simulated by agent B_2 which consumes the object l_h and that stops the computation.

The P colony Π correctly simulates the computation of the register machine M. The computation of the Π starts with no object a_r, which indicates the content of the register r, placed in the environment, in the same way as the computation in the register machine M starts with zeros in all registers. Then the agents simulate the computation by simulating ADD and SUB instructions. The computation of the P colony Π stops if the symbol l_h is placed inside the corresponding agent as well as the register machine 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.

□

4 Modularity in the terms of P colonies

During the evolution unicellular organisms have evolved into multicellular. Some cells specialized their activities for the particular function and have to cooperate with other specialized cells to be alive. In that way the organs have evolved and living organisms have become more complex. But the cooperating organs and more complex living organisms are more sophisticated, live longer and their life is improving.

From the previous section we can observe that some agents in the P colonies are providing the same function during the computation. This inspired us to introduce the modules in the P colonies. We have defined five modules, where each of them is providing one specific function. These modules are the module for the duplication, the module for the addition, the module for the subtraction, the balance-wheel module, the control module (see Fig. 1). Definition of each module's function is given in the proof of following theorem.

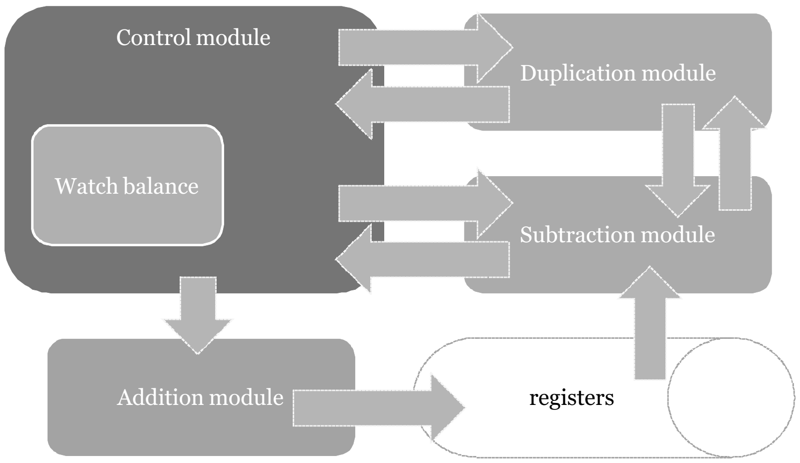

Fig. 1. Modular P colony

Theorem 3. $NPCOL_{par}(1, 8, *) = NRE$.

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

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

We can group the agents of the P colony into five modules. Each module needs for its work an imput and requires some objects. The result of its computation is an output:

(1) module for the duplication (uses 2 agents):

P_1 :	P_2 :
$1:\ \langle e \to D_i \rangle\,,$	$8:\ \left\langle e \leftrightarrow \boxed{D_i} \right\rangle,$
$2:\ \left\langle D_i \to \boxed{D_i} \right\rangle,$	$9:\ \left\langle \boxed{D_i} \to i' \right\rangle,$
$3:\ \left\langle \boxed{D_i} \leftrightarrow d \right\rangle,$	$10:\ \langle i' \leftrightarrow e \rangle\,,$
$4:\ \left\langle d \to \boxed{i} \right\rangle,$	$11:\ \left\langle e \leftrightarrow \boxed{i} \right\rangle,$
$5:\ \left\langle \boxed{i} \leftrightarrow i' \right\rangle,$	$12:\ \left\langle \boxed{i} \to \textcircled{i} \right\rangle,$
$6:\ \langle i' \to i \rangle\,,$	$13:\ \left\langle \textcircled{i} \leftrightarrow e \right\rangle.$
$7:\ \langle i \leftrightarrow e \rangle\,,$	

Duplicating module is activated when the object D_i appears in the environment. This object carries a message "Duplicate object i.".

input:	one object D_i
output:	one object i after 10 steps and one object $\textcircled{i}$ after 11 steps
requirements:	one object d

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	e	e	dD_i	1	–
2.	D_i	e	d	2	–
3.	$\boxed{D_i}$	e	d	3	–
4.	d	e	$\boxed{D_i}$	4	8
5.	$\boxed{i}$	$\boxed{D_i}$		–	9
6.	$\boxed{i}$	i'		–	10
7.	$\boxed{i}$	e	i'	5	–
8.	i'	e	$\boxed{i}$	6	11
9.	i	$\boxed{i}$		7	12
10.	e	$\textcircled{i}$	i	–	13
11.	e	e	$\textcircled{i}i$	–	–

Duplicating module duplicates requested object.

(2) module for the addition (uses 1 agent):

P_1 :
$1:\ \langle e \leftrightarrow A_r \rangle\,,$
$2:\ \langle A_r \to a_r \rangle\,,$
$3:\ \langle a_r \leftrightarrow e \rangle\,.$

input:	one object A_r
output:	one object a_r after 4 steps
requirements:	$\emptyset$

	configuration of Π		
	B_1	Env	P_1
1.	e	A_r	1
2.	A_r		2
3.	a_r		3
4.	e	a_r	–

Module for the addition adds one symbol into the environment.

(3) module for the subtraction (uses 3 agents):

P_1 :	P_2 :	P_3 :
$1: \langle e \leftrightarrow S_r \rangle$,	$11: \langle e \leftrightarrow \enclose{circle}{B_r} \rangle$,	$19: \langle e \leftrightarrow y' \rangle$,
$2: \langle S_r \rightarrow D_{B_r} \rangle$,	$12: \langle \enclose{circle}{B_r} \rightarrow \enclose{circle}{B'_r} \rangle$,	$20: \langle y' \rightarrow y \rangle$,
$3: \langle D_{B_r} \leftrightarrow d \rangle$,	$13: \langle \enclose{circle}{B'_r} \leftrightarrow a_r \rangle$	$21: \langle y \leftrightarrow B'_r \rangle$,
$4: \langle d \leftrightarrow B_r \rangle$,	$14: \langle \enclose{circle}{B'_r} \leftrightarrow B'_r \rangle$	$22: \langle B'_r \rightarrow e \rangle$,

P_1 :	P_2 :	P_3 :
$5: \langle B_r \rightarrow \underline{B}_r \rangle$,	$15: \langle a_r \rightarrow y' \rangle$	
$6: \left\langle \underline{B}_r \rightarrow \overline{B}_r \right\rangle$,	$16: \langle B'_r \rightarrow n \rangle$	
$7: \left\langle \overline{B}_r \rightarrow B'_r \right\rangle$,	$17: \langle y' \leftrightarrow e \rangle$	
$8: \langle B'_r \leftrightarrow d \rangle$,	$18: \langle n \leftrightarrow e \rangle$	
$9: \langle d \leftrightarrow \enclose{circle}{B'_r} \rangle$,		
$10: \langle \enclose{circle}{B'_r} \rightarrow e \rangle$,		

input:	one object B_r
output:	one object y after 23 steps or one object n after 22 steps
requirements:	two objects d, object a_r (if there is at least one in the environment)
uses:	duplication module

	configuration of Π						
	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	e	e	e	$S_r a_r d$	1	–	–
2.	S_r	e	e	$a_r d$	2	–	–
3.	D_{B_r}	e	e	$a_r d$	3	–	–
4.	d	e	e	$a_r D_{B_r}$	–	–	–
	waiting for objects from duplication unit						
14.	d	e	e	$a_r B_r$	4	–	–
15.	B_r	e	e	$a_r \enclose{circle}{B_r}$	5	11	–
16.	$\underline{B}_r$	$\enclose{circle}{B_r}$	e	a_r	6	12	–
17.	$\overline{B}_r$	$\enclose{circle}{B'_r}$	e	a_r	7	13	–
18.	B'_r	a_r	e	$\enclose{circle}{B'_r}$	8	15	–
19.	d	y'	e	$\enclose{circle}{B'_r} B'_r$	9	17	–
20.	$\enclose{circle}{B'_r}$	e	e	$y' B'_r$	10	–	19
21.	e	e	y'	B'_r	–	–	20
22.	e	e	y	B'_r	–	–	21
23.	e	e	B'_r	y	–	–	22
24.	e	e	e	y	–	–	–

	configuration of Π						
	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	e	e	e	$S_r d$	1	–	–
2.	S_r	e	e	d	2	–	–
3.	D_{B_r}	e	e	d	3	–	–
4.	d	e	e	D_{B_r}	–	–	–
	waiting for objects from duplication unit						
14.	d	e	e	B_r	4	–	–
15.	B_r	e	e	$\textcircled{B_r}$	5	11	–
16.	$\underline{B}_r$	$\textcircled{B_r}$	e		6	12	–
17.	$\overline{B}_r$	$\textcircled{B'_r}$	e		7	–	–
18.	B'_r	$\textcircled{B'_r}$	e		8	–	–
19.	d	$\textcircled{B'_r}$	e	B'_r	–	14	–
20.	d	B'_r	e	$\textcircled{B'_r}$	9	16	–
21.	$\textcircled{B'_r}$	n	e		10	–	–
22.	e	e	e	n	–	–	–

Module for the subtraction removes requested object from the environment.

(4) Balance-wheel module (uses 1 agent):

P_1 :
$1: \ \langle e \to d \rangle$
$2: \ \langle d \leftrightarrow e \rangle$
$3: \ \langle d \leftrightarrow \textcircled{f} \rangle$

The balance-wheel module "keeps the computation alive". It inserts the objects d into the environment until it consumes a special symbol $\textcircled{f}$ from the environment. This action makes it stop working. The object $\textcircled{f}$ gets into the environment from the duplicating module which is activated by the simulation of the halt instruction by the control module.

(5) Control module (uses 2 agents):

a) initialization:

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

First agent in this module generates label of the first instruction of the register machine.

b) adding instruction $l_1 : (ADD(r), l_2, l_3)$:

P_1 :	notes
$1: \ \langle l_1 \to D_1 \rangle$,	
$2: \ \langle D_1 \leftrightarrow d \rangle$,	$\longrightarrow$ Duplication module
$3: \ \langle d \leftrightarrow 1 \rangle$,	$\longleftarrow$ Duplication module
$4: \ \langle 1 \to B_r \rangle$,	
$5: \ \langle B_r \leftrightarrow \textcircled{1} \rangle$,	$\longleftarrow$ Duplication module $\longrightarrow$ Addition module
$6: \ \langle \textcircled{1} \to l_2 \rangle$,	
$7: \ \langle \textcircled{1} \to l_3 \rangle$,	

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	l_1	e	d	1	–
2.	D_1	e	d	2	–
3.	d	e	D_1	–	–
	waiting for response of duplication unit				
13.	d	e	1	3	–
14.	1	e	$d\textcircled{1}$	4	–
15.	B_r	e	$d\textcircled{1}$	5	–
16.	$\textcircled{1}$	e	B_r	**6** or 7	–
17.	l_2	e	d	–	–

c) subtracting instruction $l_1 : (SUB(r), l_2, l_3)$:

P_1 :	notes	P_2 :
$1 : \langle l_1 \to D_1 \rangle$,		$16 : \langle e \leftrightarrow L_1 \rangle$,
$2 : \langle D_1 \leftrightarrow d \rangle$,	$\longrightarrow$ Duplication module	$17 : \langle L_1 \to \boxed{L_1} \rangle$,
$3 : \langle d \leftrightarrow 1 \rangle$,	$\longleftarrow$ Duplication module	$18 : \langle \boxed{L_1} \leftrightarrow e \rangle$,
$4 : \langle 1 \to S_r \rangle$,		$19 : \langle e \leftrightarrow L_1' \rangle$,
$5 : \langle S_r \leftrightarrow \textcircled{1} \rangle$,	$\longleftarrow$ Duplication module $\longrightarrow$ Subtraction module	$20 : \langle e \leftrightarrow L_1'' \rangle$,
$6 : \langle \textcircled{1} \to L_1 \rangle$,		$21 : \langle L_1' \to l_2 \rangle$,
$7 : \langle L_1 \leftrightarrow y \rangle$,	$\longleftarrow$ Subtraction module	$22 : \langle L_1'' \to l_3 \rangle$,
$8 : \langle L_1 \leftrightarrow n \rangle$,	$\longleftarrow$ Subtraction module	$23 : \langle l_2 \leftrightarrow e \rangle$,
$9 : \langle y \to L_1' \rangle$,		$24 : \langle l_3 \leftrightarrow e \rangle$,
$10 : \langle n \to L_1'' \rangle$,		
$11 : \langle L_1' \leftrightarrow \boxed{L_1} \rangle$,		
$12 : \langle L_1'' \leftrightarrow \boxed{L_1} \rangle$,		
$13 : \langle \boxed{L_1} \to d \rangle$,		
$14 : \langle d \leftrightarrow l_2 \rangle$,		
$15 : \langle d \leftrightarrow l_3 \rangle$,		

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
1.	l_1	e	d	1	–
2.	D_1	e	d	2	–
3.	d	e	D_1	–	–
	waiting for response of duplication module				
13.	d	e	1	3	–
14.	1	e	$d\textcircled{1}$	4	–
15.	S_r	e	$d\textcircled{1}$	5	–
16.	$\textcircled{1}$	e	S_r	6	–
17.	L_1	e	d	–	–
	waiting for response of subtraction module				

If subtraction module generates y

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
49.	L_1	e	y	7	–
50.	y	e	L_1	9	16
51.	L_1'	L_1		–	17
52.	L_1'	$\boxed{L_1}$		–	18
53.	L_1'	e	$\boxed{L_1}$	11	–
54.	$\boxed{L_1}$	e	L_1'	13	19
55.	d	L_1'		–	21
56.	d	l_2		–	23
57.	d	e	l_2	14	–
58.	l_2	e	d	–	–

If subtraction module generates n

	configuration of Π				
	B_1	B_2	Env	P_1	P_2
48.	L_1	e	n	8	–
49.	n	e	L_1	10	16
50.	L_1''	L_1		–	17
51.	L_1''	$\boxed{L_1}$		–	18
52.	L_1''	e	$\boxed{L_1}$	12	–
53.	$\boxed{L_1}$	e	L_1''	13	20
54.	d	L_1''		–	22
55.	d	l_3		–	24
56.	d	e	l_3	15	–
57.	l_3	e	d	–	–

d) halting instruction l_h:

P_1 :	
$1:\ \langle l_f \to D_f \rangle$	
$2:\ \langle D_f \leftrightarrow d \rangle$	$\longrightarrow$ Duplication module
$3:\ \langle d \leftrightarrow f \rangle$	$\longleftarrow$ Duplication module

Control module controls all the computation. It sends necessary objects into the environment for the work of the other modules.

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

5 Conclusion

We have proved 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 also verified 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 the Czech Republic grants No. 201/06/0567 and SGS/5/2010.

References

1. Ciencialová, L., Cienciala, L.: *Variations on the theme: P Colonies*, Proceedings of the 1st International workshop WFM'06 (Kolář, D., Meduna, A., eds.), Ostrava, 2006, pp. 27–34.
2. Ciencialová, L. Cienciala, L., Kelemenová, A.: *On the number of agents in P colonies*, In G. Eleftherakis, P. Kefalas, and G. Paun (eds.), *Proceedings of the 8th Workshop on Membrane Computing (WMC'07)*, June 25-28, Thessaloniki, Greece, 2007, pp. 227–242.
3. 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.
4. 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 (H. J. Hoogeboom, Gh. Păun, G. Rozenberg, eds.), Leiden, The Netherlands, 2006, pp. 311–322.
5. 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.
6. Greibach, S. A.: *Remarks on blind and partially blind one-way multicounter machines.* Theoretical Computer Science, 7(1), 1978, pp. 311–324.
7. 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.
8. 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.
9. Minsky, M. L.: *Computation: Finite and Infinite Machines.* Prentice Hall, Engle-wood Cliffs, NJ, 1967.
10. Păun, Gh.: *Computing with membranes.* Journal of Computer and System Sciences 61, 2000, pp. 108–143.
11. Păun, Gh.: *Membrane computing: An introduction.* Springer-Verlag, Berlin, 2002.
12. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
13. P systems web page: http://psystems.disco.unimib.it