Computing and Informatics, Vol. 27, 2008, 481–496

HOMOGENEOUS P COLONIES

Luděk Cienciala, Lucie Ciencialová

Institute of Computer Science
Silesian University in Opava, Bezručovo náměstí 13
746 01 Opava, Czech Republic
e-mail: {ludek.cienciala, lucie.ciencialova}@fpf.slu.cz

Alica Kelemenová

Institute of Computer Science
Silesian University in Opava, Bezručovo náměstí 13
746 01 Opava, Czech Republic
&
Department of Computer Science
Catholic University Ružomberok
Nám. Andreja Hlinku 56/1
034 01 Ružomberok, Slovakia
e-mail: alica.kelemenova@fpf.slu.cz

Revised manuscript received 3 December 2007

Abstract. We study P colonies introduced in [8] as a class of abstract computing devices composed of independent membrane agents, acting and evolving in a shared environment. In the present paper especially P colonies are considered, which are homogeneous with respect to the type of rules in each program of agents.

The number of agents, as well as the number of programs in each agent are bounded, which are sufficient to guarantee computational completeness of homogeneous P colonies. We present results for P colonies with one and with two objects inside each agent.

Keywords: P colonies, membrane systems, generative power

Mathematics Subject Classification 2000: 68Q10, 68Q42

1 INTRODUCTION

P colonies were introduced in [8] as formal models of a computing device inspired by membrane systems ([10]) and by colonies, a special grammar systems with simple behavior of components ([6]). This model intends to catch a 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 each agent is constant and determines the capacity of the P colony. The environment contains several copies of a basic environmental object denoted by e. The number of copies of e is unlimited.

A set of programs is associated with each agent. The program determines the activity of the agent by rules. Each program consists of c rules where c is the capacity of the P colony. All the objects inside an agent are evolved (by an evolution rule) or transported (by a communication rule) in every moment of computation. Two such rules can also be combined into checking rule, which sets a priority between these rules: if the first rule is not applicable then the second one should be applied.

The computation starts in the initial configuration, which will be specified for all P colonies in the presented paper in the following way: the environment and all agents contain only copies of object e. Using their programs the agents can change their objects and possibly objects in the environment. This gives the possibility to affect the behavior of the other agents in the next computation steps. Computation is realized in parallel way. In each step of the computation, each agent with at least one applicable program nondeterministically chooses one of them and executes it. The computation halts when no agent can apply any of its programs. The result of the computation is given by the number of some specific objects present in the environment at the end of the computation.

Thus a P colony produces a set of numbers. P colonies are computationally complete. A considerable effort is devoted to minimize the parameters of P colonies preserving their computational completeness.

In the present paper we study the properties of homogeneous P colonies, i.e. the P colonies with programs having the same type of rule (evolution, communication or checking) for all objects inside an agent. Trivially, each P colony with capacity one is homogeneous. Homogeneous P colonies were first considered in [1].

In the present paper we will study the number of agents and the number of programs in the agent needed to achieve computational completeness of the homogeneous P colonies with capacity one and two.

We start with basic notations and definitions in Section 2.

In Section 3 we will deal with P colonies with one object inside each agent. It has been shown in [1] that at most seven programs for each agent as well as five agents guarantee the computational completeness of these P colonies. In the present paper we improve these results as follows: We show that at most six programs in each agent suffice with no limitation to the number of agents and we recall recent result from [2], where the number of agents is reduced to four with no limitation to the number of programs.

Homogeneous P colonies with two objects in each agent are studied in Section 4. Two objects in agents allow to maximally reduce the number of agents, in the sense that computational completeness can be realized by one agent only. Moreover, at most four programs in each agent allow to generate any computable subset of the natural numbers (with no limitation to the number of agents).

2 DEFINITIONS

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory. For more information on membrane computing, see [11], for more information on computational machines and colonies in particular, see [9] and [6, 7, 8], respectively. Activities carried out in the field of membrane computing are currently numerous and a comprehensive information about them is available also at [12].

We briefly summarize the denotations used in the present paper.

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

Let Σ be 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 finite multisets over the finite set V is denoted by V°. The support of M is the set $supp(M) = \{a \in V \mid f_M(a) \neq 0\}$. The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any finite multiset M over V can be represented as a string w over alphabet V with $|w|_a = f(a)$ for all $a \in V$. We write $M = {}_\star w$ in this case, i.e. operator ${}_\star$ associates with w corresponding multiset M. Obviously, all words obtained from w by permuting the letters can also represent the same M, and ${}_\star\varepsilon$ represents the empty multiset.

2.1 P Colonies

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

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

In the case of rules in the same form in the program, we can say that the program is rewriting, communication or checking one. In the case of P colonies with two objects inside each agent the rewriting program can be modified to the form $\langle ab \to cd \rangle$. In the same way we can modify communication programs to the form $\langle ab \leftrightarrow cd \rangle$.

Definition 1. The P colony of the capacity c is a construct

$$\Pi = (A, e, f, {}_\star v_E, B_1, \ldots, B_n),$$

where

- A is an alphabet of the colony, its elements are called objects,
- e is the basic object of the colony, $e \in A$,
- f is the final object of the colony, $f \in A$,
- ${}_\star v_E$ is an initial content of the environment, ${}_\star v_E \in (A - \{e\})^\circ$,
- $B_i,\ 1 \le i \le n$, are agents, each agent is a construct $B_i = ({}_\star o_i, P_i)$, where
  - ${}_\star o_i$ is a multiset over A, it determines the initial state (content) of agent B_i and $|{}_\star o_i| = c$,
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where each program contains exactly c rules, which are in one of the following forms each:
    - $a \to b$, 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 $({}_\star o_1, \ldots, {}_\star o_n, {}_\star v_E)$ of multisets of objects present in the P colony at the beginning of the computation, given by ${}_\star o_i$ for $1 \le i \le n$ and by ${}_\star v_E$. In general, the configuration of P colony Π is given by $({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E)$, where $|{}_\star w_i| = c,\ 1 \le i \le n$, ${}_\star w_i$ represents all the objects placed inside the i-th agent and ${}_\star w_E \in (A - \{e\})^\circ$ represents all the objects in the environment different from the object e.

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

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

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

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

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

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

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

A transition from a configuration to another one is denoted as

$$({}_\star w_1, \ldots, {}_\star w_n, {}_\star w_E) \Rightarrow ({}_\star w'_1, \ldots, {}_\star w'_n, {}_\star w'_E)\,,$$

where the following conditions are satisfied:

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

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

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

Union and " $-$ " are the multiset operations here.

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 a result of the computation can be associated. 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

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

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

Given a P colony $\Pi = (A,\ e,\ f,\ {}_\star v_E,\ B_1,\ \ldots,\ B_n)$ the maximal number of programs associated with the agents is called the height, the number of agents, n, is called the degree and the number of the objects inside each of the agents is the capacity of the P colony.

Let us use the following notations: $NPCOL_{par}(c, n, h)$ 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 c,
- the degree at most n and
- the height at most h.

If the checking rules are allowed the family of all sets of numbers computed by P colonies is denoted by $NPCOL_{par}K$. If the P colonies are restricted, we use the notation $NPCOL_{par}R$ and $NPCOL_{par}KR$. If the P colonies are homogeneous, we use notation $NPCOL_{par}H$ and $NPCOL_{par}KH$.

2.2 Register Machines

In this paper we compare the families $NPCOL_{par}(c, n, h)$ with the recursively enumerable sets of numbers. To do this 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.

$l_h : HALT$ Halt the machine. The final label l_h is only assigned to this instruction.

Without loss of generality, one can assume that in each ADD-instruction $l_1 : (ADD(r), l_2, l_3)$ and in each SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct.

The register machine M computes a set $N(M)$ of numbers in the following way: it starts with all registers empty (hence storing the number zero) with the instruction labeled l_0 and it proceeds to apply the instructions as indicated by the labels (and made possible by the contents of registers). If it reaches the halt instruction, then the number stored at that time in the register 1 is said to be computed by M and hence it is introduced in $N(M)$. (Because of the nondeterminism in choosing the continuation of the computation in the case of ADD-instructions, $N(M)$ can be an infinite set.) It is known (see e.g. [9]) that in this way we compute all Turing computable sets.

3 P COLONIES WITH ONE OBJECT INSIDE THE AGENT

In this section we analyze the behaviour of P colonies with only one object inside each agent "living" in this P colony. It means that every program is formed by only one rule. This rule is rewriting, communication or checking.

Theorem 1. $NPCOL_{par}K(1, *, 6) = NRE$.

Proof. Let us consider a register machine $M = (m, H, l_0, l_h, P)$. All the labels from H will be objects of the P colony which we construct below. The contents of a register i will be represented by the number of copies of a specific object a_i in the environment. We will construct a P colony $\Pi = (A, f, e, B_1, \ldots, B_n)$ with:

- the alphabet $A = H \cup \{a_i \mid 1 \le i \le m\} \cup \{F_i \mid 1 \le i \le |H|\} \cup \{e, d, D\}$
- final object $f = a_1$
- agent $B_i = ({}_\star e, P_i),\ 1 \le i \le |H| + 3$, and its programs are as follows:

1. We consider the starting agents $B_1,\ B_2$ with a set of programs:

P_1 :		P_2 :	
1 :	$\langle e \to l_0 \rangle$	1 :	$\langle e \to D \rangle$
2 :	$\langle l_0 \leftrightarrow D / l_0 \leftrightarrow e \rangle$	2 :	$\langle D \leftrightarrow l_0 \rangle$

The agent B_1 generates two copies of initial label l_0 of the register machine M and stops by consuming one copy of the object D. The second agent B_2 generates one copy of D and it is blocked with object l_0. Simulation of the computation can start with the second copy of l_0 in the environment.

2. We need one more agent to generate a special object d.

P_3 :	
1 :	$\langle e \to d \rangle$
2 :	$\langle d \leftrightarrow H/d \leftrightarrow e \rangle$

   In every two steps the agent B_3 places one copy of d to the environment.

3. For each instruction $l_1 : (ADD(r), l_2, l_3)$ there is one agent in P colony Π. This agent has to add one copy of the object a_r and the object l_2 or l_3 to the environment.

P_{l_1} :					
1 :	$\langle e \leftrightarrow l_1 \rangle$	3 :	$\langle a_r \leftrightarrow d \rangle$	5 :	$\langle d \to l_3 \rangle$
2 :	$\langle l_1 \to a_r \rangle$	4 :	$\langle d \to l_2 \rangle$	6 :	$\langle l_2 \leftrightarrow e/l_3 \leftrightarrow e \rangle$

   If the object l_1 is present in the environment, the agent B_{l_1} can start to be active, it can consume the object l_1, generate the object a_r, place it to the environment and finally exchange the object l_2 or l_3 by e. At the end of this part of the computation the object with the label of the next instruction of M is placed in the environment and another agent can start to work.

4. For each instruction $l_1 : (SUB(r), l_2, l_3)$ from P we consider the agent B_{l_1} with the set of programs:

P_{l_1} :					
1 :	$\langle e \leftrightarrow l_1 \rangle$	3 :	$\langle F_1 \leftrightarrow a_r \ / \ F_1 \leftrightarrow d \rangle$	5 :	$\langle l_2 \leftrightarrow e/l_3 \leftrightarrow e \rangle$
2 :	$\langle l_1 \to F_1 \rangle$	4 :	$\langle a_r \to l_2/d \to l_3 \rangle$		

   The agent brings inside the object l_1 again and changes it to another object F_1. In the next step the agent checks whether at least one copy of a_r is present in the environment. If so, the agent consumes a_r inside itself and rewrites it to the object l_2; otherwise the agent consumes the object d and rewrites it to the object l_3. In the last step the agent again exchanges the object l_2 or l_3 by e.

5. For the halting instruction labelled l_h we consider the agent B_{l_h} with the following set of programs:

P_{l_h} :			
1 :	$\langle e \leftrightarrow l_h \rangle$	3 :	$\langle H \leftrightarrow d \rangle$
2 :	$\langle l_h \to H \rangle$		

   The agent consumes the object l_h and there is no other object l_m in the environment. This agent places one copy of the object H to the environment and stops working. In the next step the object H is consumed by the agent B_3. No agent can start its work and the computation halts.

From the previous explanations, it is easy to see that 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 zeroes 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 labelled l_h. Consequently, $N(M) = N(\Pi)$, and because each agent contains at most six programs, the proof is complete. □

Another question is how many agents are necessary to simulate any register machine. In [2] the next theorem is proved:

Theorem 2. $NPCOL_{par}K(1, 4, *) = NRE$.

4 P COLONIES WITH TWO OBJECTS INSIDE AGENTS

In the case of agents with two objects each program consists of two rules. If the rules are of the same type in a program the P colony is homogeneous.

Theorem 3. $NPCOL_{par}HK(2, 1, *) = NRE$.

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

- $A = \{d, a, s, f, h, v\} \cup \{l, l' \mid l \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$,
- $f = a_1$,
- $B = (_\star ee, P)$.

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

1. For initializing of the simulation:

P :

1 :	$\langle ee \to dd\rangle$	4 :	$\langle sa \leftrightarrow ed\rangle$	7 :	$\langle se \to fg\rangle$	10 :	$\langle al_0 \leftrightarrow ge\rangle$
2 :	$\langle dd \leftrightarrow ee\rangle$	5 :	$\langle ed \to ha\rangle$	8 :	$\langle fg \leftrightarrow ae\rangle$	11 :	$\langle ge \leftrightarrow hl_0\rangle$
3 :	$\langle dd \to sa\rangle$	6 :	$\langle ha \leftrightarrow se\rangle$	9 :	$\langle ae \to al_0\rangle$	12 :	$\langle hl_0 \leftrightarrow aa\rangle$
13.	$\langle sa \to sa\rangle$						

Agent with two copies of object a inside is prepared to simulate the instruction labelled by l_i (with object l_i placed in the environment). This will be achieved in following steps: The agent starts computation with generating of objects d. For future steps of computation it has to generate four objects d. The second couple of objects d can be rewritten to auxiliary objects s and a. The program 13 ensures endless computation if the number of copies of object d is not sufficient. In the next steps the agent generates second object a, object h and some other auxiliary symbols (not to mix up steps in a computation) and finally the label l_0. If there are two copies of object a inside the agent, the agent is prepared to simulate the instruction labelled by l_i (if the object l_i is placed in the environment). The initialization is done by the following sequence of steps:

	configuration of Π		labels of applicable programs
step	B	Env	P
1.	$_\star ee$		1
2.	$_\star dd$		**2** or 3
3.	$_\star ee$	$_\star dd$	1
4.	$_\star dd$	$_\star dd$	2 or **3**
5.	$_\star sa$	$_\star dd$	**4** or 13
6.	$_\star ed$	$_\star sad$	5
7.	$_\star ha$	$_\star sad$	6
8.	$_\star se$	$_\star haad$	7
9.	$_\star fg$	$_\star haad$	8
10.	$_\star ae$	$_\star fghad$	9
11.	$_\star al_0$	$_\star fghad$	10
12.	$_\star ge$	$_\star l_0fhaad$	11
13.	$_\star hl_0$	$_\star gfaad$	12
14.	$_\star aa$	$_\star l_0gfd$	?

If the agent uses program 3 in the second step it has to execute program 13 in the next steps and the computation never ends. If there are more than one applicable programs, the agent chooses the bold one and executes it.

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

P :							
14 :	$\langle aa \leftrightarrow l_1e\rangle$	17 :	$\langle l_2'a_r \leftrightarrow ef\rangle$	20 :	$\langle ef \leftrightarrow el_3'\rangle$	23 :	$\langle l_2v \leftrightarrow aa\rangle$
15 :	$\langle el_1 \to l_2'a_r\rangle$	18 :	$\langle l_3'a_r \leftrightarrow ef\rangle$	21 :	$\langle el_2' \to l_2v\rangle$	24 :	$\langle l_3v \leftrightarrow aa\rangle$
16 :	$\langle el_1 \to l_3'a_r\rangle$	19 :	$\langle ef \leftrightarrow el_2'\rangle$	22 :	$\langle el_3' \to l_3v\rangle$		

When the agent takes objects l_1 and e inside, it rewrites them to one copy of a_r and the object l_2' or l_3'. The next sequence of steps finishes by generating l_2 or l_3. This object must be sent out to the environment with object v.

	configuration of Π		labels of applicable programs
step	B	Env	P
1.	$_\star aa$	$_\star l_1fghd$	14
2.	$_\star l_1e$	$_\star fghdaa$	**15** or 16
3.	$_\star l_2'a_r$	$_\star fghdaa$	17
4.	$_\star ef$	$_\star l_2'ghdaaa_r$	19
5.	$_\star l_2'e$	$_\star fghdaaa_r$	21
6.	$_\star l_2v$	$_\star fghdaaa_r$	23
7.	$_\star aa$	$_\star l_2vfghdaaa_r$	?

3. For every SUB-instruction $l_1 : (SUB(r), l_2, l_3)$ there is a subset of programs:

P :	
$25: \langle a \leftrightarrow l_1 \ / \ a \leftrightarrow l_1; a \leftrightarrow a_r \ / \ a \leftrightarrow e\rangle$	$28: \langle l_2v \leftrightarrow aa\rangle$
$26: \langle l_1a_r \rightarrow l_2v\rangle$	$29: \langle l_3v \leftrightarrow aa\rangle$
$27: \langle l_1e \rightarrow l_3v\rangle$	

At the first step the agent checks if there is any copy of a_r on the environment (if register r is nonempty). In the positive case it brings l_1 with a_r inside, in the negative case l_1 enters the agent with symbol e. In dependence on the content of the agent, it generates the object l_2 or l_3.

The computation in the case when the register r is empty:

	configuration of Π		labels of applicable programs
step	B	Env	P
1.	$_\star aa$	$_\star l_1fghd$	25
2.	$_\star l_1e$	$_\star fghdaa$	27
3.	$_\star l_3v$	$_\star fghdaa$	29
4.	$_\star aa$	$_\star l_3vfghd$	?

The computation for the case when the register r is not empty:

	configuration of Π		labels of applicable programs
step	B	Env	P
1.	$_\star aa$	$_\star l_1fghda_r^n$	25
2.	$_\star l_1a_r$	$_\star fghdaaa_r^{n-1}$	26
3.	$_\star l_2v$	$_\star fghdaaa_r^{n-1}$	28
4.	$_\star aa$	$_\star l_2vfghda_r^{n-1}$	?

4. For the halting instruction l_h the following program is considered:

P :
$\langle aa \leftrightarrow hl_h\rangle$.

By using this program, the P colony finishes computation as well as the register machine halts its computation.

P colony Π correctly simulates any computation of the register machine M and the number contained in the first register of M corresponds to the number of copies of the object a_1 presented in the environment of Π. □

Theorem 4. $NPCOL_{par}HK(2, *, 4) = NRE$.

Proof. Let us consider a register machine $M = (m, H, l_0, l_h, P)$. All the labels from H will be objects from P colony which we construct below. The contents of a register i will be represented by the number of copies of a specific object a_i in the environment. We will construct a P colony $\Pi = (A, f, e, B_1, \dots, B_n)$ with:

- the alphabet $A = H \cup \{a_i \mid 1 \leq i \leq m\} \cup \{F_i \mid 1 \leq i \leq |H|\} \cup \{e, d, D\}$
- final object $f = a_1$

- agent $B_i = ({}_\star ee, P_i),\ 1 \le i \le |H| + 2$, and its programs are as follows:

1. We consider the starting agents $B_1,\ B_2$ with a set of programs:

P_1 :		P_2 :	
1 :	$\langle ee \to el_0 \rangle$	1 :	$\langle ee \to De \rangle$
2 :	$\langle e \leftrightarrow e/e \leftrightarrow e; l_0 \leftrightarrow D/l_0 \leftrightarrow e \rangle$	2 :	$\langle De \leftrightarrow el_0 \rangle$

The agent B_1 generates two initial labels of the register machine M and stops by consuming one copy of the object D. The second agent B_2 generates one copy of D and waits for the object l_0. After having transported it inside the agent finishes its work. Simulation of the computation can start with the second copy of l_0 in the environment. The beginning of computation can be made in the following way:

	configuration of Π			labels of applicable programs	
step	B_1	B_2	Env	P_1	P_2
1.	${}_\star ee$	${}_\star ee$		1	1
2.	${}_\star el_0$	${}_\star De$		2	$---$
3.	${}_\star ee$	${}_\star De$	${}_\star el_0$	1	2
4.	${}_\star el_0$	${}_\star el_0$	${}_\star De$	2	$---$
5.	${}_\star De$	${}_\star el_0$	${}_\star el_0$	$---$	$---$

2. For each instruction $l_1 : (ADD(r), l_2, l_3)$ there is one agent in P colony Π. This agent has to add one copy of the object a_r and the object l_2 or l_3 to the environment.

P_{l_1} :			
1 :	$\langle ee \leftrightarrow el_1 \rangle$	3 :	$\langle el_1 \to a_r l_3 \rangle$
2 :	$\langle el_1 \to a_r l_2 \rangle$	4 :	$\langle a_r \leftrightarrow e/a_r \leftrightarrow e; l_2 \leftrightarrow e/l_3 \leftrightarrow e \rangle$

If the object l_1 is present in the environment, the agent B_{l_1} can start to be active, it can consume the object l_1, generate the object a_r and the object l_2 or l_3. At the end of this part of the computation the object with the label of the next instruction of M is placed in the environment and another agent can start to work.

	configuration of Π		labels of applicable programs
step	B_{l_1}	Env	P_{l_1}
1.	${}_\star ee$	${}_\star l_1$	1
2.	${}_\star el_1$		**2** *or* 3
3.	${}_\star a_r l_2$		4
4.	${}_\star ee$	${}_\star a_r l_2$	?

3. For each instruction $l_1 : (SUB(r), l_2, l_3)$ from P we consider the agent B_{l_1} with the set of programs:

P_{l_1} :			
1 :	$\langle e \leftrightarrow l_1/e \leftrightarrow l_1; e \leftrightarrow a_r/e \leftrightarrow e \rangle$	3 :	$\langle l_2 \leftrightarrow e/l_3 \leftrightarrow e; v \leftrightarrow e/v \leftrightarrow e \rangle$
2 :	$\langle a_r \to l_2/e \to l_3; l_1 \to v/l_1 \to v \rangle$		

Again, the agent brings inside the object l_1 and one copy of a_r (if there is some a_r in the environment). In the positive case the agent generates the object l_2. In the negative case the agent generates the object l_3. In the last step the agent again exchanges the object l_2 or l_3 by e.

The computation for the case
when the register r is not empty:

step	configuration of Π B_{l_1}	Env	labels of applicable programs P_{l_1}
1.	$_\star ee$	$_\star l_1 a_r$	1
2.	$_\star a_r l_1$		2
3.	$_\star l_2 v$		3
4.	$_\star ee$	$_\star l_2 v$	?

The computation in the case when the register r is empty:

step	configuration of Π B_{l_1}	Env	labels of applicable programs P_{l_1}
1.	$_\star ee$	$_\star l_1$	1
2.	$_\star el_1$		2
3.	$_\star el_3$		3
4.	$_\star ee$	$_\star l_3 v$	?

4. For the halting instruction labelled l_h there is no program in any agent of P colony.

5. The second possible sequence of steps at the beginning of computation is as follows:

	configuration of Π				labels of applicable programs		
step	B_1	B_2	B_{l_0}	Env	P_1	P_2	P_{l_0}
1.	$_\star ee$	$_\star ee$	$_\star ee$		1	1	$---$
2.	$_\star el_0$	$_\star De$	$_\star ee$		2	$---$	$---$
3.	$_\star ee$	$_\star De$	$_\star ee$	$_\star el_0$	1	$---$	1
4.	$_\star el_0$	$_\star De$	$_\star el_0$		2	$---$	2
5.	$_\star ee$	$_\star De$	???	$_\star el_0$	1	2	
6.	$_\star De$	$_\star el_0$	???		$---$	$---$	

It follows from the previous explanations that 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 zeroes 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 labelled l_h. Consequently, $N(M) = N(\Pi)$, and because each agent contains at most five programs, the proof is complete. □

5 CONCLUSIONS

Homogeneous P colonies are computationally complete for:

1. $c = 1$, $h = 6$ and unlimited n (P colonies with one object inside each agent, which uses at most six programs)
2. $c = 1$, $n = 4$ and unlimited h (P colonies composed of four agents, each of them with one object inside the agent)
3. $c = 2$, $h = 4$ and unlimited n (P colonies with two object inside each agent, which uses at most four programs)
4. $c = 2$, $n = 1$ and unlimited h (P colonies with one agent which processes two symbols).

The results were obtained by simulating the behaviour of register machines. In this approach simulation of the ADD operation determines the obtained results.

Acknowledgement

The work of the first two authors 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 1st International workshop WFM '06 (Kolář, D., Meduna, A., eds.), Ostrava, 2006, pp. 27–34.

[2] Cienciala, L.—Ciencialová, L.—Kelemenová, A.: On the Number of Agents in P Colonies, Membrane Computing. Proc. Intern. Workshop, WMC 2007, Thessaloniki, Greece, June 2007 (G. Eleftherakis et al., eds.), LNCS 4860, Springer, Berlin, 2007, pp. 193–208.

[3] Csuhaj-Varjú, E.—Kelemen, J.—Kelemenová, A.—Păun, Gh.—Vaszil, G.: Cells in Environment: P Colonies. Journal of Multiple-valued Logic and Soft Computing, Vol. 12, 2–6, Nos. 3–4, 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] Kelemen, J.—Kelemenová, A.: A Grammar-Theoretic Treatment of Multi-Agent Systems. Cybernetics and Systems. Vol. 23, 1992, pp. 621–633.

[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, Englewood Cliffs, NJ, 1967.

[10] Păun, Gh.: Computing with Membranes. Journal of Computer and System Sciences, Vol. 61, 2000, pp. 108–143.

[11] Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag, Berlin, 2002.

[12] P Systems Web Page. 15 Jan. 2001. `http://psystems.disco.unimib.it`, 7 Sept. 2007.

Luděk Cienciala works at Institute of Computer Science, Silesian University in Opava. His main areas of research are in computer science (membrane computing) and computational graphics.

Lucie Ciencialová is finishing her Ph. D. studies at Institute of Computer Science, Silesian University in Opava. Her interests include teoretical informatics and natural computing.

Alica Kelemenová is an associated professor at Institute of Computer Science, Silesian University in Opava and Department of Computer Science, Catholic University in Ružomberok. Her main research interest is in theoretical computer science, especially formal language theory and biologically motivated generative devices like L systems and P systems.