Nat Comput (2018) 17:321–332
https://doi.org/10.1007/s11047-016-9591-0

Some new results of P colonies with bounded parameters

Luděk Cienciala[1] · **Lucie Ciencialová**[1]

Published online: 23 November 2016

Abstract P colonies were introduced in 2004 as a type of abstract computing device evolved from membrane systems—a biologically motivated computational massive parallel model. A P colony is composed of independent one-membrane agents, reactively acting and evolving in a shared environment. In this paper we summarize the results of computational power obtained for P colonies with bounded number of agents and programs; we reduce these parameters and we also add new results for so-called homogeneous P colonies with capacity two and one.

Keywords P colonies · Computational power · Register machine

1 Introduction

P colonies were introduced in Kelemen et al. (2004) as formal models of a computing device inspired by membrane systems (Păun (2000)) and by colonies, a model from the area of grammar systems theory (Kelemen and Kelemenová (1992)). This model intends to structure and functionality of a community of living organisms in a shared environment. The independent organisms living in a P colony are called agents. The agent is given by a collection of objects embedded in a membrane. Each agent contains the same number of objects. The environment contains several copies of a basic environmental object denoted by e. The number of the copies of e placed in the environment is sufficient for every computation.

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

The computation starts in the initial configuration. Using their programs the agents can change their objects and possibly objects in the environment. This gives possibility to affect the behaviour of the other agents in the next steps of computation. At each step of the computation, each agent with at least one applicable program non-deterministically 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.

There are several different ways in which the initial state of the computation can be defined.

(1) At the beginning of computation the environment and all agents contain only copies of object e.
(2) All the agents can contain various objects at the beginning of computation—the agents are in different

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science - LQ1602", by SGS/24/2013 and SGS/6/2014.

✉ Lucie Ciencialová
lucie.ciencialova@fpf.slu.cz

Luděk Cienciala
ludek.cienciala@fpf.slu.cz

[1] Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Opava, Czech Republic

initial states. The environment contains only copies of object e.
(3) Only environment can contain objects different from the object e.

P colonies were studied in conjunction with three parameters:

(1) the number of objects inside the agent – the capacity of a P colony
(2) the number of agents in a P colony – the degree of a P colony
(3) the maximal number of programs associated with one agent – the height of a P colony.

In the following results the number of necessary agents or the necessary programs stays unbounded to reach computational completeness.

In Freund and Oswald (2005), Kelemen et al. (2004), the authors studied P colonies with two objects inside agents. In this case programs consist of two rules, one for each object. If the former of these rules is an evolution and the latter is a communication or checking, we speak of restricted P colonies. If another combination of the types of the rules is also used, we obtain non-restricted P colonies. The restricted P colonies with the checking rules are computationally complete Freund and Oswald (2005).

In Csuhaj-Varjú et al. (2006b), the authors used P colonies with the third type of initial configuration to simulate small universal register machines introduced in Korec (1996) to bound all parameters in computationally complete classes of P colonies. We continue their work in finding "optimized" computationally complete classes of P colonies with all bounded parameters.

We start with definitions in Sect. 2. In Sect. 3 we will deal with P colonies using checking programs with two objects inside each agent. P colonies with programs consisting of two rules of the same type without use of checking programs are studied in Sect. 4.

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, please refer to Păun (2001). We briefly summarize notations 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^*.

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

2.1 P colonies

We briefly recall the notion of P colonies introduced in Kelemen et al. (2004). 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 the evolution rules, is of the form $a \to b$, which means that object a inside of the agent is rewritten (evolved) to the object b. The second type of rules, called a communication, is in the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and the object d outside of the agent change their positions after execution of the rule object d appears inside the agent and c is placed outside in the environment.

In Kelemen et al. (2004) the ability of agents was extended by checking rule. This rule gives to the agents an opportunity to opt between two possibilities. It has form r_1/r_2. If the checking rule is performed, the rule r_1 has higher priority to be executed than the rule r_2. It means that the agent checks the possibility to use rule r_1. If it can be executed, the agent has to use it. If the rule r_1 cannot be applied, the agent uses the rule r_2.

Definition 1 A P colony of the capacity c is a construct

$$\Pi = (A, e, f, v_E, B_1, \ldots, B_n),$$

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, $f \in A$,
- v_E is a multiset over $A - \{e\}$,
- B_i, $1 \le i \le n$, 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 agent B_i and $|o_i| = c$,
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where each program contains exactly c rules, which

are of one of the following forms: (1) evolution rule $a \to b$, (2) communication rule $c \leftrightarrow d$ and (3) checking rule r_1/r_2; where r_1, r_2 are evolution or communication rules.

The initial configuration of a 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 $1 \leq i \leq n$ and by the set v_E. Formally, the configuration of the P colony Π is given by $(w_1, \ldots, w_n, w_E)$, where $|w_i| = c,\ 1 \leq i \leq n,\ w_i$ 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 object e.

At each step of the computation, the contents of the environment and of the agents change in the following manner: In the maximally parallel derivation mode, each agent which can use any of its programs should use one (non-deterministically chosen), whereas in the sequential derivation mode, one agent (non-deterministically chosen from the set of agents with at least one applicable program) uses one of its programs at a time. If the number of applicable programs for the chosen agent is higher than one, then the agent non-deterministically chooses one of the programs.

A sequence of transitions is called a computation. A computation is said to be halting, if a configuration is reached where no program can be applied any more. With a halting computation we associate a result which is given as the number of copies of the objects f present in the environment in the halting configuration.

Because of the non-determinism in choosing the programs, starting from the initial configuration we obtain several computations, hence, with a P colony we can associate a set of numbers, denoted by $N(\Pi)$, computed by all possible halting computations of the given P colony.

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 of the agents.

If the programs are composed of one rewriting and one communication (or checking) rule in the case of P colony with capacity two, we call such P colony restricted. Restricted program is of one of following forms: $<a \to b, c \leftrightarrow d>$ and $<a \to b, c \leftrightarrow d/f \leftrightarrow g>$.

Let us use the following notations:

$NPCOL_{par}(c, n, h)$ for the family of all sets of numbers computed by these 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 notate as $NPCOL_{par}R$ and $NPCOL_{par}KR$, respectively.

2.2 Register machines

In what follows we want to examine the computational power of P colonies. We compare the families of sets of natural numbers computed by P colonies with the recursively enumerable sets of numbers. To achieve this aim we use the notion of a register machine.

Definition 2 Minsky (1967), Korec (1996) 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 labelled 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 (labelled with) l_2 or l_3.
- $l_1 : (SUB(r), l_2)$ – If the register r stores the value different from zero, then subtract 1 from its content, otherwise leave it unchanged and go to the instruction labelled l_2.
- $l_1 : (CHECK(r), l_2, l_3)$ – If the value stored in register r is zero, go to the instruction labelled l_2, otherwise go to instruction labelled l_3.
- $l_1 : (CHECKSUB(r), l_2, l_3)$ – If register r is non-empty, then subtract 1 from its content and go to the instruction labelled l_2, otherwise go to instruction labelled l_3.
- $l_h : HALT$– Halt the machine. The final label l_h is only assigned to this instruction.

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 labelled 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 non-determinism 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.Minsky (1967)) that in this way register machines using ADD,

CHECKSUB and *HALT* instructions compute all Turing computable sets.

In Korec (1996) the several results on small universal register machines are presented. In this framework the register machines are used to compute result of the function of non-negative integers by having this argument of the function stored in one of the registers at the beginning of computation and the result can be found in other register after halting computation. The universal machines have eight registers and they can simulate computation of register machine M with the information stored as a natural number $code(M)$ coding the particular machine M. The $code(M)$ is placed in the second register.

Theorem 1 Korec (1996) *Let $\mathbb{M}$ be the set of register machines. Then, there are register machines U_1, U_2, U_3 with eight registers and a recursive function $g : \mathbb{M} \to N$ such that for each $M \in \mathbb{M}, N(M) = N(U_i(g(M)))$, where $N(U_i(g(M)))$ denotes the set of numbers computed by $U_i, 1 \leq i \leq 3$, with initially containing $g(M)$ in the second register. All these machines have one HALT instruction labelled by l_h, one instruction of the type ADD labelled l_0, and:*

- U_1 has $8 + 11 + 13 = 32$ instructions of the type ADD, SUB and $CHECK$, respectively,
- U_2 has $9 + 13 = 22$ instructions of the type ADD and $CHECKSUB$, respectively,
- U_3 has $8 + 1 + 12 = 21$ instructions of the type $ADD, CHECK$ and $CHECKSUB$, respectively.

Moreover, these machines either halt using HALT instruction and have the result of the computation in the first register, or their computation goes on infinitely.

The Theorem 1 is reformulated to meet notations and definitions by Minsky (1967).

3 Using checking rules in P colonies with capacity two

We open this section with list of results for classes of P colonies with capacity two. The reader can find them in literature described below.

1. $NPCOL_{par}KR(2, *, 5) = NRE$ in Csuhaj-Varjú et al. (2006a), Kelemen et al. (2004),
2. $NPCOL_{par}R(2, *, 5) = NRE$ in Freund and Oswald (2005),
3. $NPCOL_{par}KR(2, 1, *) = NRE$ in Freund and Oswald (2005),
4. $NPCOL_{par}R(2, 2, *) = NRE$ in Cienciala et al. (2007),
5. $NPCOL_{par}KR\ (2, 23, 5) = NPCOL_{par}KR\ (2, 22, 6) = NRE$ in Csuhaj-Varjú et al. (2006b),
6. $NPCOL_{par}K(2, 22, 5) = NRE$ in Csuhaj-Varjú et al. (2006b),
7. $NPCOL_{par}(2, 35, 8) = NPCOL_{par}R(2, 57, 8) = NRE$ in Csuhaj-Varjú et al. (2006b).

The results 1.–4. do not allow nonempty environment in the initial configuration. In this paper the results allow nonempty environment in the initial configuration. If we sum the programs associated with one agent in the P colony defined in the proof of the result 3. (we can omit the programs for initialization of simulation generating label l_0) we obtain:

$$NPCOL_{par}KR(2, 1, 93) = NRE.$$

The next theorem determines computational power of P colonies working with checking rules.

Theorem 2 $NPCOL_{par}K(2, 1, 66) = NRE.$

Proof Let us consider a register machine M. We construct a P colony $\Pi_1 = (A_1, e, f, v_{E_1}, B_1)$ simulating the computations of the register machine U_2 with 8 registers from Theorem 1 with:

- $A_1 = \{l_i, l_i' \mid l_i \in H\} \cup \{a_m \mid 1 \leq m \leq 8\}$,
- $v_{E_1} = a_2^{g(M)} l_0$,
- $f = a_1$,
- $B_1 = (ee, P_1)$

At the beginning of the computation the agent consumes the object l_0 (the label of starting instruction of U_2) and generates a_r because the first instruction is of the type ADD.

Then it starts to simulate instruction labelled l_0 and it generates the label of the next instruction. The set of programs is as follows:

(1) For the simulation of the initial instruction $l_0 : (ADD(r), l_j, l_k)$ there are programs in P:

$$1: \langle e \leftrightarrow l_0; e \to a_r \rangle, \qquad 2: \langle l_0 \to l_j; a_r \leftrightarrow e \rangle,$$
$$3: \langle l_0 \to l_k; a_r \leftrightarrow e \rangle$$

The initial configuration of Π_1 is $\left(ee, l_0 a_2^m\right), m = g(M)$. After the first step of computation (only the program 1 is applicable) the system enters configuration $\left(l_0 a_r, a_2^m\right)$. Now the second or the third program is applicable and agent uses one of them. After the 2^{nd} step the P colony is in the configuration $\left(ie, a_r a_2^m\right), i \in \{l_j, l_k\}$.

(2) For every ADD-instruction $l_i : (ADD(r), l_j, l_k)$ we add to P the programs:

$$4: \langle l_i \to l_i'; e \to a_r \rangle, \qquad 5: \langle l_i' \to l_j; a_r \leftrightarrow e \rangle,$$
$$6: \langle l_i' \to l_k; a_r \leftrightarrow e \rangle$$

When there is object l_i 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 5 and 6). The sequence of configurations and labels of applicable programs are shown in Table 1.

(3) For every *CHECKSUB*-instruction $l_i : (CHECKSUB(r), l_j, l_k)$, the next programs are added to set P:

$$7: \ \langle l_i \to l_i'; e \leftrightarrow a_r / e \leftrightarrow e \rangle, \qquad 8: \ \langle l_i' \to e; a_r \to l_j \rangle$$
$$9: \ \langle l_i' \to e; e \to l_k \rangle$$

The simulation of the *CHECKSUB* instruction is done in two steps. At the first step agent uses program no. 7 to check whether there is any copy of object a_r in the environment. In positive case it consumes one a_r. The second step is done in accordance to the content (state) of agent. If it contains a_r agent generates object—label l_j, if there is no a_r inside the agent it generates object—label l_k. Instruction $l_i : (CHECKSUB(r), l_j, l_k)$ is simulated by the sequence of steps shown in Table 2.

(4) For *CHECK* instruction we construct three programs similar to previous programs.

$$10: \ \langle l_i \to l_i'; e \leftrightarrow a_r / e \leftrightarrow e \rangle, \qquad 11: \ \langle l_i' \to l_j; a_r \leftrightarrow e \rangle$$
$$12: \ \langle l_i' \to l_k; e \to e \rangle$$

Instruction $l_i : (CHECK(r), l_j, l_k)$ is simulated by the sequence of steps shown in Table 3.

(5) For halting instruction l_h there is no corresponding program in the set P_1.

P colony Π_1 correctly simulates all computations of the register machine U_2 and the number contained on the first register of U_2 corresponds to the number of copies of the object a_1 present in the environment of Π_1. The machine U_2 has one instruction l_0, 8 *ADD*-instructions, 12 *CHECKSUB*-instructions, one *CHECK*-instruction and finally one *HALT*-instruction. Notation $l_0(ADD)$ means that the number bellow it corresponds to the instruction l_0. If we count the programs used for simulation of function of register machine we obtain:

$$h = \overbrace{3}^{l_0(ADD)} + \overbrace{8 \cdot 3}^{ADD} + \overbrace{12 \cdot 3}^{CHECKSUB} + \overbrace{1 \cdot 3}^{CHECK} = 66$$

and the proof is complete. □

Now we add result for restricted P colonies with checking programs.

Table 1 The execution of *ADD*-instruction in P colony Π_1

	B_1	Env_1	P_1
1.	$l_i e$	$a_r^x w$	4
2.	$l_i' a_r$	$a_r^x w$	**5** *or* 6
3.	$l_j e$	$a_r^{x+1} w$	

Table 2 The execution of *CHECKSUB*-instruction in P colony Π_1

	B_1	Env_1	P_1
If the register r stores nonzero value:			
1.	$l_i e$	$a_r^x w$	7
2.	$l_i' a_r$	$a_r^{x-1} w$	8
3.	$l_j e$	$a_r^{x-1} w$	
If the register r stores value zero:			
1.	$l_i e$	w	7
2.	$l_i' e$	w	9
3.	$l_j e$	w	

Table 3 The execution of *CHECK*-instruction in P colony Π_1

	B	Env	P
If the register r stores nonzero value:			
1.	$l_i e$	$a_r^x w$	10
2.	$l_i' a_r$	$a_r^{x-1} w$	11
3.	$l_j e$	$a_r^x w$	
If the register r stores value zero:			
1.	$l_i e$	w	10
2.	$l_i' e$	w	12
3.	$l_k e$	w	

Theorem 3 $NPCOL_{par}KR(2, 1, 74) = NRE$.

Proof Let us consider a register machine M. We construct a P colony $\Pi_2 = (A_2, e, f, v_{E_2}, B_2)$ simulating the computations of register machine U_2 with 8 registers from Theorem 1 with:

- $A_2 = \{e\} \cup \{l_i, l_i' \mid l_i \in H\} \cup \{a_m \mid 1 \le m \le 8\}$,
- $v_{E_2} = a_2^{g(M)} l_0; f = a_1$,
- $B_2 = (ee, P_2)$

The beginning of simulation is very similar to the one in previous theorem.

(1) For the simulation of the initial instruction $l_0 : (ADD(r), l_j, l_k)$ there are programs in P_2:

$$1: \ \langle e \to a_r; e \leftrightarrow l_0 \rangle, \qquad 2: \ \langle l_0 \to l_j; a_r \leftrightarrow e \rangle,$$
$$3: \ \langle l_0 \to l_k; a_r \leftrightarrow e \rangle$$

At the beginning of the computation the agent consumes object l_0 (the label of starting instruction of U_2) and generates a_r because the first instruction is of the type *ADD*. Then it generates the label of the next instruction.

The initial configuration of Π_2 is $(ee, ee, l_0 a_2^m)$, $m = g(M)$. After the first step of computation (only the program 1 is applicable) the system enters configuration $(l_0 a_r, ee, a_2^m)$. Now the second or the third program is

applicable and agent uses one of them. After the second step the P colony is in the configuration $(ie, ee, a_r a_2^m), i \in \{l_j, l_k\}$.

(2) For every *ADD*-instruction $l_i : (ADD(r), l_j, l_k)$ we add to P_2 the programs:

$4: \langle e \to e; l_i \leftrightarrow e\rangle$, $5: \langle e \to a_r; e \leftrightarrow l_i\rangle$,
$6: \langle l_i \to l_j; a_r \leftrightarrow e\rangle$ $7: \langle l_i \to l_k; a_r \leftrightarrow e\rangle$

When there is object l_i inside the agent, it generates one copy of a_r, puts it into the environment and generates the label of the next instruction (it nondeterministically chooses one of the last two programs 6 and 7). The part of the computation is shown in Table 4.

(3) For every *CHECKSUB*-instruction $l_i : (CHECKSUB(r), l_j, l_j)$, the following programs are added to set P_2:

$8: \langle l_i \to l_i'; e \leftrightarrow a_r / e \leftrightarrow e\rangle$, $9: \langle a_r \to l_j; l_i' \leftrightarrow e\rangle$
$10: \langle l_i' \to l_k; e \leftrightarrow e\rangle$

The simulation of the *CHECKSUB* instruction is done in two steps. In the first step agent uses program no. 8 to check whether there is any copy of object a_r in the environment. In positive case it consumes one a_r. The second step is done in accordance to the content (state) of agent. If it contains a_r agent generate objectlabel l_2, if there is no a_r inside the agent it generate object—label l_3. Instruction $l_i : (CHECKSUB(r), l_j, l_k)$ is simulated by the sequence of steps shown in Table 5. $w \in A_2^*$

(4) For *CHECK* instruction $l_i : (CHECK(r), l_j, l_k)$ we construct three programs similar to previous programs.

$11: \langle l_i \to l_i'; e \leftrightarrow a_r / e \leftrightarrow e\rangle$, $12: \langle l_i' \to l_j; a_r \leftrightarrow e\rangle$
$13: \langle l_i' \to l_k; e \leftrightarrow e\rangle$

Instruction $l_i : (CHECK(r), l_j, l_k)$ is simulated by the sequence of steps shown in Table 6.

(5) For halting instruction l_h there is no corresponding program in the set P_2.

P colony Π_2 correctly simulates all computations of the register machine U_2 and the number contained on the first register of U_2 corresponds to the number of copies of the object a_1 present in the environment of Π_2. If we count the programs used for simulation of function of register machine we obtain:

$$h = \overbrace{3}^{l_0(ADD)} + \overbrace{8 \cdot 4}^{ADD} + \overbrace{12 \cdot 3}^{CHECKSUB} + \overbrace{1 \cdot 3}^{CHECK} = 74$$

and the proof is complete. □

Table 4 The execution of *ADD*-instruction in P colony Π_2

	B_2	Env_2	P_2
1.	$l_i e$	$a_r^x w$	4
2.	ee	$l_i a_r^x w$	5
3.	$a_r l_i$	$a_r^x w$	**6** *or* 7
4.	$l_j e$	$a_r^{x+1} w$	

Table 5 The execution of *CHECKSUB*-instruction in P colony Π_2

	B_2	Env_2	P_2
If the register r stores nonzero value:			
1.	$l_i e$	$a_r^x w$	8
2.	$l_i' a_r$	$a_r^{x-1} w$	9
3.	$l_j e$	$a_r^{x-1} l_i' w$	
If the register r stores value zero:			
1.	$l_i e$	w	8
2.	$l_i' e$	w	10
3.	$l_k e$	w	

Table 6 The execution of *CHECK*-instruction in P colony Π_2

	B_2	Env_2	P_2
If the register r stores nonzero value:			
1.	$l_i e$	$a_r^x w$	11
2.	$l_i' a_r$	$a_r^{x-1} w$	12
3.	$l_j e$	$a_r^x w$	
If the register r stores value zero:			
1.	$l_i e$	w	11
2.	$l_i' e$	w	13
3.	$l_k e$	w	

4 Bounded classes of homogeneous P colonies

The program is said to be homogeneous if it is composed of rules of the same type. A P colony having only homogeneous programs is called homogeneous. Each P colony with capacity one that does not use checking rules is homogeneous. Let us summarize results found in papers described below and state the unbounded parameter that we can compute from proofs of the theorems:

- $NPCOL_{par}KH\ (1,*,6) = NPCOL_{par}KH\ (1,26,6) = NRE$ in Cienciala et al. (2008)
- $NPCOL_{par}H(1,4,*) = NPCOL_{par}H(1,4,302) = NRE$ in Cienciala et al. (2007)
- $NPCOL_{par}KH\ (2,*,4) = NPCOL_{par}KH\ (2,25,4) = NRE$ in Cienciala et al. (2008)
- $NPCOL_{par}\ KH(2,1,*) = NPCOL_{par}KH\ (2,1,176) = NRE$ in Cienciala et al. (2008)
- $NPCOL_{par}KH\ (3,2,*) = NPCOL_{par}KH\ (3,2,236) = NRE$ in Cienciala and Ciencialová (2011)

It seems that no result is published related to homogeneous P colonies with capacity two that do not use checking rules and related to P colonies with capacity one using checking programs.

Theorem 4 $NPCOL_{par}H(2,2,163) = NRE$.

Proof Let us consider a register machine M. We construct a P colony $\Pi_3 = (A_3, e, f, v_{E_3}, B_{3_1}, B_{3_2})$ simulating the computations of register machine U_2 with 8 registers from Theorem 1 with:

- $A_3 = \{e, e'\} \cup \{l_i, l_i', l_i''', \overline{l_i} \mid l_i \in H\} \cup \cup \{a_m \mid 1 \le m \le 8\}$,
- $v_{E_3} = a_2^{g(M)} l_0, f = a_1$,
- $B_{3_n} = (ee, P_{3_n}), n = \{1, 2\}$

At the beginning of the computation the agent B_{3_1} consumes the object l_0 (the label of starting instruction of U_2).

(1) For the simulation of the initial instruction $l_0 : (ADD(r), l_j, l_k)$ there are programs in P_1:

1 : $\langle e \leftrightarrow l_0; e \leftrightarrow e \rangle$, 2 : $\langle l_0 \rightarrow l_0'; e \rightarrow a_r \rangle$,
3 : $\langle l_0' \leftrightarrow e; a_r \leftrightarrow e \rangle$, 4 : $\langle e \leftrightarrow l_0'; e \leftrightarrow e \rangle$,
5 : $\langle l_0' \rightarrow l_j; e \rightarrow e \rangle$, 6 : $\langle l_0' \rightarrow l_k; e \rightarrow e \rangle$

The initial configuration of Π_3 is $(ee, ee, l_0 a_2^m), m = g(M)$. Agent B_{3_1} consumes object l_0 and then it starts to simulate instruction labelled l_0. It generates the label of the next instruction. Because each program is homogeneous the agent can only rewrite all its content or exchange both objects inside it for another two objects from the environment. If the content of agent B_{3_1} is ee only programs with communication rules are applicable.

(2) For every *ADD*-instruction $l_i : (ADD(r), l_j, l_k)$ we add to P_{3_1} the programs:

7 : $\langle l_i \rightarrow l_i'; e \rightarrow a_r \rangle$, 8 : $\langle l_i' \leftrightarrow e; a_r \leftrightarrow e \rangle$,
9 : $\langle e \leftrightarrow l_i'; e \leftrightarrow e \rangle$, 10 : $\langle l_i' \rightarrow l_j; e \rightarrow e \rangle$,
11 : $\langle l_i' \rightarrow l_j; e \rightarrow e \rangle$

When there is object l_i 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 10 and 11). The sequence of steps is shown in Table 7.

(3) For every *CHECKSUB*-instruction $l_i : (CHECKSUB(r), l_j, l_k)$, the next programs are added to sets P_{3_1} and P_{3_2}:

P_{3_1}
12 : $\langle l_i \rightarrow l_i'; e \rightarrow l_i'' \rangle$, 13 : $\langle l_i' \leftrightarrow e; l_i'' \leftrightarrow e \rangle$,
14 : $\langle e \leftrightarrow l_i''; e \leftrightarrow a_r \rangle$, 15 : $\langle l_i'' \rightarrow \overline{l_i}; a_r \rightarrow e' \rangle$,
16 : $\langle \overline{l_i} \leftrightarrow e; e' \leftrightarrow e \rangle$, 17 : $\langle e \leftrightarrow \overline{l_i}; e \leftrightarrow e \rangle$,
18 : $\langle \overline{l_i} \rightarrow l_j; e \rightarrow e \rangle$, 19 : $\langle e \leftrightarrow l_i''; e \leftrightarrow l_i''' \rangle$,
20 : $\langle l_i'' \rightarrow l_k; l_i''' \rightarrow e \rangle$

P_{3_2}
21 : $\langle e \leftrightarrow l_i'; e \leftrightarrow e \rangle$, 22 : $\langle l_i' \rightarrow l_i'''; e \rightarrow e \rangle$,
23 : $\langle l_i''' \leftrightarrow e; e \leftrightarrow e \rangle$, 24 : $\langle e \leftrightarrow l_i'''; e \leftrightarrow e' \rangle$,
25 : $\langle l_i''' \rightarrow e; e' \rightarrow e \rangle$

The simulation of the *CHECKSUB* instruction as follows: Agent B_{3_1} puts to object (l_i', l_i'') corresponding to given instruction into the environment; object l_i' is consumed by agent B_{3_2}; at the next step object l_i'' can be consumed by agent B_{3_1} only together with object a_r. If there is no a_r in the environment, agent B_{3_1} has to wait until object l_i''' appears in the environment (agent B_{3_2} generates it). Now program 19 is applicable. The next step is done in accordance to the content (state) of agent B_{3_1}. If it contents a_r agent generates object—label l_2 and puts object l_i' to the environment, if there is no a_r inside the agent it generates object—label l_3. Instruction $l_i : (CHECKSUB(r), l_j, l_k)$ is simulated by the sequence of steps shown in Table 8. Multiset $w \in \{a_m \mid 1 \le m \le 8\}^*$ is placed in the environment.

(4) For *CHECK* instruction we construct three programs similar to previous programs for *CHECKSUB* instruction.

Table 7 The execution of *ADD*-instruction in P colony Π_3

	B_{3_1}	B_{3_2}	Env_3	P_{3_1}	P_{3_2}
1.	$l_i e$	ee	w	7	–
2.	$l_i' a_r$	ee	w	8	–
3.	ee	ee	$l_i' a_r w$	9	–-
4.	$l_i' e$	ee	$a_r w$	**10** *or* 11	–
6.	$l_j e$	ee	$a_r w$	–	–

Table 8 The execution of *CHECKSUB*-instruction in P colony Π_3

	B_{3_1}	B_{3_2}	Env_3	P_{3_1}	P_{3_2}
If the register r stores non-zero value:					
1.	$l_i e$	ee	$a_r^x w$	12	–
2.	$l_i' l_i''$	ee	$a_r^x w$	13	–
3.	ee	ee	$l_i' l_i'' a_r^x w$	14	21
4.	$l_i'' a_r$	$l_i' e$	$a_r^{x-1} w$	15	22
5.	$\overline{l_i} e'$	$l_i''' e$	$a_r^{x-1} w$	16	23
6.	ee	ee	$\overline{l_i} e' l_i''' a_r^{x-1} w$	17	24
7.	$\overline{l_i} e$	$l_i''' e'$	$a_r^{x-1} w$	18	25
8.	$l_j e$	ee	$a_r^{x-1} w$	–	–
If the register r stores value zero:					
1.	$l_i e$	ee	w	12	–
2.	$l_i' l_i''$	ee	w	13	–
3.	ee	ee	$l_i' l_i'' w$	–	21
4.	ee	$l_i' e$	$l_i'' w$	–	22
5.	ee	$l_i''' e$	$l_i'' w$	–	23
6.	ee	ee	$l_i''' l_i'' w$	19	–
7.	$l_i''' l_i''$	ee	w	20	–
8.	$l_k e$	ee	w	–	–

P_{3_1}

$26: \langle l_i \to l_i'; e \to l_i'' \rangle$, $27: \langle l_i' \leftrightarrow e; l_i'' \leftrightarrow e \rangle$,
$28: \langle e \leftrightarrow l_i''; e \leftrightarrow a_r \rangle$, $29: \langle l_i'' \to \overline{l_i}; a_r \to a_r \rangle$,
$30: \langle \overline{l_i} \leftrightarrow e; a_r \leftrightarrow e \rangle$, $31: \langle e \leftrightarrow \overline{l_i}; e \leftrightarrow l_i''' \rangle$,
$32: \langle \overline{l_i} \to l_j; l_i''' \to e \rangle$, $33: \langle e \leftrightarrow l_i''; e \leftrightarrow l_i''' \rangle$,
$34: \langle l_i'' \to l_k; l_i''' \to e \rangle$

P_{3_2}

$35: \langle e \leftrightarrow l_i'; e \leftrightarrow e \rangle$, $36: \langle l_i' \to l_i'''; e \to e \rangle$,
$37: \langle l_i''' \leftrightarrow e; e \leftrightarrow e \rangle$

Instruction $l_i : (CHECK(r), l_2, l_3)$ is simulated by the sequence of steps shown in Table 9.

(5) For halting instruction l_h there is no corresponding program in the set $P_{3_1} \cup P_{3_2}$.

P colony Π_3 correctly simulates all computations of the register machine M and the number contained on the first register of U_2 corresponds to the number of copies of the object a_1 present in the environment of Π_3. If we count the programs used for simulation of function of register machine we obtain:

$$h_1 = \overbrace{6}^{l_0(ADD)} + \overbrace{8 \cdot 5}^{ADD} + \overbrace{12 \cdot 9}^{CHECKSUB} + \overbrace{1 \cdot 9}^{CHECK};$$
$$h_2 = \overbrace{0}^{l_0(ADD)} + \overbrace{8 \cdot 0}^{ADD} + \overbrace{12 \cdot 5}^{CHECKSUB} + \overbrace{1 \cdot 3}^{CHECK};$$
$$h = \max\{h_1, h_2\} = 163$$

and the proof is complete. □

In next result we reduce the number of programs associated with agent.

Theorem 5 $NPCOL_{par}H(2, 92, 3) = NRE$.

Proof Let us consider a register machine M. We construct a P colony $\Pi_4 = (A_4, e, f, v_{E_4}, B_{4_1}, \ldots, B_{4_92})$ simulating the computations of register machine U_2 with 8 registers from Theorem 1 with:

- $A_4 = \{e, e'\} \cup \{l_i, l_i', l_i''', \overline{l_i} \mid l_i \in H\} \cup \cup \{a_m \mid 1 \le m \le 8\}$,
- $v_{E_4} = a_2^{g(M)} l_0, f = a_1$,
- $B_{4_i} = (ee, P_{4_i}), i = \{1, \ldots, 92\}$

Because we want to minimize the number of programs associated with each agent we have to divide simulation of each instruction among more agents. To set order among agents we use the following labelling: $B_{l_i,j}$ implies that this is the j-th agent associated with instruction l_i.

At the beginning of the computation the agent B_1 consumes the object l_0 (the label of starting instruction of U_2).

(1) For the simulation of the initial instruction $l_0 : (ADD(r), l_j, l_k)$ and every ADD-instruction $l_i = (ADD(r), l_j, l_k)$ there are programs in $P_{l_i,p}$, $p = \{1, 2, 3\}$. To simulate the ADD-instruction we need three agents: one agent to generate object a_r and two agents to generate object – label of the next instruction.

$P_{l_i,1}$

$1: \langle e \leftrightarrow l_i; e \leftrightarrow e \rangle$, $2: \langle l_i \to l_i'; e \to a_r \rangle$,
$3: \langle l_i' \leftrightarrow e; a_r \leftrightarrow e \rangle$

$P_{i,2}$

$4: \langle e \leftrightarrow l_i'; e \leftrightarrow e \rangle$, $5: \langle l_i' \to l_2; e \to e \rangle$,
$6: \langle l_2 \leftrightarrow e; e \leftrightarrow e \rangle$

$P_{i,3}$

$7: \langle e \leftrightarrow l_i'; e \leftrightarrow e \rangle$, $8: \langle l_i' \to l_3; e \to e \rangle$,
$9: \langle l_3 \leftrightarrow e; e \leftrightarrow e \rangle$

In Table 11 the reader can find a part of computation—simulation of execution of initial instruction l_0 – sequence of configurations and the labels of used programs.

If the agent $B_{i,3}$ uses the program 7 in the configuration 4, the label l_3 is generated instead of l_2.

(2) For every $CHECKSUB$-instruction $l_i : (CHECKSUB(r), l_j, l_k)$, the next programs are added to sets $P_{l_i,p}, p \in \{1, \ldots, 5\}$:

Table 9 The execution of $CHECK$-instruction in P colony Π_3

	B_{3_1}	B_{3_2}	Env_3	P_{3_1}	P_{3_2}
If the register r stores non-zero value:					
1.	$l_i e$	ee	$a_r^x w$	26	–
2.	$l_i' l_i''$	ee	$a_r^x w$	27	–
3.	ee	ee	$l_i' l_i'' a_r^x w$	28	35
4.	$l_i'' a_r$	$l_i' e$	$a_r^{x-1} w$	29	36
5.	$\overline{l_2} a_r$	$l_i''' e$	$a_r^{x-1} w$	30	37
6.	ee	ee	$\overline{l_2} l_i''' a_r^x w$	31	–
7.	$\overline{l_2} l_i'''$	ee	$a_r^x w$	32	–
8.	$l_j e$	ee	$a_r^x w$	–	–
If the register r stores value zero:					
1.	$l_i e$	ee	w	26	–
2.	$l_i' l_i''$	ee	w	27	–
3.	ee	ee	$l_i' l_i'' w$	–	35
4.	ee	$l_i' e$	$l_i'' w$	–	36
5.	ee	$l_i''' e$	$l_i'' w$	–	37
6.	ee	ee	$l_i''' l_i'' w$	33	–
7.	$l_i''' l_i''$	ee	w	34	–
8.	$l_k e$	ee	w	–	–

$P_{l_i,1}$

10 : $\langle e \leftrightarrow l_i; e \leftrightarrow e\rangle$, 11 : $\langle l_i \rightarrow l_i'; e \rightarrow l_i''\rangle$,

12 : $\langle l_i' \leftrightarrow e; l_i'' \leftrightarrow e\rangle$

$P_{i,2}$

13 : $\langle e \leftrightarrow l_i'; e \leftrightarrow a_r\rangle$, 14 : $\langle l_i' \rightarrow l_j; a_r \rightarrow \overline{l_i}\rangle$,

15 : $\langle l_j \leftrightarrow e; \overline{l_i} \leftrightarrow e\rangle$

$P_{i,3}$

16 : $\langle e \leftrightarrow l_i''; e \leftrightarrow e\rangle$, 17 : $\langle l_i' \rightarrow l_i'''; e \rightarrow e\rangle$,

18 : $\langle l_i''' \leftrightarrow e; e \leftrightarrow e\rangle$

$P_{i,4}$

19 : $\langle e \leftrightarrow l_i'; e \leftrightarrow l_i''''\rangle$, 20 : $\langle l_i' \rightarrow l_k; l_i'''' \rightarrow e\rangle$,

21 : $\langle l_k \leftrightarrow e; e \leftrightarrow e\rangle$

$P_{i,5}$

22 : $\langle e \leftrightarrow l_i''''; e \leftrightarrow \overline{l_i}\rangle$, 23 : $\langle l_i'''' \rightarrow e; \overline{l_i} \rightarrow e\rangle$

The simulation of the *CHECKSUB* instruction as follows: Agent $B_{l_i,1}$ puts objects (l_i', l_i'') corresponding to given instruction to the environment; object l_i' is consumed by agent $B_{l_i,2}$ if there is at least one copy of a_r in the environment agent $B_{l_i,2}$ rewrites objects a_r and l_i' to object corresponding to label of the next instruction and to object $\overline{l_i}$ – the message for agent $B_{l_i,5}$ that the unused object l_i''' must be erased from the environment. The agent $B_{l_i,3}$ consumes object l_i'' and in the next step it rewrites this object to object l_i'''' and in the following step agent puts this object into the environment.

In the case that register r stores value zero, agent $B_{l_1,4}$ consumes objects l_i' and l_i'''' and finally it generates object—label l_k. Instruction $l_i : (CHECKSUB(r), l_j, l_k)$ is simulated by the sequence of steps as present in Table 10. Multiset $w \in \{a_m \mid 1 \leq m \leq 8\}^*$ is placed in the environment.

(3) For *CHECK* instruction we construct three programs similar to programs in previous paragraph. The only change is in programs associated with agent $B_{l_i,5}$.

$P_{i,5}$

22 : $\langle e \leftrightarrow l_i''''; e \leftrightarrow \overline{l_i}\rangle$, 23 : $\langle l_i'''' \rightarrow e; \overline{l_i} \rightarrow a_r\rangle$,

24 : $\langle a_r \leftrightarrow e; e \leftrightarrow e\rangle$

(4) For halting instruction l_h there is no corresponding program in the set P_i, $1 \leq i \leq 92$.

P colony Π_4 correctly simulates all computations of the register machine U_2 and the number contained on the first register of U_2 corresponds to the number of copies of the object a_1 present in the environment of Π_4. If we count the programs used for simulation of function of register machine we obtain:

$$n = \overbrace{1 \cdot 3}^{l_0(ADD)} + \overbrace{8 \cdot 3}^{ADD} + \overbrace{12 \cdot 5}^{CHECKSUB} + \overbrace{1 \cdot 5}^{CHECK} = 92$$

and the proof is complete. □

Table 10 Simulation of execution *CHECKSUB* instruction in P colony Π_4

	$B_{l_i,1}$	$B_{l_i,2}$	$B_{l_i,3}$	$B_{l_i,4}$	$B_{l_i,5}$	Env_4	$P_{l_i,1}$	$P_{l_i,2}$	$P_{l_i,3}$	$P_{l_i,4}$	$P_{l_i,5}$
If the register r stores non-zero value:											
1.	ee	ee	ee	ee	ee	$l_i a_r^x w$	10	–	–	–	–
2.	$l_i e$	ee	ee	ee	ee	$a_r^x w$	11	–	–	–	–
3.	$l_i' l_i''$	ee	ee	ee	ee	$a_r^x w$	12	–	–	–	–
4.	ee	ee	ee	ee	ee	$l_i' l_i'' a_r^x w$	–	13	16	–	–
5.	ee	$l_i' a_r$	$l_i'' e$	ee	ee	$a_r^{x-1} w$	–	14	17	–	–
6.	ee	$l_j \overline{l_i}$	$l_i''' e$	ee	ee	$a_r^{x-1} w$	–	15	18	–	–
7.	ee	ee	ee	ee	ee	$l_j \overline{l_i} l_i''' a_r^{x-1} w$	–	–	–	–	22
8.	ee	ee	ee	ee	$\overline{l_i} l_i'''$	$a_r^{x-1} w$	–	–	–	–	23
9.	ee	ee	ee	ee	ee	$a_r^{x-1} w$	–	–	–	–	–
If the register r stores value zero:											
1.	ee	ee	ee	ee	ee	$l_i w$	10	–	–	–	–
2.	$l_i e$	ee	ee	ee	ee	w	11	–	–	–	–
3.	$l_i' l_i''$	ee	ee	ee	ee	w	12	–	–	–	–
4.	ee	ee	ee	ee	ee	$l_i' l_i'' w$	–	–	16	–	–
5.	ee	ee	$l_i'' e$	ee	ee	$l_i' w$	–	–	17	–	–
6.	ee	ee	$l_i''' e$	ee	ee	$l_i' w$	–	–	18	–	–
7.	ee	ee	ee	ee	ee	$l_i' l_i''' w$	–	–	–	19	–
8.	ee	ee	ee	$l_i' l_i'''$	ee	w	–	–	–	20	–
9.	ee	ee	ee	$l_k e$	ee	w	–	–	–	21	–
10.	ee	ee	ee	ee	ee	$l_k w$	–	–	–	–	–

Theorem 6 $NPCOL_{par}H(2,70,5) = NRE$.

It is very easy to see that we obtain the result by union of agents $B_{l_i,2}$ and $B_{l_i,3}$ constructed in previous proof for *ADD*-instructions.

The last result of this paper is devoted to the homogeneous P colony with capacity one using checking programs.

Theorem 7 $NPCOL_{par}KH(1,3,325) = NRE$.

Proof Let us consider a register machine M. We construct a P colony $\Pi_5 = (A_5, e, f, v_{E_5}, B_{5_1}, B_{5_2}, B_{5_3})$ simulating the computations of register machine U_2 with 8 registers from Theorem 1 with:

- $A_5 = \{e, d, g\} \cup \{l_i, l_i', l_i'', l_i''', \overline{l_i}, \overline{\overline{l_i}}, \underline{l_i}, \underline{\underline{l_i}}, l_i^a, l_i^b, l_i^c, l_i^d, L_i^a, L_i^b, L_i', L_i'' \mid l_i \in H\} \cup \{a_m \mid 1 \le m \le 8\} \cup \{g_k \mid 1 \le k \le 5\}$,
- $v_{E_5} = a_2^{g(M)} l_0 dg, f = a_1$,
- $B_{5_j} = (e, P_{5_j}), j = \{1,2,3\}$

At the beginning of the computation the agent B_{5_1} generates the object l_0' (corresponding to the label of starting instruction of U_2).

(1) For the simulation of the initial instruction $l_0 : (ADD(r), l_j, l_k)$ there are programs in P_{5_1} and in P_{5_2}:

P_{5_1}

1 : $\langle e \to l_0' \rangle$, 2 : $\langle l_0' \leftrightarrow l_0 / l_0' \leftrightarrow d \rangle$, 3 : $\langle l_0 \to \underline{l_0} \rangle$
4 : $\langle \underline{l_0} \to l_j \rangle$, 5 : $\langle \underline{l_0} \to l_k \rangle$, 6 : $\langle d \to d \rangle$,

P_{5_2}

7 : $\langle e \leftrightarrow l_0' \rangle$, 8 : $\langle l_0' \to a_r \rangle$, 9 : $\langle a_r \leftrightarrow e \rangle$

The initial configuration of Π_5 is $(e, e, l_0 a_2^m g d), m = g(M)$. Agent B_{5_1} generates object l_0' and ensures itself by consuming object l_0 from the environment that it generated the right object (corresponding to the label of initial instruction of the machine M). In negative case the agent consumes object d, agent start to rewrite object d in the circle and computation never ends. In positive case agent generates the label of the next instruction. The agent B_{5_2} executes adding one object a_r to the environment. The situation when simulation is done correctly, is shown in Table 12.

The part of computation when the agent B_{5_1} generates in the first step wrong object is shown in Table 13.

(2) For every *ADD*-instruction $l_i : (ADD(r), l_j, l_k)$ we add to P_{5_1} the programs:

P_{5_1}

10 : $\langle l_i \leftrightarrow e \rangle$, 11 : $\langle e \to \underline{l_i} \rangle$, 12 : $\langle \underline{l_i} \to \underline{\underline{l_i}} \rangle$
13 : $\langle \underline{\underline{l_i}} \to l_i' \rangle$, 14 : $\langle l_i' \leftrightarrow l_i'' / l_i' \leftrightarrow d \rangle$, 15 : $\langle l_i'' \to \overline{l_i} \rangle$,
16 : $\langle \overline{l_i} \to l_j \rangle$, 17 : $\langle \overline{l_i} \to l_k \rangle$,

P_{5_2}

18 : $\langle e \leftrightarrow l_i \rangle$, 19 : $\langle l_i \to l_i'' \rangle$, 20 : $\langle l_i'' \leftrightarrow e \rangle$
21 : $\langle e \leftrightarrow l_i' \rangle$, 22 : $\langle l_i' \to a_r \rangle$, 23 : $\langle a_r \leftrightarrow e \rangle$

When there is object l_i inside the agent B_{5_1}, it puts it to the environment and tries to generate object l_i'. Then it exchanges object l_i' by object l_i'' from the environment. Finally agent B_{5_1} generates the label of the next instruction (it non-deterministically chooses one of the last two programs 16 and 17) and the agent B_{5_2} generates one object a_r and puts it to the environment (Table 14).

(3) For every *CHECKSUB*-instruction $l_i : (CHECKSUB(r), l_j, l_k)$, the next programs are added to sets P_{5_1}, P_{5_2} and P_{5_3}:

P_{5_1}

24 : $\langle l_i \leftrightarrow e \rangle$, 25 : $\langle e \to \underline{l_i} \rangle$, 26 : $\langle \underline{l_i} \to \underline{\underline{l_i}} \rangle$,
27 : $\langle \underline{\underline{l_i}} \to l_i' \rangle$, 28 : $\langle l_i' \leftrightarrow l_i'' / l_i' \leftrightarrow d \rangle$, 29 : $\langle l_i'' \to \overline{l_i} \rangle$,
30 : $\langle \overline{l_i} \to \overline{\overline{l_i}} \rangle$, 31 : $\langle \overline{\overline{l_i}} \to L_i \rangle$, 32 : $\langle L_i \leftrightarrow a_r / L_i \leftrightarrow l_i''' \rangle$,
33 : $\langle l_i''' \to l_i^a \rangle$, 34 : $\langle l_i^a \to l_i^b \rangle$, 35 : $\langle l_i^b \to l_i^c \rangle$,
36 : $\langle l_i^c \to l_i^d \rangle$, 37 : $\langle l_i^d \to l_k \rangle$, 38 : $\langle a_r \to L_i^a \rangle$,
39 : $\langle L_i^a \to L_i^b \rangle$, 40 : $\langle L_i^b \to L_i'' \rangle$, 41 : $\langle L_i'' \leftrightarrow L_i' / L_i'' \leftrightarrow d \rangle$,
42 : $\langle L_i' \to l_j \rangle$,

P_{5_2}

43 : $\langle e \leftrightarrow l_i \rangle$, 44 : $\langle l_i \to l_i'' \rangle$, 45 : $\langle l_i'' \leftrightarrow e \rangle$
46 : $\langle e \leftrightarrow l_i' \rangle$, 47 : $\langle l_i' \to l_i''' \rangle$, 48 : $\langle l_i''' \leftrightarrow g \rangle$
49 : $\langle g \to g_1 \rangle$, 50 : $\langle g_1 \to g_2 \rangle$, 51 : $\langle g_2 \to g_3 \rangle$
52 : $\langle g_3 \to g_4 \rangle$, 53 : $\langle g_4 \to g_5 \rangle$, 54 : $\langle g_5 \leftrightarrow L_i'' / g_5 \leftrightarrow L_i'' \rangle$
55 : $\langle L_i'' \to e \rangle$, 56 : $\langle L_i' \to e \rangle$,

P_{5_3}

56 : $\langle e \leftrightarrow L_i \rangle$, 57 : $\langle L_i \to L_i' \rangle$, 58 : $\langle L_i' \leftrightarrow l_i''' / L_i' \leftrightarrow e \rangle$
59 : $\langle l_i''' \to e \rangle$, 60 : $\langle e \leftrightarrow g_5 \rangle$, 61 : $\langle g_5 \to g \rangle$
62 : $\langle g \leftrightarrow e \rangle$

The simulation of the *CHECKSUB* instruction as follows: Agent B_{5_1} puts the object (l_i) corresponding to given instruction into the environment; object l_i is consumed by agent B_{5_2}; Agent B_{5_1} generates object l_i' and exchanges it by object l_i''. Agent B_{5_1} rewrites object l_i'' to L_i. Then agent B_{5_1} tries to consume object a_r. Three agents help each other to generate object corresponding to the label of the next

Table 11 The execution of initial instruction in P colony Π_4

	$B_{i,1}$	$B_{i,2}$	$B_{i,3}$	Env_4	$P_{1,1}$	$P_{i,2}$	$P_{i,3}$
1.	ee	ee	ee	wl_i	1	–	–
2.	$l_i e$	ee	ee	w	2	–	
3.	$l_i' a_r$	ee	ee	w	3	–	
4.	ee	ee	ee	$l_i' a_r w$	–	**4**	or 7
5.	ee	$l_i' e$	ee	$a_r w$	–	5	–
6.	ee	$l_j e$	ee	$a_r w$	–	6	–
7.	ee	ee	ee	$l_j a_r w$	–	–	–

Table 12 The execution of initial instruction in P colony Π_5—the right program was executed

	B_{5_1}	B_{5_2}	Env	P_{5_1}	P_{5_2}
1.	e	e	$a_2^k l_0 dg$	1	–
2.	l'_0	e	$a_2^k l_0 dg$	2	–
3.	l_0	e	$a_2^k l'_0 dg$	3	7
4.	$\underline{l_0}$	l'_0	$a_2^k dg$	**4** *or* 5	8
5.	l_j	a_r	$a_2^k gd$	?	9
6.	?	e	$a_r a_2^k gd$	–	–

Table 13 The execution of initial instruction in P colony Π_5—non-corresponding object was generated

	B_{5_1}	B_{5_2}	Env	P_{5_1}	P_{5_2}
1.	e	e	$a_2^k l_0 dg$	x	–
2.	l'_y	e	$a_2^k l_0 dg$	2	–
3.	d	e	$a_2^k l'_0 g$	6	7
4.	d	l'_0	$a_2^k g$	6	8
5.	d	a_r	$a_2^k g$	6	9
6.	d	e	$a_r a_2^k g$	6	–

Table 14 The execution of *ADD*-instruction in P colony Π_5

	B_{5_1}	B_{5_2}	Env	P_{5_1}	P_{5_2}
1.	l_i	e	w	10	–
2.	e	e	$l_i w$	11	18
3.	$\underline{l_i}$	l_i	w	12	19
4.	$\underline{\underline{l_i}}$	l''_i	w	13	20
5.	l'_i	e	$l''_i w$	14	–
6.	l'''_i	e	$l'_i w$	15	21
7.	$\overline{l_i}$	l'_i	w	**16** *or* 17	22
8.	l_j	a_r	w	?	23
9.	e	e	$a_r w l_j$	–	–

Table 15 The execution of *CHECKSUB*-instruction in P colony Π_5

	B_{5_1}	B_{5_2}	B_{5_3}	Env	P_{5_1}	P_{5_2}	P_{5_3}
If the register r stores non-zero value:							
1.	l_i	e	e	$a_r^x gw$	24	–	–
2.	e	e	e	$l_i a_r^x gw$	25	43	–
3.	$\overline{l_i}$	l_i	e	$a_r^x gw$	26	44	–
4.	$\overline{\overline{l_i}}$	l''_i	e	$a_r^x gw$	27	45	–
5.	l'_i	e	e	$l''_i a_r^x gw$	28	–	–
6.	l''_i	e	e	$l'_i a_r^x gw$	29	46	–
7.	$\overline{l_i}e$	l'_i	e	$a_r^x gw$	30	47	–
8.	$\overline{\overline{l_i}}$	l'''_i	e	$a_r^x gw$	31	48	–
9.	L_i	g	e	$l'''_i a_r^x w$	32	49	–
10.	a_r	g_1	e	$L_i l'''_i a_r^{x-1} w$	38	50	57
11.	L_i^a	g_2	L_i	$l'''_i a_r^{x-1} w$	39	51	58
12.	L_i^b	g_3	L'_i	$l'''_i a_r^{x-1} w$	40	52	59
13.	L''_i	g_4	l'''_i	$L'_i a_r^{x-1} w$	41	53	60
14.	L'_i	g_5	e	$L''_i a_r^{x-1} w$	42	54	–
15.	l_j	L''_i	e	$g_5 a_r^{x-1} w$	?	55	61
16.	e	e	g_5	$l_j a_r^{x-1} w$	?	–	62
17.	?	?	g	$?a_r^{x-1} w$	?	?	63
18.	?	?	e	$?a_r^{x-1} gw$	?	–	–
If the register r stores value zero:							
1.	l_i	e	e	gw	24	–	–
2.	e	e	e	$l_i gw$	25	43	–
3.	$\overline{l_i}$	l_i	e	gw	26	44	–
4.	$\overline{\overline{l_i}}$	l''_i	e	gw	27	45	–
5.	l'_i	e	e	$l''_i gw$	28	–	–
6.	l''_i	e	e	$l'_i gw$	29	46	–
7.	$\overline{l_i}e$	l'_i	e	gw	30	47	–
8.	$\overline{\overline{l_i}}$	l'''_i	e	gw	31	48	–
9.	L_i	g	e	$l'''_i w$	32	49	–
10.	l'''_i	g_1	e	$L_i w$	33	50	57
11.	l_i^a	g_2	L_i	w	34	51	58
12.	l_i^b	g_3	L'_i	w	35	52	59
13.	l_i^c	g_4	e	$L'_i w$	36	53	–
14.	l_i^d	g_5	e	$L'_i w$	37	54	–
15.	l_k	L'_i	e	$g_5 w$	?	56	61
16.	e	e	g_5	$l_k w$	?	–	62
17.	?	?	g	$?w$	?	?	63
18.	?	?	e	$?gw$	?	–	–

instruction. The part of computation the reader can see on Table 15.

(4) For halting instruction l_h there is no corresponding program in the set $P_j, j \in \{1, 2, 3\}$.

P colony Π_5 correctly simulates all computations of the register machine U_2 and the number contained on the first register of U_2 corresponds to the number of copies of the object a_1 present in the environment of Π_5. If we count the programs used for simulation of function of register machine we obtain:

$$h_1 = \overbrace{6}^{l_0(ADD)} + \overbrace{9 \cdot 8}^{ADD} + \overbrace{13 \cdot 19}^{CHECKSUB};$$
$$h_2 = \overbrace{3}^{l_0(ADD)} + \overbrace{9 \cdot 6}^{ADD} + \overbrace{13 \cdot 9 + 5}^{CHECKSUB};$$
$$h_3 = \overbrace{0}^{l_0(ADD)} + \overbrace{9 \cdot 0}^{ADD} + \overbrace{13 \cdot 4 + 3}^{CHECKSUB};$$
$$h = \max\{h_1, h_2, h_3\} = 325$$

and the proof is complete. □

5 Conclusions

In this paper we focused on P colonies with all bounded parameters. In the first part we improve results for P colonies with checking rules. In the second part we focus on homogeneous P colonies with or without use of checking programs. We can summarize our results in following list:

- $NPCOL_{par}K(2, 1, 66) = NRE$
- $NPCOL_{par}KR(2, 1, 74) = NRE$
- $NPCOL_{par}H(2, 2, 163) = NRE$
- $NPCOL_{par}H(2, 92, 3) = NRE$
- $NPCOL_{par}H(2, 70, 5) = NRE$
- $NPCOL_{par}KH(1, 3, 325) = NRE$

For more information on membrane computing, see Păun (2001); for more on computational machines and colonies in particular, see Minsky (1967) and Kelemen and Kelemenová (1992), Kelemen et al. (2004), respectively. Activities carried out in the field of membrane computing are currently numerous and they are available also at P systems web page (2001).

References

(2001) P systems web page. http://ppage.psystems.eu

Cienciala L, Ciencialová L (2011) Computation, cooperation, and life. Springer-Verlag, Berlin, Heidelberg, chap P Colonies and Their Extensions, pp 158–169

Cienciala L, Ciencialová L, Kelemenová A (2007) On the number of agents in P colonies. In: Proceedings of the 8th international conference on membrane computing, Springer-Verlag, Berlin, Heidelberg, WMC'07, pp 193–208

Cienciala L, Ciencialová L, Kelemenová A (2008) Homogeneous P colonies. Comput Inform 27(3+):481–496

Csuhaj-Varjú E, Kelemen J, Kelemenová A (2006a) Computing with cells in environment: P colonies. Mult Valued Logic Soft Comput 12(3–4):201–215

Csuhaj-Varjú E, Margenstern M, Vaszil G (2006b) P colonies with a bounded number of cells and programs. In: Hoogeboom HJ, Paun G, Rozenberg G, Salomaa A (eds) Membrane Computing, 7th international workshop, WMC 2006, Leiden, The Netherlands, July 17–21, 2006, Revised, Selected, and Invited Papers, Springer, Lecture Notes in Computer Science, vol 4361, pp 352–366, doi:10.1007/11963516_22

Freund R, Oswald M (2005) P colonies working in the maximally parallel and in the sequential mode. In: Zaharie D, Petcu D, Negru V, Jebelean T, Ciobanu G, Cicortas A, Abraham A, Paprzycki M (eds) Seventh International symposium on symbolic and numeric algorithms for scientific computing (SYNASC 2005), 25–29 September 2005, Timisoara, Romania, IEEE Computer Society, pp 419–426, doi:10.1109/SYNASC.2005.55

Kelemen J, Kelemenová A (1992) A grammar-theoretic treatment of multiagent systems. Cybern Syst 23(6):621–633. doi:10.1080/01969729208927485

Kelemen J, Kelemenová A, Păun Gh (2004) Preview of P colonies: a biochemically inspired computing model. In: Workshop and tutorial proceedings. Ninth international conference on the simulation and synthesis of living systems (Alife IX), Boston, Massachusetts, USA, pp 82–86

Korec I (1996) Small universal register machines. Theor Comput Sci 168(2):267–301. doi:10.1016/S0304-3975(96)00080-1

Minsky ML (1967) Computation: finite and infinite machines. Prentice-Hall Inc, Upper Saddle River

Păun G (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143. doi:10.1006/jcss.1999.1693

Păun G (2001) Current trends in theoretical computer science. World Scientific Publishing Co., Inc., River Edge, NJ, USA, chap Computing with Membranes (P Systems): An Introduction, pp 845–866