

Models of P Colonies

Lucie Cencialová^{1,*}, Luděk Cenciala^{1,†}

¹*Institute of Computer Science, Faculty of Philosophy and Science in Opava, Silesian University in Opava, Opava, Czech Republic*

Abstract

This paper explores different models of P colonies, including restricted, homogeneous, and those with senders and consumers. P colonies, inspired by the behavior of simple unicellular organisms in a shared environment, are theoretical computational models where agents interact through finite programs within a common environment. The study examines transformations between these P colony types and their impact on new findings related to the computational completeness of P colonies under specific parameter constraints.

Keywords

membrane computing, P Colonies, agent-based model, nature-inspired computation model

1. Introduction

In this paper, we concentrate on various previously published P colony models, specifically the restricted P colony, the homogeneous P colony, and P colonies with senders and consumers. The original model of P colony was introduced in [1] as a theoretical computing model inspired by structure and behavior of simple one-cell organisms living in a shared environment.

The P colony consists of basic units called agents, each equipped with programs. The environment plays a crucial role, storing the products of agent activities and enabling agents to send "messages" to each other through it. The agents operate based on objects.

Each agent contains a finite multiset of objects, which are processed by a finite set of unique programs associated with each agent. The number of objects within each agent remains constant during the computation of the agent community, known as the "capacity" of the P colony. In this paper, we will focus on P colonies with a capacity of 2, specifically on P colonies where each agent contains exactly two objects.

The agents share a common environment, represented by another multiset of objects. Among these objects, a specific type called the "environmental object" is assumed to exist in an infinitely countable number of copies. It is worth noting that some literature may also describe cases where the environmental symbol appears in a very large, but finite, number of copies.

By utilizing their respective programs, the agents can

modify the objects they possess and exchange some of their objects with those in the environment. These coordinated actions result in a configuration change, or transition, within the P colony. A finite sequence of consecutive configuration changes, starting from the initial configuration, constitutes a computation. The output of this computation is determined by counting the number of copies of a specific distinguished object, known as the "final object", present in the environment at the end of the process.

The environment serves a dual purpose: it acts as a communication channel for the agents and also functions as a storage medium for objects. Its critical role lies in synchronizing the collaborative efforts of the agents throughout the entire computation process.

The programs thus allow P colony agents to change both their own contents and the contents of the environment. The programs consist basically of six distinct types of rules: rewriting, communication, checking, generating, consuming and transporting rules. The first two types of rules are used in restricted and homogeneous P colonies. Communication and rewriting rules can be combined to checking rules. In this paper, we do not consider the use of this combination of rules, which determine the priority between two participating rules. P colonies with senders and consumers uses insertion and deletion rules.

The structure of the paper is as follows: after an introductory section, we introduce the basic concepts of the original P colony model, its restricted and homogeneous versions and P colony with senders and consumers. In the third part, we will focus on transformations between these types of P colonies. We will compare the results regarding the computational power of these types of P colonies in terms of the proposed transformations.

ITAT 2024: Information Technologies – Applications and Theory, September 20–24, 2024, Čerčgovské Vrchy, Slovakia

*Corresponding author.

†These authors contributed equally.

✉ lucie.cencialova@fpf.slu.cz (L. Cencialová);

ludek.cenciala@fpf.slu.cz (L. Cenciala)

🆔 0000-0002-0877-7063 (L. Cencialová); 0000-0001-7116-9338

(L. Cenciala)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

2. Preliminaries and Definitions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory and membrane computing [2, 3].

For an alphabet Σ , the set of all words over Σ (including the empty word, ε), is denoted by Σ^* . We denote the length of a 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 $M = (O, f)$, where O is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : O \rightarrow \mathbb{N}$; f assigns to each object in O its multiplicity in M . Any multiset of objects M with the set of objects $O = \{x_1, \dots, x_n\}$ can be represented as a string w over alphabet O with $|w|_{x_i} = f(x_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M , and ε represents the empty multiset.

2.1. P Colonies

In the following we describe the concept of a P Colony. The original definition of P colony was introduced in [1]. In this paper, we will use an extended definition [4], which we have slightly modified by excluding the possibility of an evolving environment.

Rewriting rule $a \rightarrow b$ allows an agent to rewrite (evolve) one object a placed inside the agent to object b .

Communication rule $a \leftrightarrow b$ exchanges one object a placed inside the agent for object b from the environment.

Checking rule r_1/r_2 , where each of r_1, r_2 is a rewriting or a communication rule, sets a priority between these two rules. The agent try to apply the first rule and if it cannot be performed, the agent executes the second rule.

Generating rule $a \rightarrow bc$ creates two objects b, c from one object a .

Consuming rule $ab \rightarrow c$ rewrites two objects a, b to one object c .

Transporting rule of the form (a, in) or (a, out) is used to transport one object from the environment into the agent, or from the agent to the environment, respectively. The rule is always associated with a consuming/generating rule to keep a constant number of objects inside the agent.

Definition 1. A P colony with capacity $c \geq 1$ is the structure

$$\Pi = (\Sigma, e, f, v_E, B_1, \dots, B_n), \text{ where}$$

- Σ is the alphabet of the colony, its elements are called objects,
- e is the basic (environmental) object of the colony, $e \in \Sigma$,
- f is final object of the colony, $f \in \Sigma$,
- v_E is the initial content of the environment, $v_E \in (\Sigma - \{e\})^*$,
- $B_i, 1 \leq i \leq n$, are the agents, every agent is the structure $B_i = (o_i, P_i)$, where o_i is a multiset over Σ , it defines the initial state (content) of the agent B_i and $|o_i| = c$ and $P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is the finite set of programs of three types:

- (1) generating program with generating rules $a \rightarrow bc$ and transporting rules (d, out) - the number of generating rules is the same as the number of transporting rules.
- (2) consuming program with consuming rules $ab \rightarrow c$ and transporting rules (d, in) - the number of consuming rules is the same as the number of transporting rules.
- (3) rewriting/communication program can contain three types of rules:
 - ◊ $a \rightarrow b$, called a rewriting rule,
 - ◊ $c \leftrightarrow d$, called a communication rule,
 - ◊ r_1/r_2 , called a checking rule; each of r_1, r_2 is a rewriting or a communication rule.

We first note that throughout the paper, we use term "object a is inside agent A " and term " $a \in w$, where w is the state of agent A " as equivalent.

The functioning of the P colony starts from its initial configuration (state).

The initial configuration of a P colony is an $(n + 1)$ -tuple of multisets of objects present in the P colony at the beginning of the computation. It is given by the multisets o_i for $1 \leq i \leq n$ and by multiset v_E . Formally, the configuration of the P colony Π is given by (w_1, \dots, w_n, w_E) , where $|w_i| = c, 1 \leq i \leq n$, w_i represents all the objects present inside the i -th agent, and $w_E \in (\Sigma - \{e\})^*$ represents all the objects in the environment different from the object e .

At each step of the computation (at each transition), the state of the environment and that 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 uses one of its programs at a time (non-deterministically chosen). If the number of applicable programs for one 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 given P colony.

In the original model (see [1]) the number of objects inside each agent is set to two. Since the application of a program must involve all objects within the agent, the original P colony model requires that the number of rules in a program equals the number of objects, which is two. Moreover the initial configuration was defined as $(n + 1)$ -tuple $(ee, \dots, ee, \varepsilon)$ so the environment of the P colony is at the beginning of the computation “empty”, without an input information.

A P colony is called *restricted* if each program consists of one evolving rule and one communication rule. A P colony is called *homogeneous* if each program consists of one type of rules - rewriting or communication. The third type of P colonies we will cover in this article are P colonies using generating and consuming programs. If agents have only generating programs in their program set, we call them senders. If an agent has only consuming programs available, we call it a consumer. In this article, we will not be so strict and we will also consider agents that use both generating and consuming programs.

The number of agents in a given P colony is called the degree of Π ; the maximal number of programs of an agent of Π is called the height of Π and the number of the objects inside an agent is called the capacity of Π .

The family of all sets of numbers $N(\Pi)$ computed as above by P colonies of capacity at most $c \geq 0$, degree at most $n \geq 0$ and height at most $h \geq 0$, using checking programs, and working in the maximally parallel way are denoted by $NPCOL_{par}K(c, n, h)$.

If one of the parameters n, h is not bounded, then we replace it with $*$. If only P colonies using programs without checking rules are considered, then we omit parameter K . The family of all sets of numbers $N(\Pi)$ computed as above by restricted P colonies of capacity at most $c \geq 0$, degree at most $n \geq 0$ and height at most $h \geq 0$, not using checking programs, and working in the maximally parallel way are denoted by $NPCOL_{par}R(c, n, h)$, if the P colony is homogeneous the notation of corresponding family is $NPCOL_{par}H(c, n, h)$. In the case of P colonies with generating and consuming programs we can use notation $NPCOL_{par}GC(c, n, h)$.

3. Program Transformations

Let $\Pi = (A, e, f, v_E, B_1, \dots, B_n)$ be a P colony of capacity 2 with n agents.

For all transformations, we will assume a unique labeling of programs such that each program has its own labeling within all agents of the P colony.

Let $P = \bigcup_{i=1}^n P_i$ and R be a set of all program labels such that $R = \{r_i \mid 1 \leq i \leq |P|\}$.

Now we will focus on transforming the different types of programs into one another. We will start with the conversion of restricted programs into homogeneous ones.

3.1. Transformation from Restricted to Homogeneous Programs

If we need to transform restricted programs into homogeneous ones, we must address the situation where homogeneous programs change their entire content in each step—they either rewrite it or exchange it with the environment. During the simulation of executing a restricted program, there is a situation where, due to the rules of the homogeneous program, it is necessary to pull a different object into the agent than the original restricted program requires, and we do not have objects available that the agent could have generated in the previous phase (such as r_i, r'_i, \dots). Even though the environment contains a large number of environmental objects, we cannot use them because these objects may be part of the agent's restricted programs (for example, $\langle a \rightarrow e, e \leftrightarrow b \rangle$). Therefore, we will need to generate an appropriate number of objects that are not part of the original P colony's alphabet. Let this object be $h \notin \Sigma$. The generation of the required number of h objects can be handled by special agents that perform only one program—placing their content (2 objects h) into the environment. Alternatively, the required number of h objects can simply be added to the initial configuration of the environment.

\forall programs r_i of type $\langle a \rightarrow b; c \leftrightarrow d \rangle$ there are programs in the subset C of programs:

- C1. $\langle a \rightarrow r_i; c \rightarrow r'_i \rangle$
- C2. $\langle r_i \leftrightarrow h; r'_i \leftrightarrow e \rangle$
- C3. $\langle h \leftrightarrow r_i; e \leftrightarrow h \rangle$
- C4. $\langle r_i \rightarrow c; e \rightarrow h \rangle$
- C5. $\langle c \leftrightarrow r'_i; h \leftrightarrow d \rangle$
- C6. $\langle r'_i \rightarrow b; d \rightarrow d \rangle$
- C7. $\langle c \leftrightarrow r'_i; h \leftrightarrow h \rangle$
- C8. $\langle r'_i \rightarrow r'_i; h \rightarrow h \rangle$

There are agents of type D in the P colony:

- D1. $\langle h \leftrightarrow e; h \leftrightarrow e \rangle$

Here, we have presented a variant that uses n agents to generate $2n$ objects h .

	agent	agent D	env.	prog.	prog.
1.	ac	hh	xd	$C1$	$D1$
2.	$r_i r'_i$	ee	xdh	$C2$	—
3.	he	hq	$xhdr_i r'_i$	$C3$	—
4.	$r_i h$	hq	$xdr'_i h$	$C4$	—
5.	ch	hq	$xdr'_i h$	$C5$	—
6.	$r'_i d$	hq	xch	$C6$	—
7.	bd	hq	xch	—	—

The program r_i is simulated by performing six programs in six computation steps. Last two programs are used for entering and performing infinite loop.

3.2. Transformation from Generating and Consuming to Restricted Programs

To transform a P colony with insertion and deletion programs into a P colony with restricted programs, we need to use a similar tactic to add a certain number of auxiliary objects h to the initial configuration of the P colony's environment or introduce new agents that place such objects into the environment themselves.

Agents of type G serve as generators of h objects.

$$G1. \langle h \rightarrow h; h \leftrightarrow e \rangle$$

$$G2. \langle e \rightarrow q; h \leftrightarrow e \rangle$$

\forall programs r_i of type $\langle a \rightarrow bd; (c, out) \rangle$ there are programs in subset E of programs of agents:

$$E1. \langle a \rightarrow r_i; c \leftrightarrow e \rangle$$

$$E2. \langle e \rightarrow r'_i; r_i \leftrightarrow h \rangle$$

$$E3. \langle r'_i \rightarrow b; h \leftrightarrow r_i \rangle$$

$$E4. \langle r_i \rightarrow r''_i; b \leftrightarrow h \rangle$$

$$E5. \langle r''_i \rightarrow d; h \leftrightarrow b \rangle$$

Using this program, the agent does not need an object from the environment; instead, it places an object into the environment. Two objects h are required for the simulation.

	agent	env.	prog.
1.	ac	xhh	$E1$
2.	$r_i e$	$xhhc$	$E2$
3.	$r'_i h$	$xchr_i$	$E3$
4.	$r_i b$	$xhhc$	$E4$
5.	$r''_i h$	$xhcb$	$E5$
6.	db	$xhhc$	—

\forall programs r_i of type $\langle ac \rightarrow b; (d, in) \rangle$ there is subset F of programs of the agent:

$$F1. \langle a \rightarrow r_i; c \leftrightarrow d \rangle$$

$$F2. \langle d \rightarrow r'_i; r_i \leftrightarrow c \rangle$$

$$F3. \langle c \rightarrow b; r'_i \leftrightarrow r_i \rangle$$

$$F4. \langle r_i \rightarrow h; b \leftrightarrow r'_i \rangle$$

$$F5. \langle r'_i \rightarrow d; h \leftrightarrow b \rangle$$

To execute the consuming program, the presence of object d in the environment is required. This condition also applies to the execution of the first restricted program in the simulation of rule r_i . If this check were to occur at a later stage in the simulation, it could lead to an improper application of the program, as the simulation of other

programs might temporarily place some objects into the environment that will be back inside these agents by the end of the simulation. Consider a configuration that does not contain object d . During the simulation of a single computation step, one of the agents uses a program of type E4 and temporarily places object d into the environment, intending to remove it in the next computation step using a rule of type E5. If, during the computation phase when object d is present in the environment, a check for the presence of d for a consuming program is performed, then the programs of both agents would be applicable.

	agent	env.	prog.
1.	ac	$xhhd$	$F1$
2.	$r_i d$	$xhhc$	$F2$
3.	$r'_i c$	$xhhr'_i$	$F3$
4.	$r_i b$	$xhhr'_i$	$F4$
5.	$r'_i h$	xhb	$F5$
6.	db	xhh	—

3.3. Transformation from Restricted to Generating and Consuming Programs

Since executing a restricted program involves the agent exchanging an object with another from the environment, both generating and consuming programs must be executed during the simulation. In this paper, we allow one agent to contain both types of these programs.

\forall programs r_i of type $\langle a \rightarrow b; c \leftrightarrow d \rangle$ there are following programs in the subset H of programs of the corresponding agent:

$$H1. \langle ac \rightarrow r_i; (d, in) \rangle$$

$$H2. \langle r_i d \rightarrow r'_i; (e, in) \rangle$$

$$H3. \langle r'_i \rightarrow r''_i c; (e, out) \rangle$$

$$H4. \langle r''_i \rightarrow r'''_i b; (c, out) \rangle$$

$$H5. \langle r'''_i \rightarrow \bar{r}_i d; (b, out) \rangle$$

$$H6. \langle \bar{r}_i d \rightarrow d; (b, in) \rangle$$

Objects r_i, r'_i, r''_i, r'''_i , and \bar{r}_i serve to mark the phase of the restricted program simulation, and they are used to generate the necessary objects accordingly.

step	agent H	env.	prog.
1.	ac	xd	$H1$
2.	$r_i d$	x	$H2$
3.	$r'_i e$	x	$H3$
4.	$r''_i c$	x	$H4$
5.	$r'''_i b$	xc	$H5$
6.	$\bar{r}_i d$	xbc	$H6$
7.	db	xc	—

3.4. Transformation from Homogeneous to Generating and Consuming Programs

Homogeneous programs can be of two types: either both objects in the agent are rewritten into new objects (not

necessarily different from the original ones), or both objects inside the agent are exchanged with two objects that were originally in the environment.

We begin the transformation by simulating a homogeneous program in which the agent exchanges its entire content with the environment. Both objects inside the agent are moved to the environment, and two objects from the environment are moved to the agent. This second transfer imposes a strong applicability condition for the rule, which cannot be checked by consuming programs in a single step. Therefore, we perform the check in two consecutive steps. We must also adjust the simulation of the second type of homogeneous program to ensure that, during the first two steps of their application, the agent does not introduce any objects into the environment that might influence the applicability of the second program in the simulation. If another agent adds an object (e.g., b) to the environment in the first step, which was not previously present, it can change the applicability of the program.

\forall programs r_i of type $\langle a \leftrightarrow b; c \leftrightarrow d \rangle$ there is a subset I of set of programs of the agent:

- I1. $\langle ac \rightarrow r_i; (b, in) \rangle$
- I2. $\langle r_i b \rightarrow r_{i1}; (d, in) \rangle$
- I3. $\langle r_{i1} d \rightarrow r_{i2}; (e, in) \rangle$
- I4. $\langle r_{i2} \rightarrow r_{i3} a; (e, out) \rangle$
- I5. $\langle r_{i3} \rightarrow r_{i4} c; (a, out) \rangle$
- I6. $\langle r_{i4} \rightarrow r_{i5} e; (c, out) \rangle$
- I7. $\langle r_{i5} \rightarrow bd; (e, out) \rangle$
- I8. $\langle r_i b \rightarrow l; (e, in) \rangle$
- I9. $\langle l \rightarrow le; (e, out) \rangle$

Program I8 allows the computation to continue if the homogeneous program r_i was selected for simulation but was not applicable in the original P colony due to the absence of object d in the environment. Program I9 is included in the agent's set of programs only once, and its execution results in an infinite loop.

step	agent	env.	prog.
1.	ac	xbd	I1
2.	$r_i b$	xd	I2
3.	$r_{i1} d$	x	I3
4.	$r_{i2} e$	x	I4
5.	$r_{i3} a$	x	I5
6.	$r_{i4} c$	xa	I6
7.	$r_{i5} e$	xac	I7
8.	db	xac	—

\forall programs r_i of type $\langle a \rightarrow b; c \rightarrow d \rangle$ there is a subset J of set of programs of the agent:

- J1. $\langle ac \rightarrow r_i; (e, in) \rangle$
- J2. $\langle r_i \rightarrow r_{i1} e; (e, out) \rangle$
- J3. $\langle r_{i1} \rightarrow r_{i2} e; (e, out) \rangle$
- J4. $\langle r_{i2} \rightarrow r_{i3} e; (e, out) \rangle$
- J5. $\langle r_{i3} \rightarrow r_{i4} e; (e, out) \rangle$
- J6. $\langle r_{i4} \rightarrow r_{i5} e; (e, out) \rangle$
- J7. $\langle r_{i5} \rightarrow bd; (e, out) \rangle$

In addition to ensuring that during the first step of simulating program r_i , the agent does not emit any objects that could affect the applicability of other programs in the P colony, it is also necessary to match the length of the program simulation of r_i to the number of steps required for a successful simulation of the above-mentioned type of homogeneous program. The reader can notice that otherwise, the simulation could be completed in two steps by using two programs (J1 and the modified J7). If the simulation of one type of homogeneous program took a different number of steps than the simulation of the other type, the simulation of steps of computation would start to overlap, and the applicability of individual programs might not reflect the configuration of the original model.

step	agent	env.	prog.
1.	ac	x	J1
2.	$r_i e$	x	J2
3.	$r_{i1} e$	x	J3
4.	$r_{i2} e$	x	J4
5.	$r_{i3} a$	x	J5
6.	$r_{i4} e$	x	J6
7.	$r_{i5} e$	x	J7
8.	db	x	—

3.5. Transformation from Homogeneous to Restricted Programs

We will conclude the section on transformations with the conversion of homogeneous programs into restricted programs.

To begin with, it should be noted that this transformation is not complete. In situations where the P colony is in a halt configuration, but the agent configuration and the environment's content allow for the initiation of the simulation of a homogeneous program with communication rules, it results in an infinite loop instead of halting the computation. We present the transformation here for the sake of completeness.

\forall programs r_i of type $\langle a \rightarrow b; c \rightarrow d \rangle$:

1. $\langle a \rightarrow r_i; c \leftrightarrow e \rangle$
2. $\langle r_i \rightarrow r'_i; e \leftrightarrow c \rangle$
3. $\langle c \rightarrow r''_i; r'_i \leftrightarrow e \rangle$
4. $\langle r''_i \rightarrow d; e \leftrightarrow e \rangle$
5. $\langle d \rightarrow d; e \leftrightarrow r'_i \rangle$
6. $\langle r'_i \rightarrow r'''_i; d \leftrightarrow e \rangle$
7. $\langle r'''_i \rightarrow b; e \leftrightarrow d \rangle$

To better illustrate how executing individual restricted programs leads to the same outcome as executing the homogeneous program r_i , we will provide a step-by-step demonstration. The following table presents the configurations of the agent and the environment, along with the applicable rule for the agent.

	agent	env.	prog.
1.	ac	x	1
2.	$r_i e$	xc	2
3.	$r'_i c$	x	3
4.	$r''_i e$	xr'_i	4
5.	de	xr'_i	5
6.	dr'_i	x	6
7.	$r''_i e$	xd	7
8.	bd	x	–

The program $r_i : \langle a \rightarrow b; c \rightarrow d \rangle$ is simulated by performing seven programs in seven computation steps.

\forall programs r_i of type $\langle a \leftrightarrow b; c \leftrightarrow d \rangle$:

To simulate execution of the program r_i the agent has a subset A of programs :

- A1. $\langle a \rightarrow r_i; c \leftrightarrow d \rangle$
- A2. $\langle d \rightarrow r''_i; r_i \leftrightarrow c \rangle$
- A3. $\langle c \rightarrow d; r'_i \leftrightarrow e \rangle$
- A4. $\langle e \rightarrow e; d \leftrightarrow r''_i \rangle$
- A5. $\langle r''_i \rightarrow e; e \leftrightarrow r'_i \rangle$
- A6. $\langle r'_i \rightarrow b; e \leftrightarrow d \rangle$

Agent B (for every agent in original P colony there is one agent of type B , or for every communication program there is one such agent):

- B1. $\langle e \rightarrow a; e \leftrightarrow r_i \rangle$
- B2. $\langle r_i \rightarrow r'_i; a \leftrightarrow b \rangle$
- B3. $\langle b \rightarrow c; r'_i \leftrightarrow e \rangle$
- B4. $\langle e \rightarrow e; c \leftrightarrow e \rangle$
- B5. $\langle r_i \rightarrow l; a \leftrightarrow e \rangle$
- B6. $\langle l \rightarrow l; e \leftrightarrow e \rangle$

The agent, by executing programs from set A , sends object r_i to the environment, which acts as a label for the executed program and simultaneously checks for the presence of object d in the environment. Agent B , based on the object r_i , checks for the presence of object b in the environment. This checking process involves moving the checked object inside the agent. Programs $B5$ and $B6$ ensure an infinite loop if the program r_i being simulated is not applicable (i.e., object b is missing from the environment). Program $B5$ can also be applied even if object b is present in the environment because the selection of a program from the set of applicable programs is random. Since the computation is defined as nondeterministic, a corresponding computation exists for the case where program r_i is applied in the correct configuration. Other computations result in infinite loops and thus do not lead to a result.

	agent	agent B	env.	prog.A	prog.B
1.	ac	ee	bd	A1	–
2.	$r_i d$	ee	bc	A2	–
3.	$r'_i c$	ee	br_i	A3	B1
4.	ed	ar_i	br''_i	A4	B2
5.	er''_i	$r'_i b$	ad	–	B3
6.	er'_i	ce	adr'_i	A5	B4
7.	er_i	ee	adc	A6	–
8.	bd	ee	ac	–	–

4. Computational Power of P colonies

For restricted P colonies, not using checking rules, the following results are known:

- $NPCOL_{par}R(2, *, 5) = NRE$ in [5],
- $NPCOL_{par}R(2, 2, *) = NRE$ in [6],
- $NPCOL_{par}R(2, 57, 8) = NRE$ in [7].

For homogeneous P colonies, not using checking rules, the following results are known:

- $NPCOL_{par}H(2, 92, 3) = NRE$ in [7],
- $NPCOL_{par}H(2, 70, 5) = NRE$ in [7],
- $NPCOL_{par}H(2, 2, 163) = NRE$ in [7].

For P colonies using generating and consuming programs there are known results only for cases when agents contain only one type of such programs, the agents are called senders and consumers.

- $NPCOL_{sc}(2, 2, *) = NRE$ in [8, 9].

From the transformations described, we can derive data for new results related to restricted, homogeneous, and especially P colonies with generating and consuming programs.

For homogeneous P colonies, we can add the following result:

- $NPCOL_{par}H(2, 57, 64) = NRE$.

For homogeneous P colonies, we find that the class of P colonies with at most 57 agents, each having up to 64 programs, is computationally complete. This result allows us to reduce the number of agents from 70 to 57 by increasing the number of programs associated with each agent.

For P colonies with generating and consuming programs, there are four new results:

- $NPCOL_{par}GC(2, 2, 1305) = NRE$,
- $NPCOL_{par}GC(2, 57, 48) = NRE$,
- $NPCOL_{par}GC(2, 70, 41) = NRE$,
- $NPCOL_{par}GC(2, 92, 25) = NRE$.

5. Conclusion

In this study, we explored various P colony models—restricted, homogeneous, and those using generating and consuming programs—highlighting their operational differences. Our analysis showed that each model has unique computational capabilities, shaped by its rule types and agent structure. For example, restricted

P colonies, with their mix of rewriting and communication rules, exhibit different computational behaviors than homogeneous colonies, which use only one type of rule. Transformations between these models also reveal insights into their computational power.

In the final section of the paper, we presented six new results that illustrate the dependence of the number of agents and the maximum number of programs associated with a single agent on the computational completeness of P colonies constrained by these parameters. For P colonies with generating and consuming programs, these are the first results, as the computational power of colonies using such program combinations associated with a single agent has not yet been explored.

6. Acknowledgments

This work is supported by the Silesian University in Opava under the Student Funding Plan, project SGS/9/2024.

References

- [1] J. Kelemen, A. Kelemenová, Gh. Păun, 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, 2004, pp. 82–86.
- [2] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
- [3] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, Inc., New York, NY, USA, 2010.
- [4] L. Ciencialová, L. Cienciala, P. Sosík, Generalized P colonies with passive environment, *Theoretical Computer Science* 724 (2018) 61–68. doi:<https://doi.org/10.1016/j.tcs.2017.12.009>.
- [5] R. Freund, M. Oswald, P colonies working in the maximally parallel and in the sequential mode, in: D. Zaharie, D. Petcu, V. Negru, T. Jebelean, G. Ciobanu, A. Cicortas, A. Abraham, M. Paprzycki (Eds.), Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25–29 September 2005, Timisoara, Romania, IEEE Computer Society, 2005, pp. 419–426. doi:[10.1109/SYNASC.2005.55](https://doi.org/10.1109/SYNASC.2005.55).
- [6] L. Cienciala, L. Ciencialová, A. Kelemenová, On the number of agents in P colonies, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4860 LNCS (2007) 193–208.
- [7] L. Cienciala, L. Ciencialová, Some new results of P colonies with bounded parameters, *Natural Computing* (2016) 1–12. doi:[10.1007/s11047-016-9591-0](https://doi.org/10.1007/s11047-016-9591-0).
- [8] L. Cienciala, L. Ciencialová, P colonies and their extensions, in: J. Kelemen, A. Kelemenová (Eds.), *Computation, Cooperation, and Life*, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 158–169.
- [9] L. Ciencialová, L. Cienciala, P. Sosík, Generalized P colonies with passive environment, in: C. Graciani, D. Orellana-Martín, A. Riscos-Núñez, Á. Romero-Jiménez, L. Valencia-Cabrera (Eds.), *Fourteen Brainstorming Week on Membrane Computing*, 2016, pp. 151–162.