Working with Positive Integers in P Colony rw-Automata

Lucie Ciencialová[1,*,†], Luděk Cienciala[1,†]

[1]Institute of Computer Science, Faculty of Philosophy and Science in Opava, Silesian Univerity in Opava, Opava, Czech Republic

Abstract

We introduce new variant of P colonies that we call P colony rw-automaton - a theoretical model from the membrane computing model family. Its inspiration comes from nature and the structure and functioning of living organisms. The model is formed from agents - a collection of objects embedded in a membrane, equipped with programs to manipulate objects. The agents share objects placed on the tape and in the environment. In this paper, we focus on natural number encoding and sorting of positive natural numbers by an algorithm similar to Bead sort executed by the P colony rw-automata.

Keywords

membrane computing, P Colony rw-automata, natural number representation, sorting, Bead sort

1. Introduction

P colony, introduced in [1], is a theoretical computing model inspired by structure and behavior of simple one-cell organisms living in a shared environment. The P colony is formed from basic units agents equipped with programs. The crucial role play the environment, it can store products of agents functioning and through the environment the agents can send "messages" to each other. The functioning of agents is based on objects.

Within each agent, there exists a finite multiset of objects. These objects undergo processing by a finite set of associated programs unique to each agent. The number of objects residing in each agent remains constant throughout the operation of the agent community, and this fixed quantity is referred to as the "capacity" of the P colony.

The agents collectively share an environment, which is represented by another multiset of objects. Among these objects, one particular type is identified as the "environmental object." This type is assumed to exist in an infinitely countable number of copies within the environment. (It should be noted that in the literature, one may also encounter instances where the environmental symbol appears in an arbitrarily large number of copies in the environment).

By utilizing their respective programs, the agents can alter the objects available to them and exchange some of their objects with those found in the environment. These coordinated actions lead to a configuration change, or transition, within the P colony. A finite sequence of consecutive configuration changes, initiated 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 conclusion of the process.

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

The programs in the P colony comprise three distinct types of rules. The first type, known as "evolution rules", takes the form of $a \rightarrow b$, indicating that an object a within the agent is rewritten or evolved into object b.

The second type, referred to as "communication rules", follows the pattern $c \leftrightarrow d$. Upon executing a communication rule, object c inside the agent swaps positions with object d in the environment. As a result, object d is now located inside the agent, and object c resides in the environment.

The third type of rules, called "checking rules," are derived from two rules of either evolution or communication types. When a checking rule r_1/r_2 is executed, rule r_1 takes precedence over rule r_2. This means that the agent first checks if rule r_1 is applicable; if so, it must be used. In case rule r_1 is not applicable, the agent uses rule r_2.

In P colony rw-automaton we distinguish between communication rules that work with objects in a multiset (called non-tape rules) and communication rules that work with objects on tape (called tape rules).

We will present the possibilities of this new variant of P colonies by examples of working with natural numbers (conversions to different number systems) and sorting a given number of natural numbers.

The structure of the paper is as follows: after an introductory section, we introduce the basic concepts related

ITAT 2023: Information Technologies – Applications and Theory, September 22–26, 2023, Vysoké Tatry, Slovakia

*Corresponding author.

†These authors contributed equally.

lucie.ciencialova@fpf.slu.cz (L. Ciencialová);
ludek.cienciala@fpf.slu.cz (L. Cienciala)
0000-0002-0877-7063 (L. Ciencialová); 0000-0001-7116-9338 (L. Cienciala)

to the original P colony model, which we then develop into a new variant called P colony rw-automaton. In the third section we give examples of the P colony rw-automaton, which implements the conversion of a natural number given in the decimal system (on a tape) into the unary system (the number of certain objects in the environment). In the next example we will discuss the opposite conversion, i.e. the conversion of a number in the unary representation to a number in the decimal system, which will be placed on the tape at the end of the calculation. Last we will give how to construct a P colony rw-automaton that is able to sort a given number of natural numbers. These numbers are placed on the tape in such a way that the beginning and end of their enumeration is marked, and they are also separated from each other by a special object. At the end of the calculation, the tape will list the same numbers ordered by size from largest to smallest.

2. Preliminaries and Basic Notions

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 $f : O \to N$; 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 Colony

In the following we describe the concept of a P Colony. Consider original definition of P colony introduced in [1].

Definition 1. *A P colony of capacity k, $k \geq 1$, is a construct*

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

- *A is an alphabet, its elements are called objects;*
- *$e \in A$ is the basic (or environmental) object of the colony;*
- *$f \in A$ is the final object of the colony;*
- *v_E is a finite multiset over $A - \{e\}$, called the initial state (or initial content) of the environment;*
- *B_i, $1 \leq i \leq n$, are agents, where each agent $B_i = (o_i, P_i)$ is defined as follows:*
  - *o_i is a multiset over A consisting of k objects, the initial state (or the initial content) of the agent;*
  - *$P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite set of programs, where each program consists of k rules, which are in one of the following forms each: (1) $a \to b$, $a, b \in A$, called an evolution rule; (2) $c \leftrightarrow d$, $c, d \in A$, called a communication rule; (3) r_1/r_2, called a checking rule; r_1, r_2 are evolution rules or communication rules.*

The agent's activity is governed by its programs, enabling the agent to modify its state and/or the state of the environment.

The environment consists of a finite number (including zero) of copies of non-environmental objects and a countably infinite number of copies of the environmental object, denoted as e.

When an agent executes a program, each object within the agent is affected. Depending on the rules within the program, the execution may also impact the environment. This interaction between agents and the environment is pivotal to the operation of the P colony.

The functioning of the P colony begins from its initial configuration or initial state. The initial configuration is represented as an $(n + 1)$-tuple of multisets of objects present in the P colony at the start of computation. These multisets are denoted as o_i for $1 \leq i \leq n$ and v_E for the environment.

In each step of the computation, both the environment and agent's states undergo changes. In the "maximally parallel" derivation mode, all agents that can employ any of their programs do so simultaneously (non-deterministically chosen). Conversely, in the "sequential" derivation mode, only one agent at a time is allowed to use one of its programs (non-deterministically chosen). If an agent has multiple applicable programs, it non-deterministically selects one.

A sequence of transitions constitutes a "computation." A computation becomes "halting" when it reaches a configuration where no further programs can be applied. The result of a halting computation is determined by the number of copies of a specific object, denoted as f, present in the environment during the halting configuration.

Due to the non-determinism in program selection, multiple computations can be derived from the initial configuration. Thus, a P colony is associated with a set of numbers, denoted as $N(\Pi)$, which are computed through all possible halting computations of the given P colony.

2.2. P Colony rw-Automaton

In P colony([1]), the agents are placed in the environment – multiset of objects. One special kind of objects is called environmental and it is placed in the environment in sufficient number of copies. In addition to the multiset of objects, the P colony rw-automaton environment (like in P colony automaton [4]) contains a tape on which the objects are placed in sequence one by one. We refer to them as a string on the tape, and we call a part of the sequence of objects a substring.

As in the case of standard P colonies, agents of the P colony rw-automaton contain objects, each being an element of a finite alphabet. With every agent, a set of programs is associated. To work with tape, agents need to have rules in their programs that they can read and write to tape. Similar rules have been introduced for PCol Automaton in [5] and in generalized P colony automata[4, 6]. However, the function of the tape rules was different. The agents did not directly interfere with the objects on the tape, but merely followed the objects from the part of the tape they were currently reading. The tape rules we use here for the P colony rw-automaton directly modify objects on the tape, similar to the model named APcol system([7]). In this model, programs are an ordered pairs of rules. If these are communication rules, then there is an exchange of objects between the tape and the agent. When a program contains two communication rules, it is the order of the rules that determines the order of two consecutive objects on the tape on which the agent will apply the program. For more details about definitions, features and computational power of APCol systems see [7, 8, 9] and details about PCol automata can be found in [5, 10].

Definition 2. *A P colony rw-automaton of capacity* 2 *and with* n *agents,* $n \geq 1$*, is a construct* $\Pi = (A, e, v_E, (o_1, P_1), \ldots, (o_n, P_n))$ *where*

- A *is an alphabet, the alphabet of the P colony rw-automaton, its elements are called objects;*
- $e \in A$ *is the environmental object of the P colony automaton;* $v_E \in (A - \{e\})^*$ *is a string representing the multiset of objects different from* e*, called the initial state of the environment ;*
- $(o_i, P_i), 1 \leq i \leq n$*, is the* i*-th agent; where*
  - o_i *is a multiset over* V*,* $|o_i| = 2$*, the initial state (contents) of the agent,*
  - P_i *is a set of programs, where every program is a pair rules, each of them is one of the following types:*
    - *tape rules of the form* $a \overset{T}{\leftrightarrow} b$*, called communication tape rules; or*
    - *non-tape rules of the form* $a \rightarrow b$*, or* $c \leftrightarrow d$*, called rewriting (non-tape) rules and communication (non-tape) rules, respectively.*

The computation starts in the initial configuration $(v_E, o_1, \ldots, o_n, s_0)$, i.e., when the environment and all agents are in initial state and the input string s_0 is on the input tape.

Let us look at the tape in more detail. The tape is potentially infinite on both sides. Since agents need another object to insert objects onto the tape, they replace object on the tape by the one inside the agent - they use the tape communication rules - the tape cannot be empty. We place environmental objects on it. If we place the string s on the tape, then the tape contains e^*se^*. The string s itself can contain environmental objects. When working with tape, writing configuration or giving output we omit environment objects to the left of the first object different from e and environment objects to the right of the last occurrence of a symbol different from e.

For a configuration $(w_E, w_1, \ldots, w_n, s)$, where s is a string placed on the tape, the sets of applicable programs, $\mathcal{P}$, can be constructed in such a way that: The agent must use all the rules in the program. Tape rules are used to an arbitrary object of the required type on the tape if one rule is tape rule and the other one non tape. In the case of program $\left\langle a \overset{T}{\leftrightarrow} b; c \overset{T}{\leftrightarrow} d \right\rangle$, a two consequent objects b and d in this order must be on the tape. They are replaced by objects a and c, respectively. If the program is of the form $\left\langle c \overset{T}{\leftrightarrow} d; a \overset{T}{\leftrightarrow} b \right\rangle$, then a substring db of the tape is find and it is replaced by string ca.

From the set of applicable programs $\mathcal{P}$ the programs to be executed are selected. These programs form a set P for which the following conditions must be satisfied: (1) Each agent has at most one program in P. (2) One object in the environment or on the tape is affected by at most one agent, with at most one rule. (3) If we add any program from $\mathcal{P}$ to P, the previous conditions cannot be satisfied.

The automaton passes from one configuration to another by using all programs from the set of selected applicable programs. Such transitions between configurations form a computation. Since the set P can be selected from the set of all applicable programs in multiple ways, the P colony rw-automaton can pass from one configuration to multiple different configurations, resulting in multiple computations starting from the initial configuration.

The computation by a P colony rw-automaton may end by halting, when there is no applicable program in the last configuration.

P colony rw-automaton can work in the accepting, generating or computing mode. In accepting mode, the P colony rw-automaton starts computation with the input on the tape, and if it stops, the input is accepted. When the P colony rw-automaton is working in generating

mode, it starts with only objects e on the tape and the result of the computation can be found on the tape after the computation has halted. In computing mode, the input is transformed to an output, and the output is valid only when the computation stops.

3. Natural number representation

To express the number of things, animals, etc. we use usually decimal representation of numbers. In Computer Science also binary and hexadecimal representations are used. What about number representation in area of membrane systems? Mostly the unary representation is used. It means that the number is determined by the number of some kind of objects. This is used in proofs of computational power of such systems when simulation of register machine takes part. In this section we focus on transformation between decimal (and possibly any reasonable) representation of number to unary and vice versa.

Consider a number in decimal representation that is written on the tape of P colony rw-automaton in a form $\#number\$$ ($\#$ is specific symbol that indicates the beginning of the given number and $ indicates its end). The agents in P colony rw-automaton initially containing two objects e are grouped into modules having some function. There are four modules in P colony rw-automaton to translate decimal number to its unary representation – Controller, Module 1 (translate digit onto the corresponding number of object $\overline{a}$), Module 2 (multiply the number of objects $\overline{a}$ in the environment by 10).

The first module is Controller. It is formed from one agent and it performs reading from a tape and calling other modules to work. It is done by passing object "message" to the environment. The functioning of the agent controller can be described as:

1. read symbol from the tape (the one next to mark of currently read symbol)
2. if the symbol is a digit continue 3. else stop (empty input)
3. call Module 1 - translate digit into the number of objects $\overline{a}$
4. read symbol from the tape (the one next to mark of currently read symbol)
5. if the symbol is a digit then
   a) call Module 2 - multiplier of $\overline{a}$ present in the environment
   b) call Module 1 - translate digit into the number of objects $\overline{a}$
   c) continue with 4.
6. else (if the symbol is $) stop the translation.

Programs for reading from the tape:

1. $< e \to H;\ e \to k >$
2. $< H \overset{T}{\leftrightarrow} \#;\ k \overset{T}{\leftrightarrow} i > \quad 0 \leq i \leq 9$
3. $< H \overset{T}{\leftrightarrow} \#;\ k \overset{T}{\leftrightarrow} \$ >$
4. $< \# \to e;\ \$ \to F >$

Program 4. is processed only if there is no number in the input (there is no digit between symbols $\#$ and $)

5. $< \# \to c_1;\ i \leftrightarrow e > \qquad 0 \leq i \leq 9$
6. $< c_1 \to c_1;\ e \to m_1 >$
7. $< c_1 \to w_1;\ m_1 \leftrightarrow e >$
8. $< w_1 \to k;\ e \leftrightarrow f_1 >$

Symbol f_1 placed in the environment means that Module 1 finished its work.

9. $< k \to k;\ f_1 \to e >$
10. $< e \overset{T}{\leftrightarrow} k;\ k \overset{T}{\leftrightarrow} i > \qquad 0 \leq i \leq 9$
11. $< k \to C_2;\ i \leftrightarrow e > \qquad 0 \leq i \leq 9$
12. $< C_2 \to C_2;\ e \to m_2 >$
13. $< C_2 \to w_2;\ m_2 \leftrightarrow e >$
14. $< w_2 \to C_1;\ e \leftrightarrow f_2 >$
15. $< C_1 \to C_1;\ f_2 \to m_1 >$
16. $< C_1 \to w_1;\ m_1 \leftrightarrow e >$
17. $< e \overset{T}{\leftrightarrow} k;\ k \overset{T}{\leftrightarrow} \$ >$
18. $< k \to e;\ \$ \to F >$

Module 1 is formed from one agent. When the module is called (object m_1 appears in the environment) the agent can consume it together with object corresponding to digit read from the tape.

$A1.$ $< e \leftrightarrow m_1;\ e \leftrightarrow i > \qquad 0 \leq i \leq 9$
$A2.$ $< m_1 \to f_1;\ 0 \to e >$
$A3.$ $< f_1 \leftrightarrow e;\ e \to e >$
$A4.$ $< m_1 \to \overline{a};\ i \to (i-1)' > \quad 1 \leq i \leq 9$
$A5.$ $< \overline{a} \leftrightarrow e;\ i' \to i > \qquad 0 \leq i \leq 8$
$A6.$ $< e \to \overline{a};\ i \to i' > \qquad 1 \leq i \leq 8$
$A7.$ $< e \to f_1;\ 0 \to e >$

Module 2 is formed from two agents. When Module 2 is called, the first agent must multiply each occurrence of object $\overline{a}$ by ten. It means that it consume one object $\overline{a}$ and immediately generates ten objects $\boxed{a}$. When there is no object $\overline{a}$ in the environment agent generates object g and this is message for the second agent to exchange all $\boxed{a}$ by $\overline{a}$ and generate object f_2 - message for Controller that multiplication is over.

The first agent's programs are:

$B1.$ $< e \leftrightarrow m_2;\ e \leftrightarrow \overline{a}/e \leftrightarrow e >$
$B2.$ $< m_2 \to f_2;\ e \to e >$
$B3.$ $< f_2 \leftrightarrow e;\ e \to e >$
$B4.$ $< m_2 \to p_0';\ \overline{a} \to \boxed{a} >$
$B5.$ $< p_j' \to p_{j+1};\ \boxed{a} \leftrightarrow e > \qquad 0 \leq j \leq 9$
$B6.$ $< p_j \to p_j';\ e \to \boxed{a} > \qquad 1 \leq j \leq 9$
$B7.$ $< p_{10} \to m_2';\ e \leftrightarrow \overline{a}/e \leftrightarrow e >$
$B8.$ $< m_2' \to x;\ e \to e >$
$B9.$ $< x \leftrightarrow e;\ e \to e >$
$B10.$ $< m_2' \to p_0';\ \overline{a} \to \boxed{a} >$

When object x appears in the environment the second agent consumes it and starts to exchange all $\boxed{a}$ by $\overline{a}$.

$C1.$ $< e \leftrightarrow x;\ e \leftrightarrow \boxed{a}/e \leftrightarrow e >$
$C2.$ $< x \rightarrow f_2;\ e \rightarrow e >$
$C3.$ $< f_2 \leftrightarrow e;\ e \rightarrow e >$
$C4.$ $< x \rightarrow y;\ \boxed{a} \rightarrow \overline{a} >$
$C5.$ $< y \rightarrow y;\ \overline{a} \leftrightarrow e >$
$C6.$ $< y \rightarrow x;\ e \leftrightarrow \boxed{a}/e \leftrightarrow e >$

After all objects $\boxed{a}$ are replaced by $\overline{a}$ the second agent sends object f_2 to the environment and this is message for the Controller to start pass object m_1 to Module 1.

After Controller reads object \$ from the tape it stops working without passing any object to the environment so all the agents have no applicable program. P colony rw-automaton with four agents divided into three modules reads decimal number from the tape and after computation halts there are the corresponding number of $\overline{a}$ is placed in the environment.

Now we focus on the reverse transformation - from unary representation of number to decimal number. The unary representation of the number is stored in the environment as the number of copies of object $\overline{a}$. The result is decimal number stored on the tape at the end of computation.

The idea is to use one agent consuming objects $\overline{a}$. The agent in initial state $f_2 e$ uses the second object inside it as counter. If there is no $\overline{a}$ it generate object as message for the second agent to write corresponding digit onto tape and the object message for the third agent that can exchange all $\boxed{a}$ by $\overline{a}$. The third agent generate object to start work of the first agent again. If the first agent consumes ten $\overline{a}$s it generates one object $\boxed{a}$ and continue consuming $\overline{a}$.

The programs of the first agent are as follows:

$D1.$ $< f_2 \rightarrow \overline{0};\ e \leftrightarrow \overline{a}/e \leftrightarrow e >$
$D2.$ $< \overline{i} \rightarrow i_T;\ e \rightarrow x' >$ $\quad 0 \leq i \leq 9$
$D3.$ $< i_T \leftrightarrow e;\ x' \rightarrow x >$ $\quad 0 \leq i \leq 9$
$D4.$ $< x \leftrightarrow e;\ e \rightarrow e >$
$D5.$ $< \overline{i} \rightarrow (i+1)';\ \overline{a} \rightarrow e >$
$D6.$ $< i' \rightarrow \overline{i};\ e \leftrightarrow \overline{a}/e \leftrightarrow e >$
$D7.$ $< \overline{10} \rightarrow 10'';\ \overline{a} \rightarrow \boxed{a} >$
$D8.$ $< 10'' \rightarrow 0';\ \boxed{a} \leftrightarrow e >$
$D9.$ $< e \leftrightarrow f_2;\ e \rightarrow e >$

The second agent started when some $i_T, 0 \leq i \leq 9$, appears in the environment. The agent consumes it and exchange $e\#$ by $\#i$. The decimal number is written from the right to the left.

$E1.$ $< e \leftrightarrow i_T;\ e \rightarrow i' >$ $\quad 0 \leq i \leq 9$
$E2.$ $< i_T \rightarrow \#;\ i' \rightarrow i >$ $\quad 0 \leq i \leq 9$
$E3.$ $< \# \overset{T}{\leftrightarrow} e;\ i \overset{T}{\leftrightarrow} \# >$
$E4.$ $< \# \rightarrow e;\ e \rightarrow e >$

The third agent has the same programs as the agent from Module 2 (programs C1.– C6.).

The P colony rw-automaton with three agents starts the computation in initial configuration with # on the tape and n copies of $\overline{a}$ in the environment. The first agent contain $f_2 e$, the second and third agents contains ee each. When computation halts the resulting string can be found on the tape in the form $\#n$ where n is decimal representation of the number of $\overline{a}$s in the environment at the beginning of computation. If the input is given on the tape we need one more agent to "copy" objects from the tape and after reaching the end of the string it can generate object f_2. In this case the first agent initial state is ee.

Both P colony rw-automata can easily be adjusted to perform transformation to any number system using another limits instead of 8, 9 and 10. For example for hexadecimal system 14, 15 and 16 are used in programs with i instead of upper limit.

4. Sorting natural numbers

In this section, we focus on construction of such P colony rw-automaton that finds k unsorted positive integers (in decimal number system) on the tape in the form $\#n_1\ m\ n_2\ m \ldots m\ n_k\$$. The input variable for construction of P colony rw-automaton is k.

The idea is to transform given decimal numbers to unary representation (we need k different a-objects, we can use indices 1 to k), sort them and write sorted numbers onto tape in a form similar to input string.

We can use Controller, Module 1 and Module 2 from previous section to read the tape and place copies a_i into the environment. Controller has to call Modules with proper index (corresponding to the order of the number stored on the tape). The Modules generate as with the same index or we can use k pairs Modules each pair activated by another index.

The idea of sorting is a little bit similar to Bead sort. We run $k+1$ agents in parallel - k agents consuming a_is and the last one is generating results. When there is a_i to be consumed by agent i, no object is placed to the environment by agent i. When there is no a_i in the environment agent i place object q to the environment. this object is consumed by agent $k+1$. This phase can be called consuming.

Until now, the agent $k+1$ generated object R_j at each step when k agents consumed their objects. The moment q appears in the environment the agent increments the index of the generated object ($R_{(j+1)}$). In the case of multiple objects q (identical numbers in a sequence of numbers) the agent $k+1$ increases the index until it has consumed all the objects of q. Only then can the other agents continue their work. We can implement this part in this way that after agent i has consumed object a_i it applies programs with rewriting rules in the next steps, and thus wait $k-1$ steps to allow agent $k+1$ to process all possible objects q from the environment.

After this phase there is no object $a_i, 1 \leq i \leq k$, in the environment. There are some object R_j in the

environment where $j \in 1, \ldots k$. For example if the input sequence of numbers is 3, 5, 4, 3, 5, then after consuming phase the environment contains objects $R_1R_1R_1R_3R_4$.

In the next phase, the R_j objects are processed, copied and transformed into the decimal system with writing to tape. It means that every object R_1 is replaces by pair P_1R_2, then all object P_1 are transformed to decimal number and this number is written to the tape. Note that the output is written from right to left and that the number of objects P_1 corresponds to the smallest number in the input sequence. The same transformation is done by all k type of R_j objects.

We list here some programs that belong to the agents consuming a_i, and the agent $k+1$.

The set of programs of agent i, $1 \leq i \leq k$ contains programs:

$F1. \quad < B \to q;\ e \leftrightarrow a_i/e \leftrightarrow e >$
$F2. \quad < q \to B_0;\ a_i \to e >$
$F3. \quad < B_l \to B_{(l+1)};\ e \to e > \qquad 0 \leq l \leq 2k-2$
$F4. \quad < B_{(2k-1)} \to B;\ e \to e >$
$F5. \quad < q \leftrightarrow e;\ e \to s >$

The agent $k+1$ has following programs in its programs set ($1 \leq p \leq k-1; 2 \leq r \leq 2k-2; 0 \leq t \leq 2k-2$):

$G1. \quad < B_p \to B'_p;\ e \to e >$
$G2. \quad < B'_p \to B''_p;\ e \to e >$
$G3. \quad < B''_p \to p_0;\ e \leftrightarrow q/e \leftrightarrow e >$
$G4. \quad < p_0 \to p_1;\ e \to 1 >$
$G4. \quad < p_1 \to p_2;\ 1 \leftrightarrow e >$
$G4. \quad < p_{(r)} \to p_{(r+1)};\ e \to e >$
$G5. \quad < p_{(2k-2)} \to B_p;\ e \to e >$
$G6. \quad < p_0 \to (p+1)'_1;\ q \to e >$
$G7. \quad < p'_t \to (p+1)'_{t+1};\ q \to e >$
$G8. \quad < p'_t \to (p)'_{t+1};\ e \leftrightarrow q/e \leftrightarrow e >$
$G9. \quad < k'_{(2k-2)} \to s;\ q \to e >$
$G10. \quad < k'_{(2k-2)} \to s;\ e \to e >$
$G10. \quad < k_{(2k-2)} \to s;\ e \to e >$

The number of steps that P colony rw-automaton executes during the consuming phase depends on the number of given numbers (k) and the maximum of these numbers (n) and equals to $(2k+2) \cdot n$.

5. Conclusion

In this research paper we introduce a new variant of P colony – P colony rw-automaton that combines features of APcol system, generalized P colony automaton and PCol automaton. It uses tape and non-tape rules to work with objects on the tape and in the environment. The functioning of P colony rw-automaton is shown on work with natural numbers – transformation between representation of the number in unary and decimal number systems and sorting natural numbers. Within these examples of use P colony rw-automaton we show how to multiply and divide unary representation of natural number by given number (in our case 10).

6. Acknowledgments

This work is supported by the Silesian University in Opava under the Student Funding Plan,project SGS/11/2023.

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] K. Kántor, G. Vaszil, Generalized P Colony Automata 19 (2014) 145–156.

[5] L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil, PCol automata: recognizing strings with P colonies, in: M. Ángel del Amor, Gh. Păun, I. Pérez Hurtado de Mendoza, A. Riscos Núnez (Eds.), Eighth brainstorming week on membrane computing. Sevilla, 2010. (RGNC Report 01/2010.), Fénix Editora, Sevilla, 2010, pp. 65–76.

[6] K. Kántor, G. Vaszil, On the classes of languages characterized by generalized P colony automata, Theoretical Computer Science 724 (2018) 35–44. URL: https://www.sciencedirect.com/science/article/pii/S0304397517309027. doi:https://doi.org/10.1016/j.tcs.2017.12.011.

[7] L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú, P colonies processing strings, Fundamenta Informaticae 134 (2014) 51–65. doi:10.3233/FI-2014-1090.

[8] L. Ciencialová, E. Csuhaj-Varjú, G. Vaszil, L. Cienciala, APCol systems with verifier agents, in: T. Hinze, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing - 19th International Conference, CMC 2018, Dresden, Germany, September 4-7, 2018, Revised Selected Papers, volume 11399 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 95–107. URL: https://doi.org/10.1007/978-3-030-12797-8_8. doi:10.1007/978-3-030-12797-8_8.

[9] L. Ciencialová, L. Cienciala, Two notes on APCol systems, Theor. Comput. Sci. 805 (2020) 161–

174. URL: https://doi.org/10.1016/j.tcs.2018.07.006. doi:10.1016/j.tcs.2018.07.006.

[10] L. Cienciala, L. Ciencialová, Computation, cooperation, and life, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 158–169.