Transferable Knowledge in P Colonies

Lucie Ciencialová[1], Luděk Cienciala[1]

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

Abstract
In this paper we introduce the notion of transferable programs as shared knowledge applied in different types of P colonies. We will discuss the basic type of P colonies and 2D P colonies. We will show the possibilities of using transferable programs using examples.

Keywords
membrane computing, P colonies, transferable knowledge, paralell computing, multi-agent system

1. Introduction

P colony is very simple computational device related to membrane systems inspired by colonies of formal grammars. Since 2004, when P colonies were introduced in [1], many types of these systems have been developed. Basically, these are P colonies with different restrictions on the type of programs used (restricted, homogeneous) or working with an environment that is not only in the form of a multiset, but also a string or a 2D grid. Even P colonies whose environment changes dynamically have been introduced. The interested reader is referred to [2] for detailed information on membrane systems (P systems) and to [3] and [4] for more information to grammar systems theory. For more details on P colonies consult the surveys [5] and [6].

A basic P colony consists of a finite number of agents and their joint shared environment. The agents are formed from a finite number of objects and their functioning is based on programs consisting of rules. These rules are of two types: they may change the objects of the agents and they can be used for interacting with the shared environment of the agents. In the case of a basic P colony, the environment is a multiset of objects. The number of objects inside each agent is set by definition and it is usually a very small number: 1, 2 or 3. The environment is processed by the agents and it is used as a communication channel for the agents as well since through the environment, the agents can affect the behaviour of another agent.

The idea of shared knowledge lies in transferable programs. These are programs that are equipped with a condition that, when satisfied, allows the program to be transferred to the agent's program set or, conversely, from this set to the environment.

The agent's program set consists of two kinds of programs.

The first type can be named internal programs. These are the agent's essential programs and cannot be shared. This does not mean that such programs are unique in the P colony. They can be such acts of the agent that can be compared to the basic acts of living organisms such as respiration, digestion, and perception. Thus, they are functions that the organism knows by definition, does not have to learn, and cannot be taught to other organisms.

The second type, called transferable, is the opposite of the first type. It is the knowledge and abilities of an agent that can be shared with the environment under certain conditions.

The environment may also contain transferable programs. These are programs that allow agents to perform activities they did not know how to do - their program set did not contain such programs. These can be instructions on how to use various devices present in the environment, instructions contained in someone else's DNA, etc.

The paper is structured as follows: after the introductory section introducing readers to preliminaries and basic notions, there is a section on basic P colonies and their extension to transferable programs. The fourth section is devoted to shared programs in a 2D P colony environment. In conclusion, we summarize the content of the paper and outline future research goals for the behaviour of P colonies.

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 [7, 2].

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 $|w|_a$.

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia
lucie.ciencialova@fpf.slu.cz (L. Ciencialová);
ludek.cienciala@fpf.slu.cz (L. Cienciala)

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 $O = \{x_1, \ldots x_n\}$ 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.

3. Basic P colonies

The original concept of a P colony was introduced in [1] and presented in a developed form in [8, 9].

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

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

- *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;*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs, where each program consists of k rules, which are in one of the following forms each:*
    - *$a \to b$, $a, b \in A$, called an evolution rule;*
    - *$c \leftrightarrow d$, $c, d \in A$, called a communication rule;*
    - *r_1/r_2, called a checking rule; r_1, r_2 are both evolution rules or both communication rules.*

Now, we add brief explanations of the components of the P colony.

The first type of rules associated with the programs of the agents, the *evolution rules*, are of the form $a \to b$. This means that object a inside the agent is rewritten to (evolved to be) object b.

The second type of rules, the *communication rules*, are of the form $c \leftrightarrow d$. If a communication rule is performed, then object c inside the agent and object d in the environment swap their location. Thus, after executing the rule, object d appears inside the agent and object c is located in the environment.

The third type of rules is the checking rule. A checking rule is formed from two rules of one of the two previous types. If a checking rule r_1/r_2 is performed, then the rule r_1 has higher priority to be executed over the rule r_2. This means that the agent checks whether or not rule r_1 is applicable. If the rule can be executed, then the agent must use this rule. If rule r_1 cannot be applied, then the agent uses rule r_2.

The program determines the activity of the agent: the agent can change its state and/or the state of the environment.

The environment is represented by a finite number of copies of non-environmental objects and countably infinite copies of the environmental object e.

In every step, each object inside an agent is affected by the execution of a program. Depending on the rules in the program, the program execution may affect the environment as well.

The functioning of the P colony starts from its initial configuration (initial 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 $(w_1, \ldots, w_n, w_E)$, where $|w_i| = k$, $1 \leq i \leq n$, w_i represents all the objects present inside the i-th agent, and $w_E \in (A - \{e\})^*$ 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, only one agent at a time is allowed to use one of its programs (non-deterministically chosen). If the number of applicable programs for an agent is higher than one, then the agent non-deterministically chooses one of the programs.

A sequence of transitions is named a *computation.* A computation is halting if in the last configuration that is obtained there is no program that can be applied. 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, and the programs were formed from only two rules. Moreover, the initial configuration was defined as $(n + 1)$-tuple $(ee, \ldots, ee, \varepsilon)$ so at the beginning of the computation the environment of the P colony is “empty”, it is without input information.

Although P colonies are very simple computing devices, due to their (mainly parallel) working mode and distributed nature they demonstrate large expressive (computational) power. In most cases, computational completeness can be obtained with these constructs even with very few components and very few restrictions on the programs.

3.1. P Colonies with transferable programs

In this section, we focus on the definition of transferable programs. Under what conditions can a program be transferred into an agent, and under what conditions can an agent transfer a program into an environment?

A transferable program is an ordered pair (program, condition). We distinguish two kinds of conditions: 1. Object condition - for a program to be transferred, the destination must contain (or must not content) certain objects. A condition is specified as a set of multisets of objects. Each of the multisets has a size equal to the capacity P of the colony. 2. program condition - for a program to be transferred, the destination must contain (or not contain) a certain program.

Let Π be a P colony of capacity 2 with one agent and working with objects a, b, e, f. Examples of transferable programs are:

- $(\langle a \to b;\ a \leftrightarrow e\rangle\,;\{aa\})$ – when such program is placed in the environment an agent can consume it only if it contains two copies of object a; when such a program is placed in the agent it can be transferred into environment only if the environment contains at least two copies of object a;
- $(\langle a \to b;\ a \leftrightarrow e\rangle\,;\{\neg ee, \neg be\})$ – when such program is placed in the environment an agent can consume it only if it does not contain two copies of object e or one object b and one object e; such a program cannot be transferred into the environment. The conditions can never be met, the environment always contains environmental objects.
- $(\langle a \to b;\ a \leftrightarrow e\rangle\,;\{\langle e \to a;\ e \leftrightarrow a\rangle\})$ – this program can only be moved if program $\langle e \to a;\ e \leftrightarrow a\rangle$ is contained inside the recipient (in its program set if it is an agent, or directly in the environment if the program is to be moved there)

We call a transferable program (*program*; *condition*) *permanent* (denoted by the p in the subscript) if each time the program is moved to a different destination, one copy of the program remains at the original location.

Such programs can be considered knowledge sources (and in some cases object sources). Permanent transferable programs may allow an agent to produce an object that does not exist in the environment and that the agent could not otherwise obtain before getting the program.

Initial content of the environment is now defined as multiset over $A \cup \{$*set of transferable programs* $\}$.

Using transferable programs within a computation – In every step of a computation an agent can apply one of its applicable programs or transfer program in or out. It means that moving a program takes one step of computation as well as applying the program.

Example 1. *Let Π_1 be a P colony with one agent whose environment is a workshop with machines that enable the processing of the product. If the agent has (contains) raw materials that the machine can process, then the agent uses the machine. That is, it obtains a program from the environment that allows it to process the raw materials.*

P colony $\Pi_1 = \left(A, e, \boxed{tp}\,, v_e, B\right)$ is defined as follows:

$$
\begin{array}{lll}
A = \{ & \boxed{pow} & \text{piece of wood,} \\
 & \boxed{p} & \text{paint,} \\
 & \boxed{4n} & \text{four nuts,} \\
 & \boxed{s\text{-}b} & \text{double ended screw bolt,} \\
 & \boxed{l} & \text{leg,} \\
 & \boxed{lh} & \text{leg with hole,} \\
 & \boxed{lc} & \text{complete leg,} \\
 & \boxed{tt} & \text{table top,} \\
 & \boxed{tth} & \text{table top with holes,} \\
 & \boxed{ttn} & \text{table top with nuts,} \\
 & \boxed{tt1l} & \text{table top with 1 leg,} \\
 & \boxed{tt2l} & \text{table top with 2 legs,} \\
 & \boxed{tt3l} & \text{table top with 3 legs,} \\
 & \boxed{tc} & \text{complete table,} \\
 & \boxed{ts} & \text{sanded table,} \\
 & \boxed{tp} & \text{painted table,} \\
 & e\ \} &
\end{array}
$$

$$
\begin{array}{ll}
v_e = & \boxed{pow}\,\boxed{pow}\,\boxed{pow}\,\boxed{pow}\,\boxed{pow}\,\boxed{4n}\,\boxed{p}\,\boxed{s\text{-}b} \\
 & \boxed{s\text{-}b}\,\boxed{s\text{-}b}\,\boxed{s\text{-}b} \\
 & \left(\left\langle \boxed{pow} \to \boxed{tt}\,, e \leftrightarrow e\right\rangle, \{\boxed{pow}\, e\}\right) \quad \text{saw} \\
 & \left(\left\langle \boxed{pow} \to \boxed{l}\,, e \leftrightarrow e\right\rangle, \{\boxed{pow}\, e\}\right) \quad \text{wood lathe} \\
 & \left(\left\langle \boxed{tt} \to \boxed{tth}\,, e \leftrightarrow e\right\rangle, \{\boxed{tt}\, e\}\right) \quad \text{drill} \\
 & \left(\left\langle \boxed{l} \to \boxed{lh}\,, e \leftrightarrow e\right\rangle, \{\boxed{l}\, e\}\right) \quad \text{drill} \\
 & (\langle \boxed{tc} \to \boxed{ts}\,, e \leftrightarrow e\rangle, \{\boxed{tc}\, e\}) \quad \text{angle grinder} \\
 & \left(\left\langle \boxed{ts} \to \boxed{tp}\,, p \to e\right\rangle, \{\boxed{ts}\, p\}\right) \ \text{paint spray gun}
\end{array}
$$

Agent B is in the initial state ee and its set of programs is formed from its ability to work with or without hand tools:

$P = \{$

$\left\langle e \leftrightarrow \boxed{pow}, e \leftrightarrow e \right\rangle$, take a piece of wood

$\left\langle \boxed{lh} \to \boxed{lh}, e \leftrightarrow \boxed{s\text{-}b} \right\rangle$, take a screw bolt

$\left\langle \boxed{lh} \to \boxed{lc}, \boxed{s\text{-}b} \to e \right\rangle$, mount the screw bolt

$\left\langle \boxed{lc} \leftrightarrow, e \leftrightarrow e \right\rangle$, put down complete leg

$\left\langle \boxed{tth} \to \boxed{tth}, e \leftrightarrow \boxed{4n} \right\rangle$, take four nuts

$\left\langle \boxed{tth} \leftrightarrow \boxed{ttn}, \boxed{4n} \to e \right\rangle$, put nuts on the holes

$\left\langle e \leftrightarrow \boxed{ttn}, e \leftrightarrow \boxed{lc} \right\rangle$, take the table top
and complete leg

$\left\langle \boxed{ttn} \leftrightarrow \boxed{tt1l}, \boxed{lc} \to e \right\rangle$, mount the complete leg
on the table top

$\left\langle \boxed{tt1l} \leftrightarrow e, e \leftrightarrow e \right\rangle$, put down the table top
with one leg

$\left\langle e \leftrightarrow \boxed{tt1l}, e \leftrightarrow \boxed{lc} \right\rangle$, take the table top with leg
and the second complete leg

$\left\langle \boxed{tt1l} \leftrightarrow \boxed{tt2l}, \boxed{lc} \to e \right\rangle$, mount the second leg
on the table top

$\left\langle \boxed{tt2l} \leftrightarrow e, e \leftrightarrow e \right\rangle$, put down the table top
with two legs

$\left\langle e \leftrightarrow \boxed{tt2l}, e \leftrightarrow \boxed{lc} \right\rangle$, take the table top with
2 legs and the second complete leg

$\left\langle \boxed{tt2l} \leftrightarrow \boxed{tt3l}, \boxed{lc} \to e \right\rangle$, mount the third leg
on the table top

$\left\langle \boxed{tt3l} \leftrightarrow e, e \leftrightarrow e \right\rangle$, put down the table top
with three legs

$\left\langle e \leftrightarrow \boxed{tt3l}, e \leftrightarrow \boxed{lc} \right\rangle$, take the table top
and the third complete leg

$\left\langle \boxed{tt3l} \leftrightarrow \boxed{tc}, \boxed{lc} \to e \right\rangle$, mount the fourth leg
on the table top

$\langle \boxed{ts} \to \boxed{ts}, e \leftrightarrow p \rangle$, take the paint

$\left\langle \boxed{tp} \leftrightarrow e, e \leftrightarrow e \right\rangle$, put down the painted table

$\}$

The P colony starts an computation with all supplies in the environment. Agent contents two environmental objects. In this configuration agent can use only one program. This program allow it to consume one piece of wood.

$$(ee) \xrightarrow{\left\langle e \leftrightarrow \boxed{pow}, e \leftrightarrow e \right\rangle} \left(\boxed{pow}\ e\right)$$

The agent himself can't process a piece of marrow. There are two programs in the environment that (if the agent contains a piece of wood) can be transferred to the agent. These programs allow the agent to process pieces of wood into a table top or leg. we can imagine that the agent, by transferring the program, "picks up" a saw or uses a wood lathe.

$$\left(\boxed{pow}\ e\right) \xrightarrow[\left(\left\langle \boxed{pow} \to \boxed{tt}, e \leftrightarrow e \right\rangle, \{\boxed{pow}\ e\}\right)]{\text{import}} \left(\boxed{pow}\ e\right)$$

Now the agent can use this program to make the table top from piece of wood.

$$\left(\boxed{pow}\ e\right) \xrightarrow{\left(\left\langle \boxed{pow} \to \boxed{tt}, e \leftrightarrow e \right\rangle, \{\boxed{pow}\ e\}\right)} \left(\boxed{tt}\ e\right)$$

Because of non-determinism the agent can import the second transferable program instead of using the first one. In the next step, the agent can import a new program from the environment that allows it to drill holes in the table top. In the next step, the agent use imported program and then the agent move the table top with holes into the environment.

$$\left(\boxed{tt}\ e\right) \xrightarrow[\left(\left\langle \boxed{tt} \to \boxed{tth}, e \leftrightarrow e \right\rangle, \{\boxed{tt}\ e\}\right)]{\text{import}} \left(\boxed{tt}\ e\right)$$

$$\left(\boxed{tt}\ e\right) \xrightarrow{\left(\left\langle \boxed{tt} \to \boxed{tth}, e \leftrightarrow e \right\rangle, \{\boxed{tt}\ e\}\right)} \left(\boxed{tth}\ e\right)$$

$$\left(\boxed{tth}\ e\right) \xrightarrow{\left\langle \boxed{tth} \leftrightarrow e, e \leftrightarrow e \right\rangle} (ee)$$

Now, the agent is prepared for processing another piece of wood or mounting the nuts into the holes in the table top. The next scheme shows nuts assembly to the table top with holes.

$$(ee) \xrightarrow{\left\langle e \leftrightarrow \boxed{tth}, e \leftrightarrow \boxed{4n} \right\rangle} \left(\boxed{tth}\ \boxed{4n}\right)$$

$$\left(\boxed{tth}\ \boxed{4n}\right) \xrightarrow{\left\langle \boxed{tth} \to \boxed{ttn}, \boxed{4n} \to e \right\rangle} \left(\boxed{ttn}\ e\right)$$

$$\left(\boxed{ttn}\ e\right) \xrightarrow{\left\langle \boxed{ttn} \leftrightarrow e, e \leftrightarrow e \right\rangle} (ee)$$

The leg production is similar to the table top production. The agent consumes a piece of wood, imports program for leg making using wood lathe and changes the piece of wood into the table leg. Then the agent import program for hole making with drill and by applying it the agent obtain leg with hole. Then agent put the leg to the environment and consume it together with double ended screw bolt to assembly screw to hole in the leg. The agent can put complete leg into environment.

$$\left(\boxed{pow}\ e\right) \xrightarrow[\left(\left\langle \boxed{pow} \to \boxed{l}, e \leftrightarrow e \right\rangle, \{\boxed{pow}\ e\}\right)]{\text{import}} \left(\boxed{pow}\ e\right)$$

$$\left(\boxed{pow}\ e\right) \xrightarrow{\left(\left\langle \boxed{pow}\rightarrow\boxed{l},e\leftrightarrow e\right\rangle,\{\boxed{pow}\ e\}\right)} \left(\boxed{l}\ e\right)$$

$$\left(\boxed{l}\ e\right) \xrightarrow[\left(\left\langle \boxed{l}\rightarrow\boxed{lh},e\leftrightarrow e\right\rangle,\{\boxed{l}\ e\}\right)]{\text{import}} \left(\boxed{l}\ e\right)$$

$$\left(\boxed{l}\ e\right) \xrightarrow{\left(\left\langle \boxed{l}\rightarrow\boxed{lh},e\leftrightarrow e\right\rangle,\{\boxed{l}\ e\}\right)} \left(\boxed{lh}\ e\right)$$

$$\left(\boxed{lh}\ e\right) \xrightarrow{\left\langle \boxed{lh}\leftrightarrow e,e\leftrightarrow e\right\rangle} (ee)$$

$$(ee) \xrightarrow{\left\langle e\leftrightarrow\boxed{lh},e\leftrightarrow\boxed{s\text{-}b}\right\rangle} \left(\boxed{lh}\ \boxed{s\text{-}b}\right)$$

$$\left(\boxed{lh}\ \boxed{s\text{-}b}\right) \xrightarrow{\left\langle \boxed{lh}\rightarrow\boxed{lc},\boxed{s\text{-}b}\rightarrow e\right\rangle} \left(\boxed{lc}\ e\right)$$

$$\left(\boxed{lc}\ e\right) \xrightarrow{\left\langle \boxed{lc}\leftrightarrow e,e\leftrightarrow e\right\rangle} (ee)$$

The agent can make the legs and the table tops up to a total quantity of five. Only such computations where agents makes one table top and four legs leads to successful end - production of one painted table.

$$(ee) \xrightarrow{\left\langle e\leftrightarrow\boxed{ttn},e\leftrightarrow\boxed{lc}\right\rangle} \left(\boxed{ttn}\ \boxed{lc}\right)$$

$$\left(\boxed{ttn}\ \boxed{lc}\right) \xrightarrow{\left\langle \boxed{ttn}\rightarrow\boxed{tt1l},\boxed{lc}\rightarrow e\right\rangle} \left(\boxed{tt1l}\ e\right)$$

$$\left(\boxed{tt1l}\ e\right) \xrightarrow{\left\langle \boxed{tt1l}\leftrightarrow e,e\leftrightarrow e\right\rangle} (ee)$$

$$(ee) \xrightarrow{\left\langle e\leftrightarrow\boxed{tt1l},e\leftrightarrow\boxed{lc}\right\rangle} \left(\boxed{tt1l}\ \boxed{lc}\right)$$

$$\left(\boxed{tt1l}\ \boxed{lc}\right) \xrightarrow{\left\langle \boxed{tt1l}\rightarrow\boxed{tt2l},\boxed{lc}\rightarrow e\right\rangle} \left(\boxed{tt2l}\ e\right)$$

$$\left(\boxed{tt2l}\ e\right) \xrightarrow{\left\langle \boxed{tt2l}\leftrightarrow e,e\leftrightarrow e\right\rangle} (ee)$$

$$(ee) \xrightarrow{\left\langle e\leftrightarrow\boxed{tt2l},e\leftrightarrow\boxed{lc}\right\rangle} \left(\boxed{tt2l}\ \boxed{lc}\right)$$

$$\left(\boxed{tt2l}\ \boxed{lc}\right) \xrightarrow{\left\langle \boxed{tt2l}\rightarrow\boxed{tt3l},\boxed{lc}\rightarrow e\right\rangle} \left(\boxed{tt3l}\ e\right)$$

$$\left(\boxed{tt3l}\ e\right) \xrightarrow{\left\langle \boxed{tt3l}\leftrightarrow e,e\leftrightarrow e\right\rangle} (ee)$$

$$(ee) \xrightarrow{\left\langle e\leftrightarrow\boxed{tt3l},e\leftrightarrow\boxed{lc}\right\rangle} \left(\boxed{tt3l}\ \boxed{lc}\right)$$

$$\left(\boxed{tt3l}\ \boxed{lc}\right) \xrightarrow{\left\langle \boxed{tt3l}\rightarrow\boxed{ttc},\boxed{lc}\rightarrow e\right\rangle} \left(\boxed{ttc}\ e\right)$$

During the computation, the agent can export programs that allow the processing of a piece of wood - if there is a piece of wood in the environment the program can be transferred. After processing the last piece of wood, the program remains in the agent's set of programs.

When the agent contains the complete table, it can import program that allows agent to use angle grinder and proceed from the complete table to the sanded table. Then the agent can take the paint and import the last program to spray the table with paint using a paint spray gun.

$$\left(\boxed{tc}\ e\right) \xrightarrow[\left(\left\langle \boxed{tc}\rightarrow\boxed{ts},e\leftrightarrow e\right\rangle,\{\boxed{tc}\ e\}\right)]{\text{import}} \left(\boxed{tc}\ e\right)$$

$$\left(\boxed{tc}\ e\right) \xrightarrow{\left(\left\langle \boxed{tc}\rightarrow\boxed{ts},e\leftrightarrow e\right\rangle,\{\boxed{tc}\ e\}\right)} \left(\boxed{ts}\ e\right)$$

$$\left(\boxed{ts}\ e\right) \xrightarrow{\left\langle \boxed{ts}\rightarrow\boxed{ts},e\leftrightarrow p\right\rangle} (ee)$$

$$\left(\boxed{ts}\ p\right) \xrightarrow[\left(\left\langle \boxed{ts}\rightarrow\boxed{tp},p\rightarrow e\right\rangle,\{\boxed{ts}\ e\}\right)]{\text{import}} \left(\boxed{ts}\ e\right)$$

$$\left(\boxed{ts}\ e\right) \xrightarrow{\left(\left\langle \boxed{ts}\rightarrow\boxed{tp},p\rightarrow e\right\rangle,\{\boxed{ts}\ e\}\right)} \left(\boxed{tp}\ e\right)$$

The last executed program is to put painted table into environment.

$$\left(\boxed{tp}\ e\right) \xrightarrow{\left\langle \boxed{tp}\leftrightarrow e,e\leftrightarrow e\right\rangle} \left(\boxed{e}\ e\right)$$

If we add a special object to the environment, which will be part of the condition for transfer in all transferable programs, the agent can end computation only when it "returns" all transferred programs back to the environment.

4. 2D P Colonies

In [10] a new model, called 2D P colony was introduced. As in the original model, the P colony is of capacity two and the agents are equipped with sets of the programs formed from rules – communication and evolution. The main change is in the environment. The authors put the agents into the 2D grid of square cells and they provide the agent with the possibility to move – the motion program. The direction of the movement of the agent is determined by the contents of cells surrounding the cell in which the agent is placed. The motion program can contain one motion rule and one evolution rule.

Definition 2. *A 2D P colony is a construct*

$$\Pi = (A, e, Env, B_1, \ldots, B_k, f), k \geq 1, \text{ where}$$

- *A is an alphabet of the colony, its elements are called objects,*
- *$e \in A$ is the basic environmental object of the 2D P colony,*
- *Env is a pair $(m \times n, w_E)$, where $m \times n, m, n \in N$ is the size of the environment and w_E is the initial contents of environment, it is a matrix of size $m \times n$ of multisets of objects over $A - \{e\}$.*
- *B_i, $1 \leq i \leq k$, are agents, each agent is a construct $B_i = (o_i, P_i, [o, p])$, $0 \leq o \leq m$, $0 \leq p \leq n$, where*
  - *o_i is a multiset over A, it determines the initial state (contents) of the agent, $|o_i| = 2$,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,l_i}\}, l \geq 1, 1 \leq i \leq k$ is a finite set of programs, where each program contains exactly 2 rules, which are in one of the following forms each:*
    - *$a \to b$, called the evolution rule, $a, b \in A$,*
    - *$c \leftrightarrow d$, called the communication rule, $c, d \in A$,*
    - *$[a_{q,r}] \to s, 0 \leq q, r \leq 2, s \in \{\Leftarrow, \Rightarrow, \Uparrow, \Downarrow\}$, called the motion rule,*
- *$f \in A$ is the final object of the colony.*

A configuration of the 2D P colony is given by the state of the environment - matrix of type $m \times n$ with multisets of objects over $A - \{e\}$ as its elements, and by the state of all agents - pairs of objects from alphabet A and the coordinates of the agents. An initial configuration is given by the definition of the 2D P colony.

A computational step consists of three parts. The first part lies in determining the set of applicable programs according to the current configuration of the 2D P colony. In the second part, we have to select from this set one program for each agent, in such a way that there is no collision between the communication rules belonging to different programs. The third part is the execution of the chosen programs.

A change of the configuration is triggered by the execution of programs and it involves changing the state of the environment, contents and placement of the agents.

A computation is non-deterministic and maximally parallel. The computation ends by halting when there is no agent with applicable program.

The result of the computation is the number of copies of the final object placed in the environment at the end of the computation.

The aim of introducing 2D P colonies is not studying their computational power but monitoring their behaviour during the computation.

4.1. 2D P Colonies with transferable programs

We can introduce transferable programs into 2D P colonies model.

Aspects of transferable programs composed of evolutionary and communication rules have already been mentioned in the previous section.

A transferable motion program can upgrade an agent's motion capabilities. For example, an agent that previously could only move in one direction now has the ability to move left, right, or backwards. The agent's program

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Uparrow;\ a \to a \right\rangle$$

The transferred program

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ a \to a \right\rangle$$

If the agent had any restrictions in the original motion programs under which it could move in a certain direction, these restrictions can be changed. For example, if the agent could only move to the left if there was an object L on the left, then after importing the program, it will also be able to move to the left if there is an object T to the left of the agent. The agent's program

$$\left\langle \begin{bmatrix} e & e & e \\ L & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ a \to a \right\rangle$$

The transferred program

$$\left\langle \begin{bmatrix} e & e & e \\ T & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ a \to a \right\rangle$$

Similarly, constraints can be imposed on the agent's content. The agent may be given "one-time permission" to turn left in the form of object L, which it will be able to rewrite L to another object while according to the agent's movement rule agent moves left.

The transferred program

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ L \to a \right\rangle$$

Example 2. *Consider a 2D P colony with two agents* $\Pi_2 = (A, e, Env, B_1, B_2, f)$. *The first agent can move around the environment and spread the transferable program allowing the agent to consume slave object. The second agent's programs allows the agent to move only if slave object is inside the agent. If the second agent enters special position in the environment it can import a program that enables the agent to make its own object* s. *The agent that imports such a program will then not have to follow the first agent.*

The environment has a size 5×5 and each cell contains only environmental objects except for the cell with address $[4, 4]$, which also contains transferable program

$$(\langle e \to s;\ e \to e\rangle\ ,\ \{ee\})\ .$$

The first agent $B_1 = (eF, P_1, [0, 0])$ has a set of programs P_1 that include the following programs:

$$(\langle e \to e;\ e \leftrightarrow s\rangle\ ,\ \{ee\})_p$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Rightarrow;\ F \to s \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ F \to s \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Uparrow;\ F \to s \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Downarrow;\ F \to s \right\rangle$$

$$\langle e \to F;\ s \leftrightarrow e\rangle$$

The first agent B_2 is defined as $(ee, P_2, [2, 2])$. Its set of programs is formed by following programs:

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Rightarrow;\ s \to e \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Leftarrow;\ s \to e \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Uparrow;\ s \to e \right\rangle$$

$$\left\langle \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix} \to \Downarrow;\ s \to e \right\rangle$$

The first agent starts the computation in the state eF with two options - 1. export the transferable program, or 2. apply one of the motion programs. Executing the motion program changes the agent's position and also changes its state to es. In the following step, the agent can also export a program or apply a program $\langle e \to F;\ s \leftrightarrow e\rangle$. By executing the program, the agent places the slave object s in the cell and changes its state to eF.

The second agent is in state ee at the beginning of the computation. It has no applicable program until the first agent visits its position and places transferable program $(\langle e \to e;\ e \leftrightarrow s\rangle\ ,\ \{ee\})$ there. Thus, the movement of agent B_2 is thus dependent on this program and the presence of object s in the environment.

If agent B_2 reaches cell $[4, 4]$ during the computation, it can import a program whose application can evolve object s from the environmental object and thus becomes independent of the presence of object s in the environment.

Example 3. *Imagine a student moving within the school building to acquire the necessary knowledge and pass an exam. We can simulate such a situation by using a 2D P colony.*

The single cells of the 2D environment represent parts of a school building (or multiple buildings) such as classrooms, laboratories, libraries, teachers' offices, etc. The student's goal is to obtain a grade in the course (object M - mark). At the beginning of the computation, agent – student can contain either only environmental symbols or also an object that expresses his intention to successfully complete the course (object m). Its movement through environment can be random (the agent can use any of movement program, it can move in any direction)

There may be other agents in the environment such as teachers who will move from their office to the classroom or lab and back again. If they encounter a student in these rooms (the student places their marker in the environment), they can provide a program that allows the student to advance closer to obtaining the M object. Let to get the grade require attending a class, gaining knowledge by studying in the library and lab, and finally passing an exam. Thus, object m can be acquired over time by a subscript that captures the facts of the actions taken (T - teacher, L - library, A - lab, C - consultation). We can determine under which conditions it is possible to change the index just with the help of transferable programs that are available in different rooms - cells.

We leave it to the esteemed reader to construct particular programs for student and teacher.

5. Conclusion

In this paper we introduced the notion of transferable programs in two models of P colonies - basic P colonies and 2D P colonies. One of the main features of P colonies is the use of environment agents to store shared objects. Thus, objects can be used by all agents that are in a given part of the environment and know how to handle the object (have programs that process the object). Transferable programs allow not only to share objects, but also to share the ability to handle these objects.

Our goal for the future is to introduce a type of numerical P colony with transferable programs, whose agents would acquire their abilities by moving around in an environment that has a specified structure. Agents in such a P colony would perform search algorithms known, for example, from graph theory.

Acknowledgments

This work was supported partially by European Union under European Structural and Investment Funds Operational Programme Research, Development and Education project Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje, CZ.02.2.69/0.0/0.0/18_058/0010238 and by the Silesian University in Opava under the Student Funding Scheme, project SGS/8/2022.

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] Gh. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, Inc., New York, NY, USA, 2010.

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

[4] E. Csuhaj-Varjú, J. Kelemen, Gh. Păun, J. Dassow, Grammar Systems: A Grammatical Approach to Distribution and Cooperation, 1st ed., Gordon and Breach Science Publishers, Inc., USA, 1994.

[5] A. Kelemenová, P Colonies, in: Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, 1st. ed., Oxford University Press, Inc., New York, NY, USA, 2010, pp. 584–593.

[6] L. Ciencialová, E. Csuhaj-Varjú, L. Cienciala, P. Sosík, P colonies, Journal of Membrane Computing 1 (2019) 178–197. URL: https://doi.org/10.1007/s41965-019-00019-w. doi:10.1007/s41965-019-00019-w.

[7] G. Rozenberg, A. Salomaa, Handbook of Formal Languages: Beyonds words, Handbook of Formal Languages, Springer, 1997. URL: https://books.google.hu/books?id=voLFxNxAHdkC.

[8] J. Kelemen, A. Kelemenová, On P colonies, a biochemically inspired model of computation, in: Proc. of the 6^{th} International Symposium of Hungarian Researchers on Computational Intelligence, Budapest TECH, Hungary, 2005, pp. 40–56. URL: http://conf.uni-obuda.hu/mtn2005/Kelemen.pdf.

[9] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil, Computing with cells in environment: P colonies, Journal of Multiple-Valued Logic and Soft Computing 12 (2006) 201–215.

[10] L. Cienciala, L. Ciencialová, M. Perdek, 2D P Colonies, in: Proceedings of the 13th International Conference on Membrane Computing, CMC'12, Springer-Verlag, Berlin, Heidelberg, 2012, p. 161–172. URL: https://doi.org/10.1007/978-3-642-36751-9_12. doi:10.1007/978-3-642-36751-9_12.