P Colonies and Reaction Systems

Lucie Ciencialová[1], Luděk Cienciala[1], and Erzsébet Csuhaj-Varjú[2]

[1] Institute of Computer Science, Silesian University in Opava, Czech Republic
`lucie.ciencialova@fpf.slu.cz`
`ludek.cienciala@fpf.slu.cz`

[2] Department of Algorithms and Their Applications, Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary
`csuhaj@inf.elte.hu`

Summary. P colonies are abstract computing devices modeling communities of very simple reactive agents living and acting in a joint shared environment which is given with a multiset of objects. Reaction systems were proposed as a computing device, components of which represent basic chemical reactions that take place in shared environment given with a set. Although P colonies operate with multisets of objects and reaction systems work with sets, the two models can be related. In this paper, we construct a P colony simulating interactive processes in a reaction system.

P colonies were introduced in [7] as a variant of very simple membrane systems (P systems), inspired by so-called colonies of formal grammars. (For more information on P systems the interested reader is referred to [9], for P colonies to [1], and for grammatical model colony to [8].) A P colony is formed from agents and their shared environment. Each agent is represented by a multiset of objects in a membrane and the environment is given by multiset of objects as well. Agents are equipped with programs composed from rules, the rules are applied to (multisets 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 of the P colony is used as communication channel for agents. Through the environment, the agents are able to affect the behavior of another agent.

The rules of the agents can be rewriting, communication or checking rules; these three rule types were introduced in [7]. Rewriting rule $a \to b$ allows agent to rewrite (evolve) one object a to object b. Both objects are placed inside the agent. Communication rule $a \leftrightarrow b$ provides the possibility to exchange object c placed inside the agent and object d in the environment. A checking rule is formed of two rules r_1, r_2 which are of type rewriting or communication. Checking feature sets some kind of priority between rules r_1, r_2. The agent tries to apply the first rule and if it cannot be performed, the agent executes the second rule. The agents of P colonies work in a maximally parallel manner, i.e., a maximal number of enabled agent performs one of its applicable program in parallel. An agent is enabled in a computation step if it is able to apply one of its programs. If an agent is not enabled, then its objects remain unchanged. For detailed information on operation of P colonies, see [1].

Reaction systems (R systems, for short) are computational models of another type. The model was introduced in [4] as a computing device, components of which are a simile for basic chemical reactions. Roughly speaking, a reaction system is composed of a finite set of objects that can be considered as chemicals and a finite set of reactions. Each reaction is a triplet of sets: reactants, inhibitors and products. Let T be a set of reactants. A reaction is applied if all reactants are present in T, and there are no inhibitors; then reactants are replaced by the products. All enabled reactions are applied in parallel. The final set of products is the union of all single sets of products of each reaction which is enabled in T. The reader might notice that a reaction can also be considered as an action of an agent.

It is easy to observe that P colonies and R systems have similarities: both of them can be seen as multi-agent systems of very simple reactive agents. However, there are significant differences between them. Namely, P colonies work with finite multisets of objects and R systems operate with finite sets. Furthermore, in case of P colonies, those objects that do not take part in any action at a computation step, remain unchanged, but in case of R systems disappear from the available objects.

It is an intriguing question whether or not P colonies and R systems can be related. In this paper, we focus on construction of P colony that can simulate interactive processes in given reaction system. The paper is structured as follows: The second section is devoted to definitions and notations used in the paper. The third section contains construction of P colony and the paper concludes with possibilities of future work.

1 Definitions

Throughout the paper we assume the reader to be familiar with basics of formal language theory. We introduce notions and notations used in the sequel.

We use $\mathbb{N}\cdot\mathsf{RE}$ to denote the family of recursively enumerable sets of natural numbers and $\mathbb{N}$ to denote the set of natural numbers.

Σ is a notation for the alphabet. Let Σ^* be set of all words over alphabet Σ (including empty word ε). For the length of the word $w \in \Sigma^*$ we use notation $|w|$ and the number of occurrences of symbol $a \in \Sigma$ in w is denoted by $|w|_a$.

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \to N$; f assigns to each object in V its multiplicity in M. The set of all multisets over the set of objects V is denoted by V^*. The set V' is called the support of M and denoted by $supp(M)$ if for all $x \in V'$ $f(x) \neq 0$. The cardinality of M, denoted by $card(M)$, is defined by $card(M) = \sum_{a\in V} f(a)$. Any multiset of objects M with the set of objects

$V = \{a_i, \dots\ a_n\}$ can be represented as a string w over alphabet V with $|w|_{a_i} = f(a_i);\ 1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent M, and ε represents the empty multiset.

1.1 P Colonies

The original concept of a P colony was introduced in [7] and presented in a developed form in [6, 2].

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

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

- *V is an alphabet, its elements are called objects;*
- *$e \in V$ is the basic (or environmental) object of the colony;*
- *$f \in V$ 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 V 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:*
    - *$a \to b$, $a, b \in V$, called an evolution rule;*
    - *$c \leftrightarrow d$, $c, d \in V$, called a communication rule;*
    - *r_1/r_2, called a checking rule; r_1, r_2 are both evolution rules or both communication rules.*

We add some brief explanations to the components of the P colony.

The first type of rules associated to 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 are the checking rules. 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.

We note that these types of rules are the basic ones; in some variants of P colonies other types of rules have been also considered.

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 (zero included) of copies of non-environmental objects and a 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. This interaction between the agents and the environment is the key factor of the functioning of the P colony.

An initial configuration of P colony is $(n+1)$-tuple $(o_1, \ldots, o_n, v_E)$ of the multisets of objects placed in P colony at the beginning of the computation, where o_i ($1 \leq i \leq n$) is the content of the agent B_i and v_E is the multiset of object in the environment different from e. In general, the configuration of the P colony Π is defined as $(n+1)$-tuple $(w_1, \ldots, w_n, w_E)$, where w_i represents all objects inside of i-th agent, $|w_i| = c$, $1 \leq i \leq n$, $w_E \in (V - \{e\})^*$ is composed by objects different from e placed in the environment.

At each step of the (parallel) computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, the agent non-deterministically chooses one of them. At one step of computation, the maximal possible number of agents have to be active, i.e., have to perform a program.

By applying programs, the P colony passes from one configuration to another configuration. A sequence of configurations started from the initial configuration is called a computation. A configuration is halting if the P colony has no applicable program. With halting computation the result is associated and it is the number of copies of final object placed in the environment in halting configuration.

$$N(\Pi) = \{|w_E|_f \mid (o_1, \ldots, o_n, v_E) \Rightarrow^* (w_1, \ldots, w_n, w_E)\},$$

where $(o_1, \ldots, o_n, v_E)$ is the initial configuration, $(w_1, \ldots, w_n, w_E)$ is the final configuration, and $\Rightarrow^*$ denotes reflexive and transitive closure of $\Rightarrow$.

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 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 sequential mode is denoted by $NPCOL_{seq}K(c, n, h)$; whereas the corresponding families of P colonies working in the maximally parallel way are denoted by $NPCOL_{par}K(c, n, h)$. If one of the parameters n or h is not bounded, then we replace it with $*$. If only P colonies using programs without checking rules are considered, then we omit the K. If numerical parameter is unbounded, we denote it by a $*$.

1.2 Reaction Systems

In the following we briefly summarize the basic notions concerning reaction systems (R systems), introduced in [4].

Let S be an alphabet (its elements are called molecules, or only symbols).

Definition 2. *A reaction is a triple $a = (R, I, P)$ such that R, I, P are finite non-empty sets with $R \cap I = \emptyset$.*

R is the reactant set of a,I is the inhibitor set of a, and P is the product set of a; R, I, P are also denoted as R_a, I_a, P_a, respectively. We denote by $rac(S)$ the set of all reactions in S. The set S is usually called background set. In some papers the definition is altered in such a way that set of inhibitors can be empty set.

Definition 3. *A reaction system is an ordered pair $\mathcal{A} = (S, A)$, where S is a background set and A is a nonempty finite subset of $rac(S)$.*

To describe the effect of a set of reactions on a state, we first define the effect of a single reaction.

Definition 4. *Let S be a background set, let $X \subseteq S$, and let $a \in rac(S)$. Then a is enabled by X, denoted by $en_a(X)$, if $R_a \subseteq X$ and $I_a \cap X = \emptyset$. The result of a on X, denoted by $res_a(X)$, is defined by $res_a(X) = P_a$ if $en_a(X)$, and $res_a(X) = \emptyset$. otherwise.*

The effect of a set of reactions on a state is cumulative, defined as follows.

Definition 5. *Let S be a background set, let $X \subseteq S$, and let $A \subseteq rac(S)$. The set of reactions enabled by X is denoted by $en(A, X)$ and it is defined by $en(A, X) = \{a \in A \mid en_a(X)\}$. The result of A on X, denoted by $res(A, X)$, is defined by $res(A, X) = \{res_a(X) \mid a \in A\}$. The set of reactions that can generate X, denoted by $prod(A, X)$, is defined as $prod(A, X) = \{a \in A \mid P_a \subseteq X\}$.*

The dynamic behavior of the reaction systems is captured by the notion of an interactive process.

Definition 6. *Let $\mathcal{A} = (S, A)$ be a reaction system. An interactive process in $\mathcal{A}$ is a pair $\pi = (\gamma, \varphi)$ of finite sequences such that $\gamma = C_0, C_1, \ldots, C_{n-1}, \varphi = D_1, \ldots, D_n$ with $n \geq 1$, where $C_0, \ldots, C_{n-1}, D_1, \ldots, D_n \subseteq S$, $D_1 = res(A, C_0)$, and $D_i = res(A, D_{i-1} \cup C_{i-1})$ for each $2 \leq i \leq n$.*

The sequences $C_0, \ldots, C_{n-1}$ and $D_1, \ldots, D_n$ are the context and result sequences of π, respectively. Context C_0 represents the initial state of π (the state in which the interactive process is initiated), and the contexts $C_1, \ldots, C_{n-1}$ represent the influence of the environment to the computation. It should be noticed that the context sequence $\gamma = C_0, C_1, \ldots, C_{n-1}$ is described by a regular expression over S. The sequence $sts(\pi) = W_0, \ldots, W_n$ denotes the state sequence of π, where $W_0 = C_0$ (the initial state), and $W_i = D_i \cup C_i$ for all $1 \leq i \leq n$. The sequence $act(\pi) = E_0, \ldots, E_{n-1}$ of subsets of A such that $E_i = en(A, W_i)$ for all $0 \leq i \leq n-1$ represents the activity sequence of π. Thus, the evolution can be written as

$$W_0 \xrightarrow{E_0} W_1 \xrightarrow{E_1} \cdots \xrightarrow{E_{n-1}} W_n.$$

If $E_n = en(A, W_n) = \emptyset$ then interactive process terminates.

One step of evolution – the evolution from state W_i to state W_{i+1} can be seen as transition mapping $\delta : 2^S \times 2^S \to 2^S$ defined as

$$\delta(D_i, C_i) = D_{i+1}$$

iff there exists (just one) set $E_i \subseteq A$ such that $E_i = en(A, D_i \cup C_i)$ and $D_{i+1} = res(E_i)$. For the first step of an evolution, there is transition mapping defined as $\delta(\emptyset, C_0) = D_1$.

In some sources the set of inhibitors can be empty.

2 Dynamical Behavior: P Colonies versus R Systems

Let us examine the dynamical changes of the environment of a P colony in the course of the computation. From this point of view we can find some similarities between P colonies and R systems. For example, it is easy to see that the change of the support of the environment (i.e. the support of the finite multiset of non-enviromental symbols forming the environment), resembles to a reaction or several reactions performed by a reaction system. The agents of the P colony can exchange object(s) with the environment in a way similar to actions of reactions in R systems. The exchange rule of the P colony must be enabled - reactants must be included in the environment - and then products will occur in the environment after performing reaction. We can also find some differences in behavior of these two computing devices. The first difference is in the number of active components. In R systems, all enabled reactions are executed while in P colonies, the number of active agents is limited to the number of objects that are placed in the environment. Furthermore, objects not used in any action of the P colony remain unchanged and available for further steps, but in case of reaction systems these objects disappear from the system. One other significant difference between P colony and R system is that P colony operates with finite multisets and R system with finite sets of objects.

In the following we construct a P colony which simulates the behavior of an R system. Notice that a set as a notion is different from a multiset of objects where each object appears only in (at most) one copy, however for our purposes it is enough to define a P colony where objects of certain type appear in the environment only in at most one copy during operation.

Thus, for given R system $\mathcal{A} = (S, A)$ and sequence of inputs $i_0, i_1, \ldots, i_n$ we can construct P colony $\Pi = (V, e, f, v_E, B_1, \ldots, B_n)$ that simulates interactive processes of $\mathcal{A}$.

Instead of the formal statement and its proof, we provide the description of the simulation steps and the main parts of the construction. In addition, we develop an example that helps in the easier understanding.

One step of an interactive process in $\mathcal{A}$ is simulated in five phases in Π:

1. Input Generation
2. Input multiplication
3. Simulation of reactions
4. Consuming phase

For each phase, we construct subset of agents that are responsible of executing corresponding phase. Let k be the maximum of the number of reactants, l the maximum of the number of inhibitors, and m the maximum number of products associated with one reaction of the reactions given by R system $\mathcal{A}$.

1. Generate Input

In this phase the objects (symbols) in the input set are generated. For this phase, we construct agents called i-agents. They generate input symbols in one step. Obviously, the number of i-agents is $|S|$. After generation of current input symbols, i-agents enter a waiting phase, by generating auxiliary objects to wait for the exact number of steps until then they generate another input. The set of programs of i-agent corresponding to symbol a_j is formed from following programs:

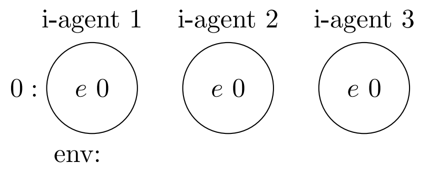
$$\begin{array}{ll}
\left\langle e \leftrightarrow a_j'' \ / \ e \leftrightarrow e; i \rightarrow i_o \right\rangle & \\
\left\langle a_j'' \rightarrow a_j; i_o \rightarrow i' \right\rangle & \\
\langle e \rightarrow a_j; i_o \rightarrow i_o' \rangle & \text{if } a_j \in C_i;\ i \geq 0 \\
\langle e \rightarrow e; i_o \rightarrow i_o' \rangle & \text{if } a_j \notin C_i;\ i \geq 0 \\
\langle a_j \leftrightarrow e; i_o' \rightarrow i' \rangle & \\
\langle e \leftrightarrow e; i_o' \rightarrow i' \rangle &
\end{array}$$

These programs are for generation of current input; the following programs are for synchronization of agents.

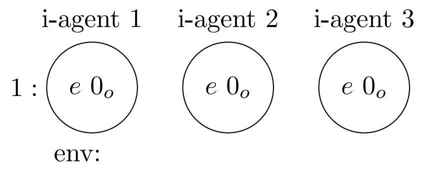
$$\begin{array}{ll}
\langle e \leftrightarrow F; i' \rightarrow i'' \rangle & \\
\langle F \rightarrow F_1; i'' \rightarrow i'' \rangle & \\
\langle F_x \rightarrow F_{x+1}; i'' \rightarrow i'' \rangle & \text{for } 1 \leq x < 3 + 2k + 2m \\
\langle F_y \rightarrow D; i'' \rightarrow i_D'' \rangle & y = 3 + 2k + 2m \\
\langle D \leftrightarrow e; i'' \rightarrow i_D \rangle & \\
\langle e \rightarrow F_y; i_D \rightarrow i'' \rangle & y = 5 + 2k + 2m + 1 \\
\langle F_z \rightarrow F_{z+1}; i'' \rightarrow i'' \rangle & 5 + 2k + 2m + 1 \leq z < 2 + 2k + 2m + 4|M| \\
\langle F_u \rightarrow E; i'' \rightarrow i_E \rangle & u = 2 + 2k + 2m + 4|A| \\
\langle E \leftrightarrow e; i_E \rightarrow i_E \rangle & \\
\langle e \rightarrow e; i_E \rightarrow (i+1) \rangle &
\end{array}$$

To help the easier understanding, we demonstrate the following example.

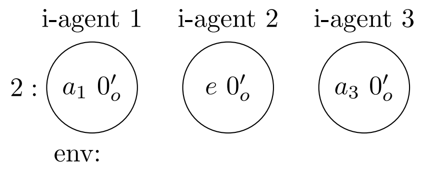
Let $\mathcal{A} = (S, A)$ be reaction system with $S = \{a_1, a_2, a_3\}$ and $A = \{r_1 : (\{a_2\}, \emptyset, \{a_2\});\ r_2 : (\{a_1, a_3\}, \{a_2\}, \{a_1, a_2\});\ r_3 : (\{a_3\}, \{a_1\}, \{a_1, a_2\})\}$. Let $C_0 = \{a_1, a_3\}$ be the input. The i-agents are initialized as follows:



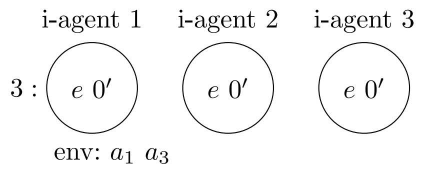
They execute the first program $\langle e \leftrightarrow a''_j \ / \ e \leftrightarrow e; i \to i_o \rangle$.



In this configuration, the i-agents have applicable programs that can generate objects corresponding to symbols from C_0.



After performing programs $\langle a_j \leftrightarrow e; 0'_o \to 0' \rangle$ (i-agent 1 and 3) or $\langle e \leftrightarrow e; 0'_o \to 0' \rangle$ (i-agent 2). All symbols in C_0 are placed in the environment.



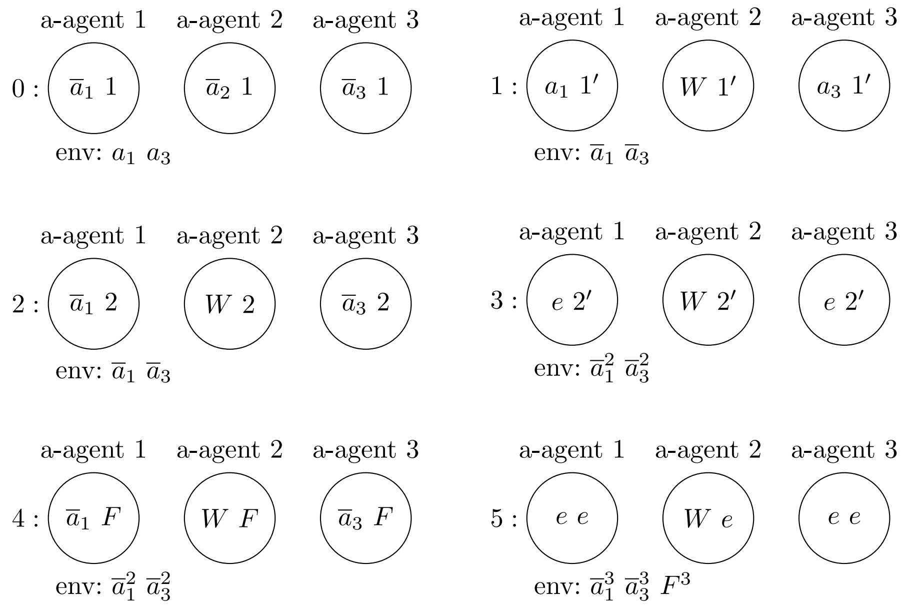
All three agents wait until object F appears in the environment.

2. Multiply Input

This phase is to multiply the input symbols to be "accessible" for every agent that simulates enabled reaction. Notice that in case of reaction systems all enabled reactions should be performed in parallel. We construct a-agents that generate $|A|$ symbols that appear in the environment after the first phase. The programs for this phase are:

$$\begin{array}{lll}
\langle \overline{a}_j \leftrightarrow a_j \ / \ \overline{a}_j \to W;\ 1 \to 1' \rangle & & \\
\langle a_j \to \overline{a}_j;\ 1' \to 2 \rangle & & \\
\langle \overline{a}_j \leftrightarrow e;\ i \to i' \rangle & \langle W \to W;\ i \to i' \rangle & 1 \le i \le |A|-1 \\
\langle e \to \overline{a}_j;\ i' \to (i+1) \rangle & \langle W \to W;\ i' \to (i+1) \rangle & 1 \le i < |A|-1 \\
\langle e \to \overline{a}_j;\ i' \to F \rangle & \langle W \to W;\ i' \to F \rangle & i = |A|-1 \\
\langle \overline{a}_j \leftrightarrow e;\ F \leftrightarrow e \rangle & \langle W \to W;\ F \leftrightarrow e \rangle & \\
\langle e \to \overline{a}_j;\ e \to 1 \rangle & \langle W \to \overline{a}_j;\ e \to 1 \rangle &
\end{array}$$

The number of a-agents is $|S|$. In the previously given example the second phase of simulation can be depicted as follows:


a-agent 1 a-agent 2 a-agent 3
$0: \overline{a}_1\ 1$ $\overline{a}_2\ 1$ $\overline{a}_3\ 1$
env: $a_1\ a_3$
a-agent 1 a-agent 2 a-agent 3
$1: a_1\ 1'$ $W\ 1'$ $a_3\ 1'$
env: $\overline{a}_1\ \overline{a}_3$
a-agent 1 a-agent 2 a-agent 3
$2: \overline{a}_1\ 2$ $W\ 2$ $\overline{a}_3\ 2$
env: $\overline{a}_1\ \overline{a}_3$
a-agent 1 a-agent 2 a-agent 3
$3: e\ 2'$ $W\ 2'$ $e\ 2'$
env: $\overline{a}_1^2\ \overline{a}_3^2$
a-agent 1 a-agent 2 a-agent 3
$4: \overline{a}_1\ F$ $W\ F$ $\overline{a}_3\ F$
env: $\overline{a}_1^2\ \overline{a}_3^2$
a-agent 1 a-agent 2 a-agent 3
$5: e\ e$ $W\ e$ $e\ e$
env: $\overline{a}_1^3\ \overline{a}_3^3\ F^3$

After generation of 3 copies of each kind of objects that appears in C_0 all a-agents stop working until object a_j appears in the environment. When the copies of F appear in the environment, i-agents consume them, i.e. import them from the environment. From this step on, they rewrite their content $y = 4 + 2k + 2m$ times. After y steps they are prepared to produce the next input.

3. Simulation of reactions

The task of the agents in this phase is to simulate the execution of reactions performed in the same step of the interactive process of the R system. The agents executing this task are called r-agents. Because they need some timing, there is another group of agents called t-agents. In certain steps, these t-agents generate objects that trigger the action of r-agents. The r-agents look for inhibitors, reactants and generate "semi-products" (a'_j) of reactions. After preparation, the search for inhibitors can be done in one step. If there is at least one inhibitor in the environment, then the reaction is not enabled. The r-agents have a program for each inhibitor that allows the agent to consume this inhibitor. If there is no inhibitor in the environment, then the agent has no applicable program. The number of r-agents is the same as the number of t-agents and it equals to $|A|$. Let $r : (R_r, I_r, P_r)$ is a reaction in $\mathcal{A}$. The programs for search for inhibitors are:

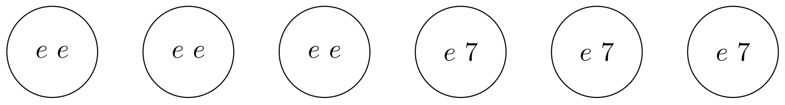
r-agents		t-agents	
a) search for inhibitors		a) activation of r-agents	
$\langle e \to e;\ e \leftrightarrow I \rangle$		$\langle e \to e;\ i \to (i+1) \rangle$	$0 \leq i < 2\lvert A \rvert + 1$
$\langle e \leftrightarrow \overline{a}_j;\ I \to C \rangle$	$a_j \in I_r$	$\langle e \to e;\ (2\lvert A \rvert + 1) \to I \rangle$	
$\langle \overline{a}_j \to e;\ C \to e \rangle$		$\langle e \to T';\ I \leftrightarrow e \rangle$	
$\langle C \to C;\ e \leftrightarrow T \rangle$		$\langle T' \to T;\ e \leftrightarrow e \rangle$	
$\langle C \to e;\ T \to e \rangle$		$\langle T \leftrightarrow e;\ e \to 0' \rangle$	
$\langle e \leftrightarrow T;\ I \to R \rangle$		$\langle e \to e;\ i' \to (i+1)' \rangle$	$0 \leq i' \leq 2k + 2m$
$\langle T \to e;\ R \to R'_0 \rangle$		$\langle e \to e;\ (2k+2m) \to 0 \rangle$	

Let us return to the previous example. $\mathcal{A}$ has three reactions:

$$r_1 = (\{a_2\}, \emptyset, \{a_2\})$$
$$r_2 = (\{a_1, a_3\}, \{a_2\}, \{a_1, a_2\})$$
$$r_3 = (\{a_3\}, \{a_1\}, \{a_1, a_2\})$$

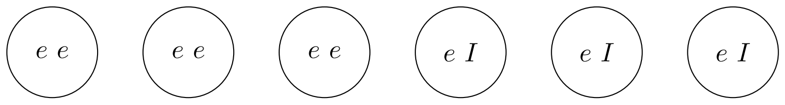
and $C_0 = \{a_1, a_3\}$. Then $k = 2,\ m = 2$ and there are three copies of $\overline{a}_1$ and three copies of $\overline{a}_3$ in the environment of the P colony. The first configuration of this phase is:

r-agent 1 r-agent 2 r-agent 3 t-agent 1 t-agent 2 t-agent 3

0 :


env: $\overline{a}_1^3\ \overline{a}_3^3\ F^3$

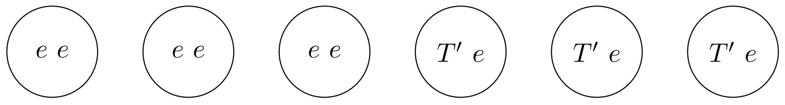
r-agent 1 r-agent 2 r-agent 3 t-agent 1 t-agent 2 t-agent 3

1 :


env: $\overline{a}_1^3\ \overline{a}_3^3$

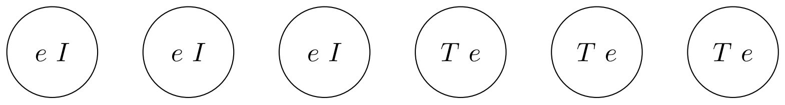
The copies of object F were consumed by i-agents (see the third program of i-agents in the first phase).

r-agent 1 r-agent 2 r-agent 3 t-agent 1 t-agent 2 t-agent 3

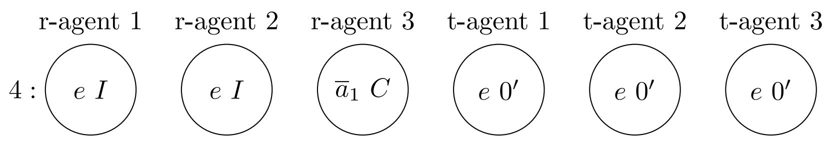
2 :


env: $\overline{a}_1^3\ \overline{a}_3^3\ I^3$

r-agent 1 r-agent 2 r-agent 3 t-agent 1 t-agent 2 t-agent 3

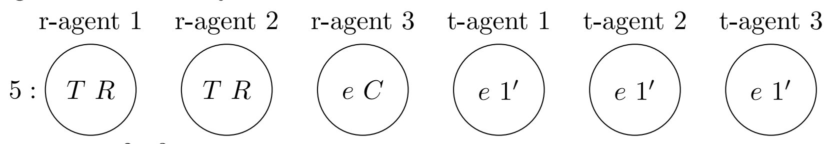
3 :


env: $\overline{a}_1^3\ \overline{a}_3^3$

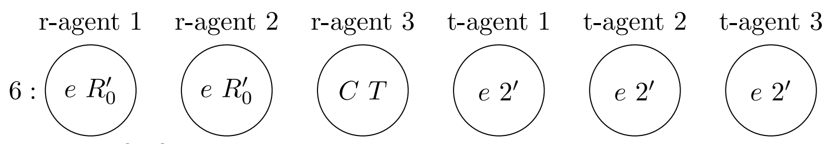


env: $\overline{a}_1^2\ \overline{a}_3^3\ T^3$

Reaction r_1 has empty inhibitor set, so the corresponding r-agent has no program to apply in this configuration. There is one inhibitor, a_2, in inhibitor set of reaction r_2 and because a_2 is not present in the environment, therefore r-agent 2 has no applicable program in current configuration. Due to the presence of a_1 in C_0, the third reaction is not enabled by C_0 and r-agent 3 has one program to execute. The agent consumes object a_1 from the environment.



env: $\overline{a}_1^2\ \overline{a}_3^3\ T$



env: $\overline{a}_1^2\ \overline{a}_3^3$

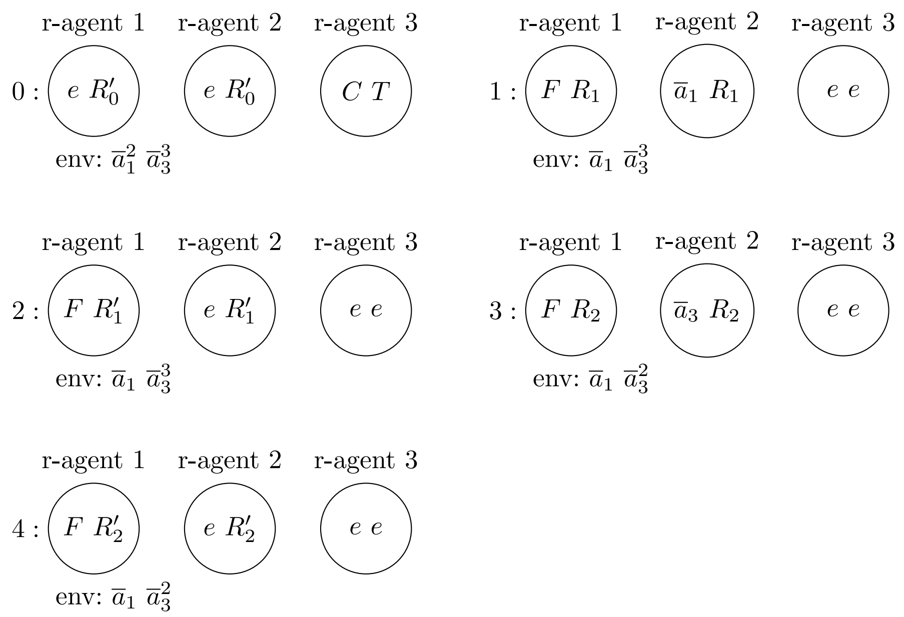
Those agents which have object R inside can continue the simulation of executing reaction by search for reactants. All reactants must be present in the environment to enable reaction. For every reaction we make a sequence of reactants $(a_j)_{j=1}^k$ in random order. Then we can construct programs for search for reactants phase:

r-agents

b) search for reactants

$\langle e \leftrightarrow \overline{a}_j\ /\ e \to F;\ R'_{j-1} \to R_j \rangle$	$a_j \in R_r;\ 1 \le j \le \vert R_r\vert$
$\langle \overline{a}_j \to e;\ R_j \to R'_j \rangle$	$a_j \in R_r;\ 1 \le j \le \vert R_r\vert$
$\langle F \to F;\ R'_{j-1} \to R_j \rangle$	$1 < j \le k$
$\langle F \to F;\ R_j \to R'_j \rangle$	$1 \le j \le k$
$\langle e \to e;\ R'_{j-1} \to R_j \rangle$	$\vert R_r\vert < j \le k$
$\langle e \to e;\ R_j \to R'_j \rangle$	$\vert R_r\vert < j \le k$
$\langle F \to e;\ R'_k \to e \rangle$	

In the example P colony, we develop, the search for reactants is performed as follows:

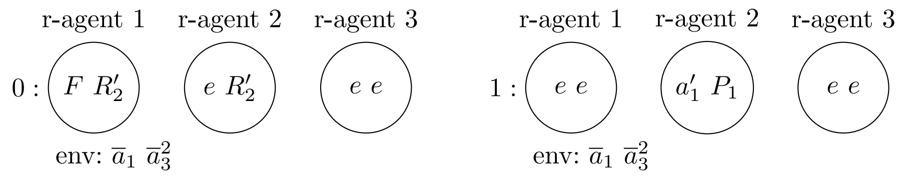

r-agent 1 r-agent 2 r-agent 3
0 : $e\ R'_0$ $e\ R'_0$ $C\ T$
env: $\overline{a}_1^2\ \overline{a}_3^3$
r-agent 1 r-agent 2 r-agent 3
1 : $F\ R_1$ $\overline{a}_1\ R_1$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^3$
r-agent 1 r-agent 2 r-agent 3
2 : $F\ R'_1$ $e\ R'_1$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^3$
r-agent 1 r-agent 2 r-agent 3
3 : $F\ R_2$ $\overline{a}_3\ R_2$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^2$
r-agent 1 r-agent 2 r-agent 3
4 : $F\ R'_2$ $e\ R'_2$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^2$

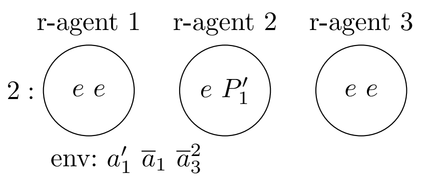
Only r-agents in state (e, R'_k) can continue the simulation. They consumed all the reactants and because they pass the phase a) the corresponding reaction is enabled by W_i. In phase of products generation the r-agents will generate and put into environment "semi-products", i.e. objects corresponding to products. For this purpose, we also need a sequence of products for each reaction.

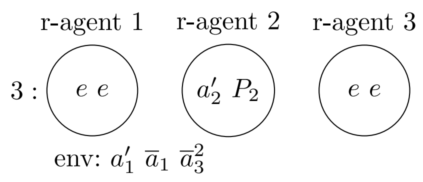
r-agents

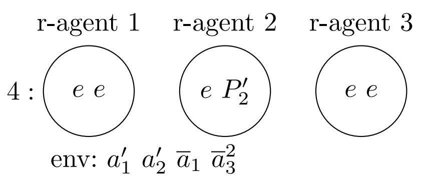
c) generation of products

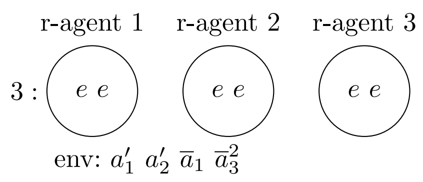
$\langle e \to a'_1;\ R'_k \to P_1 \rangle$	
$\langle e \to a'_j;\ P'_{j-1} \to P_j \rangle$	$a_j \in P_r;\ 1 < j \leq \lvert P_r \rvert$
$\langle e \leftrightarrow a'_j;\ P_j \to P'_j \rangle$	$a_j \in P_r;\ 1 \leq j \leq \lvert P_r \rvert$
$\langle e \to e;\ P'_{j-1} \to P_j \rangle$	$\lvert P_r \rvert < j \leq m$
$\langle e \to e;\ P_j \to P'_j \rangle$	$\lvert P_r \rvert < j \leq m$
$\langle e \to e;\ P'_m \to e \rangle$	


r-agent 1 r-agent 2 r-agent 3
0 : $F\ R'_2$ $e\ R'_2$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^2$
r-agent 1 r-agent 2 r-agent 3
1 : $e\ e$ $a'_1\ P_1$ $e\ e$
env: $\overline{a}_1\ \overline{a}_3^2$





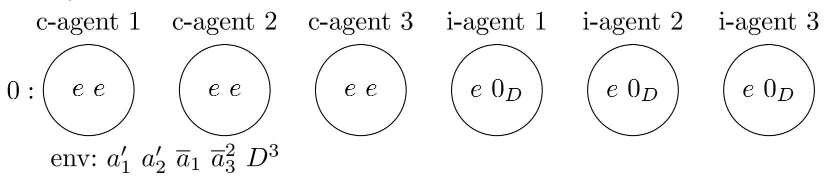


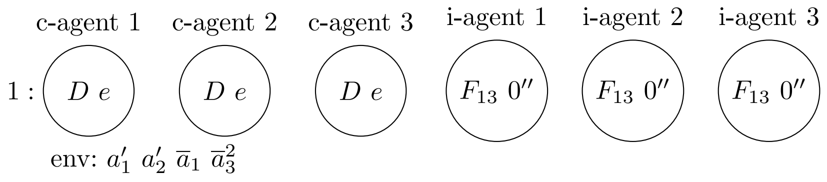


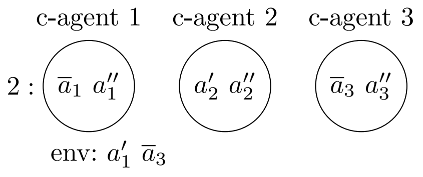
4. Consuming phase

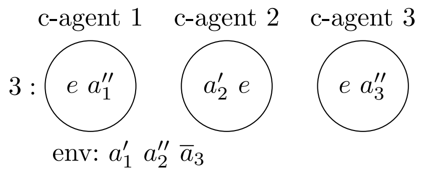
In this phase all unused over-lined objects are consumed by c-agents as well as copies of "semi-products". Only one copy of semi-product stays in the environment. The number of c-agents is $|S|$ and this phase takes at most $4|A|$ steps.

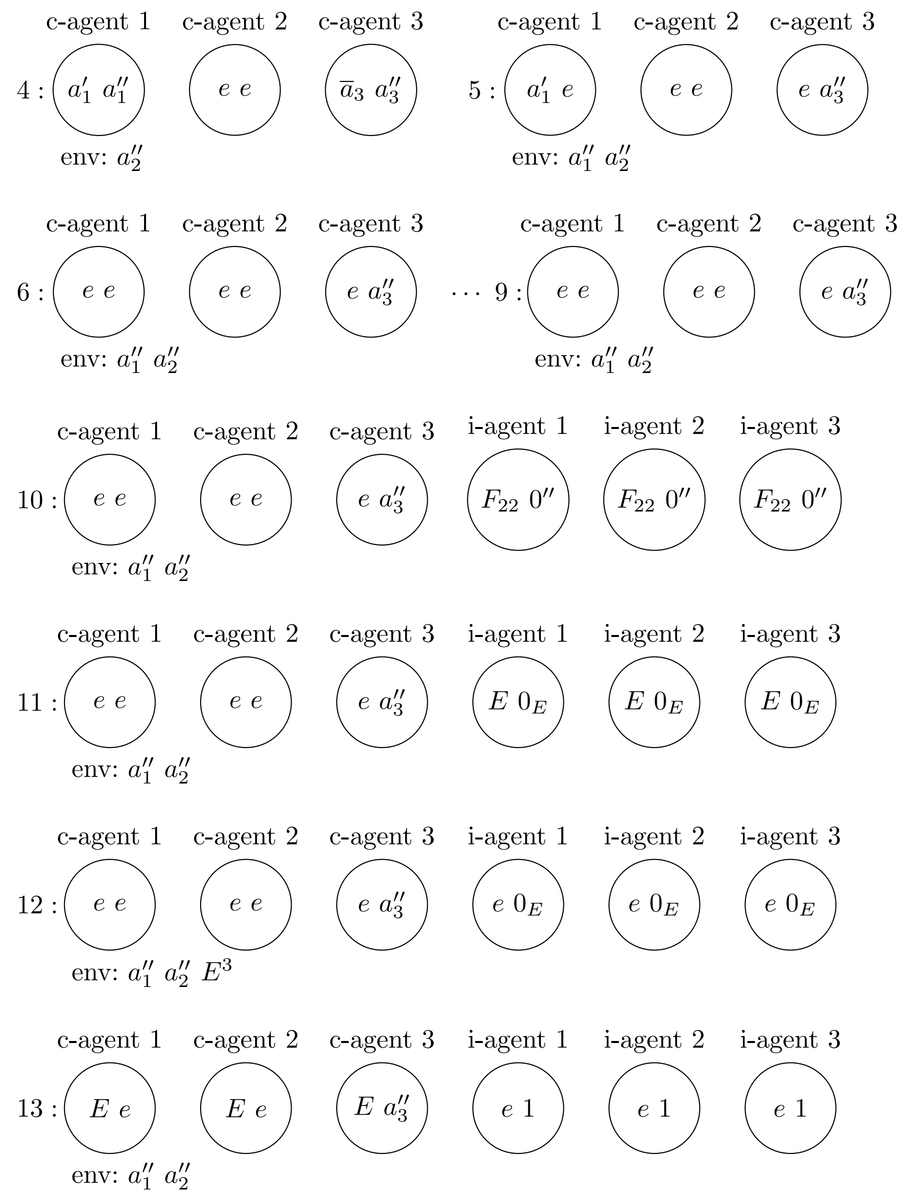
$\langle e \leftrightarrow D;\ e \to e\rangle$ $\quad\left\langle e \leftrightarrow \overline{a}_j\ /\ e \leftrightarrow a_j';\ D \to a_j''\right\rangle$
$\left\langle \overline{a}_j \to e;\ a_j'' \to a_j''\right\rangle$ $\quad\left\langle e \leftrightarrow \overline{a}_j\ /\ e \leftrightarrow a_j';\ a_j'' \to a_j''\right\rangle$
$\left\langle a_j' \to a_j';\ a_j'' \leftrightarrow e\right\rangle$ $\quad\left\langle e \leftrightarrow a_j'\ /\ e \leftrightarrow e;\ a_j' \to e\right\rangle$
$\langle e \leftrightarrow E;\ e \to e\rangle$ $\quad\langle D \to e;\ e \leftrightarrow E\rangle$
$\left\langle a_j'' \to e;\ e \leftrightarrow E\right\rangle$ $\quad\langle E \to e;\ e \to e\rangle$










c-agent 1 c-agent 2 c-agent 3 c-agent 1 c-agent 2 c-agent 3
4 : $a'_1\ a''_1$ $e\ e$ $\overline{a}_3\ a''_3$ 5 : $a'_1\ e$ $e\ e$ $e\ a''_3$
env: a''_2 env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 c-agent 1 c-agent 2 c-agent 3
6 : $e\ e$ $e\ e$ $e\ a''_3$ $\cdots$ 9 : $e\ e$ $e\ e$ $e\ a''_3$
env: $a''_1\ a''_2$ env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
10 : $e\ e$ $e\ e$ $e\ a''_3$ $F_{22}\ 0''$ $F_{22}\ 0''$ $F_{22}\ 0''$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
11 : $e\ e$ $e\ e$ $e\ a''_3$ $E\ 0_E$ $E\ 0_E$ $E\ 0_E$
env: $a''_1\ a''_2$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
12 : $e\ e$ $e\ e$ $e\ a''_3$ $e\ 0_E$ $e\ 0_E$ $e\ 0_E$
env: $a''_1\ a''_2\ E^3$
c-agent 1 c-agent 2 c-agent 3 i-agent 1 i-agent 2 i-agent 3
13 : $E\ e$ $E\ e$ $E\ a''_3$ $e\ 1$ $e\ 1$ $e\ 1$
env: $a''_1\ a''_2$

In this configuration, c-agents rewrite all objects inside them to environmental objects. Simulation can continue with the first phase - generation of input. The i-agents can consume objects of a type a''_j and they put into the environment objects a_j and objects of the next input (if they are not included in products of previous step).

3 Conclusions

In this paper we presented the result obtained by examining P colonies with connection to R systems. In future research we plan further investigation of P colonies that resemble reaction systems in terms of shared environment and computation.

Acknowledgments.

The work of L. Ciencialová and L. Cienciala was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602, by SGS/11/2019.

References

1. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., and Sosík, P.: P colonies. Bulletin of the International Membrane Computing Society 1(2):119–156 (2016).
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, Gy.: Computing with cells in environment: P colonies. Journal of Multiple-Valued Logic and Soft Computing 12(3-4 SPEC. ISS.), pp. 201–215 (2006)
3. Csuhaj-Varjú, E., Kelemen, J., Păun, Gh., Dassow, J.(eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)
4. Ehrenfeucht, A., Rozenberg, A.: Basic notions of reaction systems. In: Calude, C.S., Calude, E., Dinneen, M.J. (Eds.), Developments in Language Theory, 8th International Conference, DLT 2004. In: Lecture Notes in Computer Science, vol.3340, Springer, 27-–29 (2005)
5. Kelemenová, A.: P Colonies. Chapter 23.1, In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 584–593. Oxford University Press (2010)
6. Kelemen, J., Kelemenová, A.: 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. pp. 40–56. Hungary (2005)
7. Kelemen, J., Kelemenová, A., Păun, G.: 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). pp. 82–86. Boston, Mass (2004)
8. Kelemen, J., Kelemenová, A.: A Grammar-Theoretic Treatment of Multiagent Systems. Cybern. Syst. 23(6), 621–633 (1992),
9. Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
10. Rozenberg, G., Salomaa, A.(eds.): Handbook of Formal Languages I-III. Springer Verlag., Berin-Heidelberg-New York (1997)