Towards P Colonies Processing Strings

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

[1] Institute of Computer Science
and
Research Institute of the IT4Innovations Centre of Excellence,
Silesian University in Opava, Czech Republic
{ludek.cienciala, lucie.ciencialova,}@fpf.slu.cz
[2] Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
csuhaj@inf.elte.hu

Summary. In this paper we introduce and study P colonies where the environment is given as a string. These variants of P colonies, called Automaton-like P systems or APCol systems, behave like automata: during functioning, the agents change their own states and process the symbols of the string. After introducing the concept of APCol systems, we examine their computational power. It is shown that the family of languages accepted by jumping finite automata is properly included in the family of languages accepted by APCol systems with one agent, and it is proved that any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents.

1 Introduction

P colonies were introduced in [13] as formal models of a computing device combining properties of membrane systems and distributed systems of formal grammars called colonies. The concept was inspired by the structure and functioning of a community of living organisms in a shared environment (for more information consult [14]).

In the basic model, the cells or agents are represented by a finite collection of objects and rules for processing these objects. The agents are restricted in their capabilities, i.e., only a limited number of objects, say, k objects, are allowed to be inside any cell during the functioning of the system. These objects represent the current state of the agents, in other terms the current contents of the cells. The rules of the cells are either of the form $a \to b$, specifying that an internal object a is transformed into an internal object b, or of the form $c \leftrightarrow d$, specifying that an internal object c is exchanged by an object d in the environment. After applying these rules in parallel, the state of the agent will consist of objects b, d. Each agent is associated with a set of programs composed of such rules.

The agents of a P colony perform a computation by synchronously applying their programs to the objects representing the state of the agents and objects in the environment. At the beginning of the computation, executed by a given P colony of capacity k, i.e., where any agent has at most k symbols inside, the environment contains arbitrarily many copies of a distinguished symbol e, called the environmental symbol (and no more symbols); furthermore, each cell contains k copies of e. When a halting configuration is reached, that is, when no more rules can be applied, the result of the computation is read as the number of certain types of objects in the environment.

P colonies have been extensively examined during the years. For example, it was shown that these systems are computationally complete computing devices even with very restricted size parameters and with other (syntactic or functioning) restrictions [1, 2, 4, 5, 6, 7, 9, 10].

According to the the basic model, the impact of the environment on the behavior of the P colony is indirect. To describe the situation when the behavior of the components of the P colony is influenced by direct impulses coming from the environment step-by-step, the model was augmented with a string put on an input tape to be processed by the P colony [3]. These strings corresponds to the impulse sequence coming from the environment. In addition to their rewriting rules and the rules for communicating with the environment, the agents have so-called tape rules which are used for reading the next symbol on the input tape. This is done by changing one of the objects of the current state of the agents to the object corresponding to the current input symbol on the tape. The symbol is said to be read if at least one agent applied its corresponding tape rule. The model, called a P colony automaton or a PCol automaton, combines properties of standard finite automata and standard P colonies. The P colony automaton starts working with a string on its input tape (the input string) and with initial multisets of objects in its cells. The input string is accepted if it is read by the system and the P colony is in an accepting configuration (in an accepting state). It was shown that P colony automata are able to describe the class of recursively enumerable languages, taking various working mode into account.

In this paper we make one step further in combining properties of P colonies and automata. While in the case of PCol automata the behaviour of the system is influenced both by the string to be processed and the environment consisting of multisets of symbols, in the case of Automaton-like P colonies or APCol systems, for short, introduced in this article, the whole environment is a string. The interaction between the agents in the P colony and the environment is realized by exchanging symbols between the objects of the agents and the environment (communication rules), and the states of the agents may change both via communication and evolution; the latter one is an application of a rewriting rule to an object. The distinguished symbol, e (in the previous models the environmental symbol) have a special role: whenever it is introduced in the string by communication, the corresponding input symbol is erased.

The computation in APCol systems starts with an input string, representing the environment, and with each agents having only symbols e in its state. Every computational step means a maximally parallel action of the active agents: an agent is active if it is able to perform at least one of its programs, and the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents. The computation ends if the input string is reduced to the empty word, there are no more applicable programs in the system, and meantime at least one of the agents is in so-called final state.

After defining the model, we examined the computational power of APCol systems. We proved that the family of languages accepted by jumping finite automata is properly included in the family of languages accepted by APCol systems with one agent and it was shown that any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents. We also provided several examples to demonstrate the behaviour of an APCol system.

2 Definitions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory and membrane computing. For further details we refer to [11] and [17].

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 $M = (V, f)$, 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 with the set of objects V is denoted by V°. The set V' is called the support of M and denoted by $supp(M)$. The cardinality of M, denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V' = \{a_1, \ldots a_n\}$ 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 the same multiset M, and ε represents the empty multiset.

In the following we introduce the concept of an Automaton-like P colony (an APCol system, for short) where the environment of the agents is given in the form of a string.

As in the case of standard P colonies, agents of the APCol systems contain objects, each being an element of a finite alphabet. With every agent, a set of programs is associated. There are two types of rules in the programs. The first one, called an evolution rule, is of the form $a \to b$. It means that object a inside of the agent is rewritten (evolved) to the object b. The second type of rules, called a communication rule, is in the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and a symbol d in the string are exchanged, so, we can say that

the agent rewrites symbol d to symbol c in the input string. If $c = e$, then the agent erases d from the input string and if $d = e$, symbol c is inserted into the string.

An Automaton-like P colony works successfully, if it is able to reduce the given string to ε, i.e., to enter a configuration where at least one agent is in accepting state and the processed string is the empty word.

Definition 1. *An Automaton-like P colony (an APCol system, for short) is a construct*

$$\Pi = (O, e, A_1, \ldots, A_n), \text{ where}$$

- O *is an alphabet; its elements are called the objects,*
- $e \in O$, *called the basic object,*
- $A_i,\ 1 \leq i \leq n$, *are agents. Each agent is a triplet* $A_i = (\omega_i, P_i, F_i)$, *where*
  - ω_i *is a multiset over* O, *describing the initial state (content) of the agent,* $|\omega_i| = 2$,
  - $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ *is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:*
    - $a \to b$, *where* $a, b \in O$, *called an evolution rule,*
    - $c \leftrightarrow d$, *where* $c, d \in O$, *called a communication rule,*
  - $F_i \subseteq O^*$ *is a finite set of final states (contents) of agent* A_i.

In the following we explain the work of an Automaton-like P colony; to help the easier reading we provide only the necessary formal details.

During the work of the APCol system, the agents perform programs. Since both rules in a program can be communication rules, an agent can work with two objects in the string in one step of the computation. In the case of program $\langle a \leftrightarrow b; c \leftrightarrow d\rangle$, a substring bd of the input string is replaced by string ac. If the program is of the form $\langle c \leftrightarrow d; a \leftrightarrow b\rangle$, then a substring db of the input string is replaced by string ca. This means that the agent can act only in one place in the one step of the computation and what happens to the string depends both on the order of the rules in the program and on the interacting objects. In particular, we have the following types of programs with two communication rules:

- $\langle a \leftrightarrow b; c \leftrightarrow e\rangle$ - b in the string is replaced by ac,
- $\langle c \leftrightarrow e; a \leftrightarrow b\rangle$ - b in the string is replaced by ca,
- $\langle a \leftrightarrow e; c \leftrightarrow e\rangle$ - ac is inserted in a non-deterministically chosen place in the string,
- $\langle e \leftrightarrow b; e \leftrightarrow d\rangle$ - bd is erased from the string,
- $\langle e \leftrightarrow d; e \leftrightarrow b\rangle$ - db is erased from the string,
- $\langle e \leftrightarrow e; e \leftrightarrow d\rangle$; $\langle e \leftrightarrow e; c \leftrightarrow d\rangle$, ...- these programs can be replaced by programs of type $\langle e \to e; c \leftrightarrow d\rangle$.

At the beginning of the work of the APCol system (at the beginning of the computation), there is an input string placed in the environment, more precisely, the environment is given by a string ω of objects which are different from e.

This string represents the initial state of the environment. Consequently, an initial configuration of the Automaton-like P colony is an $(n+1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where w is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states the of agents.

A configuration of an Automaton-like P colony Π is given by $(w; w_1, \ldots, w_n)$, where $|w_i| = 2,\ 1 \leq i \leq n$, w_i represents all the objects placed inside the i-th agent and $w \in (O - \{e\})^*$ is the string to be processed.

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 Automaton-like 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 AP-Col system has no applicable program. A computation is called accepting if and only if at least one agent is in final state and the string to be processed is ε. Hence, the string w is accepted by the Automaton-like P colony Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \ldots, \omega_n)$ and the computation ends by halting in the configuration $(\varepsilon; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

3 Computational power of Automaton-like P colonies

The behaviour of Automaton-like P colonies is similar to the functioning of jumping finite automata. The jumping finite automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where the meaning of Q, Σ, δ, q_0 and F is the same as in the case of traditional finite automaton with ε-steps (the set of states, the input alphabet, the transition function, the initial state, and the set of final states). The dissimilarity of the two computing devices is in the way of performing a computational step. The computation starts in a random cell of the input tape. After reading the input symbol in the cell and changing the state of the automaton, the reading head is allowed to jump to some random location on the tape. If the symbol is read, then it is erased from the tape. The notion of a jumping finite automaton was introduced in [15], we refer to this seminal article for the precise details.

Although non-trivial languages can be recognized by jumping finite automata, the following languages cannot be accepted by them: $L_1 = \{ab\},\ L_2 = \{a^n b^n \mid n \geq 0\},\ L_3 = \{a^n b^n c^n \mid n \geq 0\}$. But, jumping finite automata accept $L_4 = \{ab, ba\}, L_5 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\},\ L_6 = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$. It is shown that the family of languages accepted by jumping finite automata is included in the family of context-sensitive languages.

Theorem 1. *For every jumping finite automaton $M = (q, \Sigma, \delta, q_0, F)$ we can construct an Automaton-like P colony $\Pi = (O, e, A)$ such that $L(M) = L(\Pi)$ holds.*

Proof. Let $M = (q, \Sigma, \delta, q_0, F)$ be a jumping finite automaton. We construct an Automaton-like P colony Π with one agent A which simulates every computation of M and only that. The simulation is as follows:

- The current state of M is stored as object inside the agent of Π, i.e., it corresponds to a state of the agent.
- One step of computation of M (except of ε-step) is simulated by two computation steps of Π. In the first step, the agent changes the object corresponding to the state of M and replaces the symbol on the tape by object e - erases the symbol has been read from the input string. In the second step, the agent prepares itself for the simulation of the next step of computation of M - it rewrites the consumed symbol from the tape to e.

Formally, we construct the Automaton-like P colony $\Pi = (A, e, B)$ as follows:

1. $O = Q \cup \Sigma \cup \{e\}$;
2. $A = (q_0 e, P_1, \{q_f e \mid q_f \in F\})$;
3. $P_1 = \{$
   $\langle q \to q', e \leftrightarrow a \rangle$ for every $q, q' \in Q, a \in \Sigma$ such that $q' \in \delta(q, a)$
   $\langle q' \to q', a \to e \rangle$ for every $q' \in Q, a \in \Sigma$
   $\langle q \to q', e \to e \rangle$ for every $q, q' \in Q$ such that $q' \in \delta(q, \varepsilon)$
   $\}$

The Automaton-like P colony starts processing an input string w in initial configuration $(q_0 e; w)$ that corresponds to the initial configuration of jumping finite automaton M (q_0, w).

If M is in the configuration (q, uav), $q \in Q, u, v \in \Sigma^*, a \in \Sigma$ and it performs computational step reaching state q' such that $q' \in \delta(q, a)$, then it enters configuration (q', uv). If the APCol system Π is in the corresponding configuration $(qe; uav)$, then the agent has an applicable program $\langle q \to q', e \leftrightarrow a \rangle$. After executing this program, Π enters configuration $(q'a; uv)$ and in the next step the program $\langle q' \to q', a \to e \rangle$ must be performed. The configuration of Π is $(q'e, uv)$ that corresponds to configuration of M. Then the APCol system Π is prepared to simulate the next step of computation of jumping finite automaton M.

If the automaton M is in the configuration (q, u), $q \in Q, u \in \Sigma^*$ and it performs computational step reaching state q' such that $q' \in \delta(q, \varepsilon)$, then it enters configuration (q', u). If the APCol system Π is in the corresponding configuration $(qe; u)$, then the agent has applicable program $\langle q \to q', e \to e \rangle$. After execution of this program the system is in the configuration $(q'; u)$. This configuration corresponds to configuration of the jumping finite automaton M.

The automaton M accepts input string w iff it passes from the initial configuration (q_0, w) to one of final configuration (q_f, ε). This computation corresponds to computation in Π that starts in the initial configuration $(q_0 e; w)$ and ends in one of final configurations $(q_f e; \varepsilon)$. The computation of Π is halting if and only

if the computation in M is halting, too. Hence the Automaton-like P colony Π accepts string w if only if jumping finite automaton M accepts w. □

Now we show that Automaton-like P colonies are able to accept languages that jumping finite automata cannot.

Example 1. Let $\Pi_1 = (\{a, b, p\}, e, A)$ be an Automaton-like P colony with one agent $A = (ee, P, \{pp\})$. Let the programs of the agent be the following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$
2. $\langle a \rightarrow p; b \rightarrow p \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π_1. There is only one applicable program in this configuration, namely, program 1. This program is applicable only if ab is substring of w. Let $w = u{\cdot}ab{\cdot}v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of Π_1 is (ab, uv). In this configuration the second program $\langle a \rightarrow p; b \rightarrow p \rangle$ is applicable and the APCol system enters configuration (pp, uv). Because of presence of two copies of object p in the state of the agent, there is no applicable program in the APCol system and the computation halts.

The language accepted by Π_1 is $L(\Pi_1) = \{ab\}$.

Example 2. Let $\Pi_2 = (\{a, b, q\}, e, B)$ be an Automaton-like P colony with one agent $B = (ee, P, \{qe\})$. The programs of the agent are following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$
2. $\langle a \rightarrow q; b \rightarrow e \rangle$
3. $\langle q \rightarrow q; e \leftrightarrow a \rangle$
4. $\langle q \rightarrow q; e \leftrightarrow b \rangle$
5. $\langle q \rightarrow q; a \rightarrow e \rangle$
6. $\langle q \rightarrow q; b \rightarrow e \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π_2. There is only one applicable program in this configuration, program 1. This program is applicable only if ab is substring of w. Let $w = u{\cdot}ab{\cdot}v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of Π_2 is (ab, uv). In this configuration the second program $\langle a \rightarrow q; b \rightarrow e \rangle$ is applicable and the APCol system enters configuration (qe, uv). Because of presence of one copy of object q inside the agent, it uses program 3 (or 4) and, consequently, program 5 (or 6). By use of these programs, the agent erases every object a and b from the string.

The language accepted by Π_2 is $L(\Pi_2) = \{u{\cdot}ab{\cdot}v \mid u, v \in \{a, b\}^*\}$.

By the following example, we present a very simple Automaton-like P colony with only one agent with two programs that accepts the context-free language $L = \{a^n b^n \mid n \geq 0\}$.

Example 3. Let $\Pi_3 = (\{a, b\}, e, B)$ be an Automaton-like P colony with one agent $A = (ee, P, \{ee\})$. The programs of the agent are the following:

1. $\langle e \leftrightarrow a; e \leftrightarrow b \rangle$

2. $\langle a \to e; b \to e \rangle$

Let (ee, w), $w \in \{a, b\}^*$ be the initial configuration of Π. There is only one applicable program in this configuration, program 1. This program is applicable only if ab is substring of w. Let $w = u \cdot ab \cdot v$, $u, v \in \{a, b\}^*$. After the first step of the computation the configuration of the APCol system is (ab, uv). In this configuration the second program $\langle a \to e; b \to e \rangle$ is applicable and Π_3 enters configuration (ee, uv). This means that in two step of computation agent erases substring ab from a string. The computation ends when no substring ab occurs in the input string, thus any accepted word must be of the form $a^n b^n, n \geq 0$.

An interesting question is if we can construct an Automaton-like P colony which accepts the language $L = \{a^n b^n c^n \mid n \geq 0\}$.

Example 4. Let $\Pi_4 = (\{a, b, B\}, e, A_1, A_2, \{((ee), (bB))\})$ be an Automaton-like P colony with two agents $A_1 = (eB, P_1, \{ee\})$ and $A_2 = (bB, P_2, \{bB\})$. The programs of the agents are the following:

Set of programs P_1:
1. $\langle e \leftrightarrow a; B \leftrightarrow b \rangle$
2. $\langle a \to e; b \to B \rangle$
3. $\langle a \to e; b \to e \rangle$
4. $\langle e \leftrightarrow B; e \leftrightarrow c \rangle$
5. $\langle B \to e; c \to e \rangle$
6. $\langle e \leftrightarrow a; e \to F \rangle$
7. $\langle e \leftrightarrow b; e \to F \rangle$

Set of programs P_2:
A. $\langle b \leftrightarrow B; B \leftrightarrow b \rangle$

It can be shown that $L(\Pi_4) = \{a^n b^n c^n \mid n \geq 0\}$. Instead of the formal proof, we provide the sketch of the main idea. Π_4 functions as follows:

1. the first agent is an "eraser" - it erases substrings from the input string - it replaces string ab by B and erases substring Bc.
2. The second agent moves object B over that part of the input string which contains objects b.
3. The computation is accepting if every object from the input string is erased and the state of the first agent consists of only objects e.

We show an example of an accepting computation of Π_4 over the string $w = aaabbbccc$.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aaabbbccc$	eB	1.	Bb	
1.	$aaBbbccc$	ab	**2.**, 3.	Bb	$A.$
2.	$aabBbccc$	eB	1.	Bb	$A.$
3.	$aBbBccc$	ab	**2.**, 3.	Bb	$A.$
4.	$abBBccc$	eB	1.	Bb	
5.	$BBBccc$	ab	2., **3.**	Bb	
6.	$BBBccc$	ee	4.	Bb	
7.	$BBcc$	Bc	5.	Bb	
8.	$BBcc$	ee	4.	Bb	
9	Bc	Bc	5.	Bb	
10.	Bc	ee	4.	Bb	
11.	ε	Bc	5.	Bb	
12.	ε	ee		Bb	

The next example is a non-accepting computation over the string $w = aaabbbccc$ on the next table.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aaabbbccc$	eB	1.	Bb	
1.	$aaBbbccc$	ab	**2.**, 3.	Bb	$A.$
2.	$aabBbccc$	eB	1.	Bb	$A.$
3.	$aBbBccc$	ab	2., **3.**	Bb	$A.$
4.	$abBBccc$	ee	**6.**, 7.	Bb	
5.	$bBBccc$	Fe		Bb	

The Automaton-like P colony processes the string $u = aabcbc$ in the following computations.

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aabcbc$	eB	1.	Bb	
1.	$aBcbc$	ab	**2.**, 3.	Bb	$A.$
2.	$aBcbc$	eB		Bb	

step of computation	string	agent A_1		agent A_1	
		content	applicable programs	content	applicable programs
0.	$aabcbc$	eB	1.	Bb	
1.	$aBcbc$	ab	2., **3.**	Bb	
2.	$aBcbc$	ee	4.	Bb	
3.	abc	Bc	5.	Bb	
4.	abc	ee	**6.**, 7.	Bb	
5.	bc	Fa		Bb	

By the previous examples we obtain the following corollary.

Corollary 1. *The family of languages accepted by Automaton-like P colonies with one agent properly includes the family of languages accepted by jumping finite automata.*

Automaton-like P colonies with one agent are able to simulate jumping finite automata. If we add one more agent, we can construct Automaton-like P colonies simulating the work of two-counter machines. For details on two-counter machines consult [11, 16].

A two-counter machine is a 3-tape Turing machine $M = (\Sigma \cup \{Z, B\}, Q, R)$ where:

- Σ is an finite set of symbols - alphabet of input symbols,
- Z is the symbol marking the beginning of second and third tape (first and second counter),
- B is the blank symbol,
- Q is a finite set of states with two distinguished elements $q_0, q_f \in Q$, q_0 is the initial state and q_f is the final state of M,
- R is a set of transition rules.

The first tape of the machine is the input tape and second and third tapes are called storage tapes. All tapes are read-only and the storage tapes are semi-infinite. At the beginning of the storage tape symbol Z is found and the rest of the tape is empty (it contains only symbols B). The input tape contains the input word and occurrences of B.

The transition rules have the form $x = \langle b, q, c_1, c_2, q', e_1, e_2, g \rangle$ where:

- $b \in \Sigma \cup \{B\}$ is the symbol scanned by the input head on the input tape,
- $q, q' \in Q$ is the state of the two-counter machine,
- $c_1, c_2 \in \{Z, B\}$ are the symbols scanned on the storage tapes,
- $e_1, e_2 \in \{-1, 0, +1\}$ describe the move of the heads on the storage tapes,
- $g \in \{0, +1\}$ describe the move of the input head.

This rule can be performed by M if symbol scanned on input tape is b, M is in the state q and symbols read from storage tapes are c_1, c_2. By applying this rule M enters state q' and the heads move in according to g, e_1, e_2. If the value is -1, then the head moves to the left, if the value is $+1$, then the head moves to the right, and, if its value is 0, then the head stays at the same position. The head on

input tape can never move to the left and if symbol on read from storage tape is Z, then the head must not to move to the left, too.

The integer stored in a counter is number of all blank symbols laying between the storage head (and under it) and symbol Z on the storage tape. If the head scans symbol Z, then the counter has value 0.

The state of M, the contents of the tapes, and the position of heads on the tapes together define configuration of M. The two-counter machine is in an initial configuration if it is in initial state, on the first tape it has the input word, on the second and third tape it has a word from ZB^*, and the heads are scanning the first symbol of the tapes. The two-counter machine M is in the accepting configuration if the input head reads the last non-blank symbol on the input tape and the machine is in the final state; in this case the input word is accepted.

Two-counter machine are computationally complete computing devices [11, 8].

Theorem 2. *Let Σ be an alphabet, $L \subseteq \Sigma^*$ be a recursively enumerable language. Let $L' = S \cdot L \cdot E$, where $S, E \notin \Sigma$. Then there exists an Automaton-like P colony Π with two agents such that $L' = L(\Pi)$ holds.*

Proof. Let $M = (\Sigma \cup \{Z, B\}, Q, R)$ be a two-counter machine accepting language L. We construct an Automaton-like P colony $\Pi = (O, e, A_1, A_2)$ which accepts $S \cdot L \cdot E$, where:

- $O = \Sigma \cup \{e, Z_1, Z_2, B_1, B_2, D, G, E, S, S', \overline{S}, \overline{\overline{S}}, Q, \overline{Q}, \overline{\overline{Q}}, M, \overline{M}, \overline{\overline{M}}, \} \cup$
  $\cup \{ {}^{A}_{B} \mid A \in \Sigma \cup \{Z_1, Z_2, B_1, B_2\},$
  $B \in \{T, T_1, T_2, H_{+1}, H_0, I_{+1}, I_{-1}, I_0, J_{+1}, J_{-1}, J_0\} \cup$
  $\cup \{q, \overline{q}, \overline{\overline{q}}, \overline{\overline{\overline{q}}} \mid q \in Q\}\}$
- $A_1 = (S'G, P_1, \{\overline{\overline{M}}\, {}^{A}_{H},$ where $A \in \Sigma \cup \{e\}, H \in \{H_{+1}, H_0\}\})$
- $A_2 = ({}^{Z_1\ Z_2}_{T_1\ T_2}, P_2, \{\overline{\overline{Q}}e\})$

The sets of programs are the following:

For the first part of computation, programs are needed to initialize the simulation. They generate the initial contents of the counters and mark the first symbols on each tape, i.e., the symbols under the reading heads (for example, replace a by ${}^{a}_{T}$).

The programs for initialization of the simulation:

A_1 :	A_2 :
1. $\langle S' \leftrightarrow S; G \leftrightarrow a \rangle$	1. $\left\langle {}^{Z_1}_{T_1} \leftrightarrow e; {}^{Z_2}_{T_2} \leftrightarrow S' \right\rangle$
2. $\left\langle S \rightarrow \overline{S}; a \rightarrow {}^{a}_{T} \right\rangle$	2. $\langle S' \rightarrow D; e \rightarrow e \rangle$
3. $\left\langle \overline{S} \rightarrow \overline{\overline{S}}; {}^{a}_{T} \leftrightarrow G \right\rangle$	
4. $\left\langle \overline{\overline{S}} \rightarrow q_0; G \rightarrow {}^{a}_{H} \right\rangle$	
$\forall a \in \Sigma,\ H \in \{H_{+1}, H_0\}$	

Let the input word of the two-counter machine be $w = av,\ a \in \Sigma,\ w, v \in \Sigma^*$. At the beginning of the computation the input tape contains the word $SavE$.

Agent A_1 uses the program 1. and replaces first two symbols Sa by string $S'G$. In the second step, both agents work. Agent A_1 rewrites objects Sa to objects $\overline{S}\,{}^{a}_{T}$ (inside it), and the second agent A_2 replaces symbol S' by string ${}^{Z_1\ Z_2}_{T_1\ T_2}$. In the third step agent A_1 replaces symbol G by symbol ${}^{a}_{T}$ and rewrites object $\overline{S}$ by object $\overline{\overline{S}}$. Agent A_2 changes its state from $(S'e)$ to De. Agent A_2 is prepared to continue the simulation, otherwise agent A_1 has to do one more step. At the last step of the initialization, agent A_1 uses program 4. and it rewrites $\overline{\overline{S}}G$ to $q_0\,{}^{a}_{H}$.

$$(SavE; S'G, {}^{Z_1\ Z_2}_{T_1\ T_2}) \Rightarrow (S'GvE; Sa, {}^{Z_1\ Z_2}_{T_1\ T_2}) \Rightarrow ({}^{Z_1\ Z_2}_{T_1\ T_2}GvE; \overline{S}\,{}^{a}_{T}, S'e) \Rightarrow$$
$$\Rightarrow ({}^{Z_1\ Z_2\ a}_{T_1\ T_2\ T}vE; \overline{\overline{S}}G, De) \Rightarrow ({}^{Z_1\ Z_2\ a}_{T_1\ T_2\ T}vE; q_0\,{}^{a}_{H}, De)$$

The second group of programs is to simulate the execution of the transition rules of the two-counter machine. Let $\langle b, q, c_1, c_2, q', e_1, e_2, g\rangle$ be a transition rule, where $b \in \Sigma \cup \{B\}$, $q, q' \in Q$, $c_1, c_2 \in \{Z, B\}$, $e_1, e_2 \in \{-1, 0, +1\}$, $g \in \{0, +1\}$. Agent A_1 has programs to check whether the reading heads are over the corresponding symbols, i.e., symbols b, c_1 and c_2, and to note the string moves of the reading heads by replacing ${}^{b}_{T}$, ${}^{c_1}_{T_1}$ and ${}^{c_2}_{T_2}$ by ${}^{b}_{g}$, ${}^{c_1}_{e_1}$ and ${}^{c_2}_{e_2}$.

A_1 :

5. $\left\langle q \to \overline{q}; {}^{b}_{g} \leftrightarrow {}^{b}_{T} \right\rangle$
6. $\left\langle \overline{q} \to \overline{q_1}; {}^{b}_{T} \to {}^{c_{1\,1}}_{I_{e_1}} \right\rangle$
7. $\left\langle \overline{q_i} \to \overline{q_{i+1}}; {}^{c_{1\,1}}_{I_{e_1}} \to {}^{c_{1\,1}}_{I_{e_1}} \right\rangle; 1 \geq i \geq 5$
8. $\left\langle \overline{q_6} \to \overline{\overline{q}}; {}^{c_{1\,1}}_{I_{e_1}} \leftrightarrow {}^{c_{1\,1}}_{T_1} \right\rangle$
9. $\left\langle \overline{\overline{q}} \to \overline{\overline{q_1}}; {}^{c_{1\,1}}_{T_1} \to {}^{c_{2\,2}}_{J_{e_2}} \right\rangle$
10. $\left\langle \overline{\overline{q_j}} \to \overline{\overline{q_{j+1}}}; {}^{c_{2\,2}}_{J_{e_2}} \to {}^{c_{2\,2}}_{J_{e_2}} \right\rangle; j \in \{1, 2\}$
11. $\left\langle \overline{\overline{q_3}} \to \overline{\overline{q}}; {}^{c_{2\,2}}_{J_{e_2}} \leftrightarrow {}^{c_{2\,2}}_{T_2} \right\rangle$
12. $\left\langle \overline{\overline{q}} \to \overline{\overline{q_1}}; {}^{c_2}_{T_2} \to {}^{a}_{H} \right\rangle$
13. $\left\langle \overline{\overline{q_k}} \to \overline{\overline{q_{k+1}}}; {}^{a}_{H} \to {}^{a}_{H} \right\rangle; k \in \{1, 2\}$
14. $\left\langle \overline{\overline{q_3}} \to q'; {}^{a}_{H} \to {}^{a}_{H} \right\rangle$

$\forall a \in \Sigma \cup \{B\},\ H \in \{H_{+1}, H_0\}$

Agent A_1 performs some "waiting" steps to let the second agent execute the movement of reading heads. At the last steps of this part of the computation agent A_1 in some way "precomputes" the symbol to be read in the next step. This symbol is non-deterministically chosen from the set $\Sigma \cup \{B\}$. If the precomputed symbol does not match the symbol on the tape in the next step, the computation halts in a non-accepting configuration. The second agent has programs to perform the movement of the reading heads.

A_2 :

the movement of reading head on the input tape

15. $\left\langle D \leftrightarrow {}^{b}_{H_0}; e \rightarrow {}^{b}_{T} \right\rangle$	17. $\left\langle D \leftrightarrow {}^{b}_{H_{+1}}; e \leftrightarrow a \right\rangle$	19. $\left\langle D \leftrightarrow {}^{b}_{H_{+1}}; e \leftrightarrow E \right\rangle$
16. $\left\langle {}^{b}_{H_0} \rightarrow e; {}^{b}_{T} \leftrightarrow D \right\rangle$	18. $\left\langle {}^{b}_{H_{+1}} \rightarrow b; a \rightarrow {}^{a}_{T} \right\rangle$	20. $\left\langle {}^{b}_{H_{+1}} \rightarrow b; E \rightarrow B' \right\rangle$
		21. $\langle b \leftrightarrow D; B' \leftrightarrow e \rangle$
		22. $\left\langle D \leftrightarrow B'; e \rightarrow {}^{B}_{T} \right\rangle$
		23. $\left\langle B' \rightarrow E; {}^{B}_{T} \rightarrow {}^{B}_{T} \right\rangle$
		24. $\left\langle {}^{B}_{T} \leftrightarrow D; E \leftrightarrow e \right\rangle$

$\forall a \in \Sigma,\ H \in \{H_{+1}, H_0\}$

In the first column there are programs for the case when the reading head does not move - $g = 0$. In the second and third column there are programs for executing the move of the head to the right - $g = +1$. In the third column there are programs to do movement to the right when the reading head is on the last symbol of the string, so the head have to move to the blank symbol after the string.

A_2 :

the movement of reading head on the first counter, $X \in \{Z_1, B_1\}$

25. $\left\langle D \leftrightarrow {}^{X}_{I_0}; e \rightarrow {}^{X}_{T_1} \right\rangle$	27. $\left\langle D \leftrightarrow {}^{X}_{I_{+1}}; e \rightarrow {}^{B_1}_{T_1} \right\rangle$	30. $\left\langle D \leftrightarrow Y; e \leftrightarrow {}^{X}_{I_{-1}} \right\rangle$
26. $\left\langle {}^{X}_{I_0} \rightarrow e; {}^{X}_{T_1} \leftrightarrow D \right\rangle$	28. $\left\langle {}^{X}_{I_{+1}} \rightarrow X; {}^{B_1}_{T_1} \rightarrow {}^{B_1}_{T_1} \right\rangle$	31. $\left\langle Y \rightarrow {}^{Y}_{T_1}; {}^{X}_{I_{-1}} \rightarrow e \right\rangle$
	29. $\left\langle X \leftrightarrow D; {}^{B_1}_{T_1} \leftrightarrow e \right\rangle$	28. $\left\langle {}^{Y}_{T_1} \leftrightarrow D; e \leftrightarrow e \right\rangle$

A_2 :

the movement of reading head on the second counter, $X \in \{Z_2, B_2\}$

32. $\left\langle D \leftrightarrow {}^{X}_{J_0}; e \rightarrow {}^{X}_{T_2} \right\rangle$	34. $\left\langle D \leftrightarrow {}^{X}_{J_{+1}}; e \rightarrow {}^{B_2}_{T_2} \right\rangle$	37. $\left\langle D \leftrightarrow Y; e \leftrightarrow {}^{X}_{J_{-1}} \right\rangle$
33. $\left\langle {}^{X}_{J_0} \rightarrow e; {}^{X}_{T_2} \leftrightarrow D \right\rangle$	35. $\left\langle {}^{X}_{J_{+1}} \rightarrow X; {}^{B_2}_{T_2} \rightarrow {}^{B_2}_{T_2} \right\rangle$	38. $\left\langle Y \rightarrow {}^{Y}_{T_2}; {}^{X}_{J_{-1}} \rightarrow e \right\rangle$
	36. $\left\langle X \leftrightarrow D; {}^{B_2}_{T_2} \leftrightarrow e \right\rangle$	39. $\left\langle {}^{Y}_{T_2} \leftrightarrow D; e \leftrightarrow e \right\rangle$

The programs devoted to reading heads on the counters are similar to the ones for the reading head on the input tape. The programs in the first columns are for staying at the same place, the second columns for movement to the right, and the last columns for the movement of the reading head to the left.

Now we finish the simulation of execution of the transition rules. The last group of programs is to finish the computation after the two-counter machine M comes to final state. M accepts the string on the input tape only if it is in the final state and the reading head on the input tape is in the position on the last non-blank symbol. The Automaton-like P colony Π ends computation by halting. The string is accepted by Π only if the input tape is empty after halting in the final state of at least one agent. So, after reaching the final state of M the Automaton-like P colony Π has to erase the symbols on and before the reading head (symbol of

the type $\begin{smallmatrix} a \\ T \end{smallmatrix}$ and all symbols placed on the left from this symbol - the contents of both counters) on the input tape, symbols B and finally the symbol E, which determines the end of the string w. If there is some non-erased symbol left on the input tape, then the two-counter machine does not accept the word w, too, because the reading head in not on the last non-blank symbol of the input tape.

A_1 :

erasing the last symbol over the reading head on the input tape
$a \in \Sigma, H \in H_0, H_{+1}$

32. $\left\langle q_f \leftrightarrow M; \begin{smallmatrix} a \\ H \end{smallmatrix} \leftrightarrow \begin{smallmatrix} a \\ T \end{smallmatrix} \right\rangle$ 34. $\left\langle \overline{M} \rightarrow \overline{\overline{M}}; Q \leftrightarrow \begin{smallmatrix} a \\ H \end{smallmatrix} \right\rangle$
33. $\left\langle M \rightarrow \overline{M}; \begin{smallmatrix} a \\ T \end{smallmatrix} \rightarrow Q \right\rangle$ 35. $\left\langle \overline{\overline{M}} \rightarrow \overline{\overline{M}}; \begin{smallmatrix} a \\ H \end{smallmatrix} \rightarrow e \right\rangle$

A_2 :

erasing symbols on the counters and at the end of the tape
$Z \in \{Z_1, Z_2, B_1, B_2, \begin{smallmatrix} Z_1 \\ T_1 \end{smallmatrix}, \begin{smallmatrix} Z_2 \\ T_2 \end{smallmatrix}, \begin{smallmatrix} B_1 \\ T_1 \end{smallmatrix}, \begin{smallmatrix} B_2 \\ T_2 \end{smallmatrix}, B, E\}$

36. $\left\langle D \rightarrow \overline{Q}; e \rightarrow Q \right\rangle$ 38. $\left\langle \overline{\overline{Q}} \leftrightarrow \overline{\overline{q}}; e \leftrightarrow Z \right\rangle$
37. $\left\langle \overline{Q} \rightarrow \overline{\overline{Q}}; Q \leftrightarrow e \right\rangle$ 39. $\left\langle \overline{\overline{Q}} \rightarrow \overline{\overline{Q}}; Z \rightarrow e \right\rangle$

The computation of Π starts in the initial configuration which corresponds to the initial configuration of two-counter machine M. After the initialization, the simulation of execution of particular transition rules runs in the same way as they are applied by M. The computation of Π halts in accepting configuration only if M processes the whole input string and ends computation in the one of final states. □

By the previous theorem we obtain the following corollary:

Corollary 2. *Any recursively enumerable language can be obtained as a projection of a language accepted by an Automaton-like P colony with two agents.*

4 Conclusions

We introduced the concept on an Automata-like P colony (an APCol system) - a variant of P colonies that works on a string. The agents communicate with the environment alike standard P colonies: they process symbols in the string. As P colony automata, the concept is a notion combining properties of P colonies and classical automata. The main difference in the two notions is in the way of interaction between the agents (the cells) and the environment. We compared the computational power of Automata-like P colonies and that of jumping finite automata, and proved that APCol systems are strictly powerful than jumping finite automata. We also provided a representation of the recursively enumerable language class in terms of APCol systems. The question of exact description of the computational power of Automata-like P colonies is still open.

Remark 1. This work was partially supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/-02.0070), by SGS/24/2011 and by project OPVK no. CZ.1.07/2.2.00/28.0014. and in part by the Hungarian Scientific Research Fund, "OTKA", project K75952.

References

1. L. Ciencialová, L. Cienciala, Variation on the theme: P colonies. In: Proc. 1st Intern. Workshop on Formal Models. (D. Kolář, A. Meduna, eds.), Ostrava, 2006, 27–34.
2. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, Gy. Vaszil, Variants of P colonies with very simple cell structure. International Journal of Computers, Communication and Control 4(3) (2009), 224–233.
3. L. Ciencala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil, PCol Automata: Recognizing strings with P colonies. In: Proc. BWMC 2010, Sevilla, 2010, Ed. by M. A. Martnez-del-Amor et al. Fnix Editora, Sevilla, 2010, 65-76.
4. L. Cienciala, L. Ciencialová, A. Kelemenová, Homogeneous P colonies. Computing and Informatics 27 (2008), 481–496.
5. L. Cienciala, L. Ciencialová, A. Kelemenová, On the number of agents in P colonies. In: Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers. (G. Eleftherakis et. al, eds.), LNCS 4860, Springer-Verlag, Berlin-Heidelberg, 2007, 193–208.
6. E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil, Computing with cells in environment: P colonies. Journal of Multi-Valued Logic and Soft Computing 12 (2006), 201–215.
7. E. Csuhaj-Varjú, M. Margenstern, Gy. Vaszil, P colonies with a bounded number of cells and programs. In: Membrane Computing. 7th International Worskhop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006. Revised, Selected and Invited Papers. (H-J. Hoogeboom et. al, eds), LNCS 4361, Springer-Verlag, Berlin-Heidelberg, (2007), 352–366.
8. P. C. Fischer, Turing machines with restricted memory access. Information and Control, 9, 364–379, 1966.
9. R. Freund, M. Oswald, P colonies working in the maximally parallel and in the sequential mode. Pre-Proc. In: 1st Intern. Workshop on Theory and Application of P Systems. (G. Ciobanu, Gh. Păun, eds.), Timisoara, Romania, 2005, 49–56.
10. R. Freund, M. Oswald, P colonies and prescribed teams. International Journal of Computer Mathematics 83 (2006), 569–592.
11. Hopcroft, J.E., Ullman, J.D., Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, Mass., 1979.
12. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems 23 (1992), 621–633.
13. 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). (M. Bedau et al., eds.), Boston Mass., 2004, 82–86.
14. A. Kelemenová, P Colonies. Chapter 23.1, In: The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010, 584–593.

15. A. Meduna, P. Zemek, Jumping Finite Automata. Int. J. Found. Comput. Sci. 23, 2012, pp. 1555–1578.
16. M. Minsky, Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
17. Gh. Păun, G. Rozenberg, A. Salomaa, eds.,The Oxford Handbook of Membrane Computing. Oxford University Press, 2010.