PCol Automata: Recognizing Strings with P Colonies*

Luděk Cienciala[1], Lucie Ciencialová[1], Erzsébet Csuhaj-Varjú[2,3], György Vaszil[2]

[1] Institute of Computer Science, Silesian University in Opava
Bezručovo nám. 13, 74601 Opava, Czech Republic
{ludek.cienciala,lucie.ciencialova}@fpf.slu.cz

[2] Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu

[3] Department of Algorithms and Their Applications
Faculty of Informatics, Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary

Summary. We introduce the concept of a P colony automaton, an automata-like construct combining properties of finite automata and P colonies. We present some preliminary results on the accepting power of several variants of these extremely simple language recognizing devices, and propose problems for future research.

1 Introduction

P colonies are particular variants of very simple tissue-like membrane systems, modeling a community of very simple cells living together in a shared environment (for P colonies, see [12, 13], for membrane computing we refer to [15, 16]. In the basic model, the cells, the basic computing agents, are represented by a 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 function of the system. Number k is said to be the capacity of the P colony. 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 the fact that an internal object c is sent out of the cell, to the environment, in exchange of the object d, which was present in the environment. Thus, after applying these rules in parallel, a cell containing the objects a, c will contain the objects b, d. With each cell, a set of programs composed

* Research supported in part by the Czech Science Foundation, "GAČR", project 201/09/P075 and by the Hungarian Scientific Research Fund, "OTKA", project K75952.

of rules is associated. In the case of P colonies of capacity k, each program has k rules; the rules of the program must be applied in parallel to the objects in the cell.

The cells of a P colony execute a computation by synchronously applying their programs to objects inside the cells and outside in the environment. 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 present in the environment.

P colonies have been extensively examined during the years: among other things, it has been shown that these extremely simple constructs are computationally complete computing devices even with very restricted size parameters and with other syntactic or functioning restrictions [1, 2, 3, 4, 5, 6, 9, 10].

In the generic model, the environment is a multiset of objects, and thus its impact 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 is augmented with a string put on an input tape to be processed by the P colony. These string corresponds to the impulse sequence coming from the environment. In addition to their rewriting rules and the rules for communicating with the environment, the cells have so-called tape rules which are used for reading the next symbol on the input tape. This is done by changing one of objects inside the cell 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. It is easy to observe that the model, called a P colony automaton or a PCol automaton, resembles to standard finite automata and P automata [7], furthermore, to colonies of formal grammars [11].

PCol automata may work in several computation modes: for example, at any step of the computation a maximal set of components may be active and each component (at least one component, or a maximal number of components) should perform a tape rule. These computation modes are the so-called t, $tmin$, and $tmax$ modes. In some other cases, transitions, i.e., simultaneous applications of non-tape rules are also allowed. These cases are the so-called nt, $ntmin$, $ntmax$, and *initial* computation modes. 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).

Due to their extreme simplicity, it is a challenging question how much accepting power can be obtained by the different variants of PCol automata. In this paper we present some preliminary results. Among other things, we show that PCol automata working in any of the nt, $ntmin$, or $ntmax$ computational modes are able to recognize every recursively enumerable language over any alphabet (thus over any unary alphabet as well). Notice that P colonies are able to generate/accept any recursively enumerable set of numbers, which set can be represented as the length set of words of a recursively enumerable language over a unary alphabet. The large recognizing power of PCol automata working in these modes is due to the unbounded "workspace" provided by the symbols sent to the environment by the components while performing n, nt, $ntmin$, or $ntmax$-transitions, respectively. In the case of t-mode, we have some preliminary results. It is shown that PCol automata are able to accept any regular language (over any alphabet). Furthermore, there is a PCol automaton which recognizes the non-context-free context-sensitive language $\{a^n b^n c^n \mid n \geq 1\}$. In the case of initial mode, we provide a PCol automaton which accepts the language $L = \{a^{2^n}\}$. Finally, we propose some research areas for future study.

2 Preliminaries and definitions

Let V be a finite alphabet, let the set of all words over V be denoted by V^*, and let ε be the empty word. We denote the number of occurrences of a symbol $a \in V$ in w by $|w|_a$.

If the set of non-negative integers is denoted by $\mathbb{N}$, then a multiset over a set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $\mathit{supp}(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $\mathit{supp}(M) = \emptyset$. We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset which is also denoted by $\emptyset$.

We say that $a \in M$ if $M(a) \geq 1$, and the cardinality of M, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \to \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \to \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

Now we define the notion of a PCol automaton.

Definition 1. A *PCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ where V is an *alphabet*, the alphabet of the PCol automaton, its elements are called *objects*; $e \in V$ is the *environmental object* of the automaton; $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially; $(w_i, P_i), 1 \leq i \leq n$, is the i-th *cell*; and F is a set of *accepting configurations* of the PCol automaton.

For each cell, (w_i, P_i), $1 \leq i \leq n$, w_i is a multiset over V, it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system; P_i is a set of *programs*, where every program is formed from k rules of the following types:

- *tape rules* of the form $a \xrightarrow{T} b$, or $a \overset{T}{\leftrightarrow} b$, called rewriting tape rules and communication tape rules, respectively; or

- *nontape rules* of the form $a \rightarrow b$, or $c \leftrightarrow d$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

For each $i,\ 1 \leq i \leq n$, the set of *tape programs* is denoted by P_i^T, they are formed from one tape rule and $k-1$ nontape rules, the set of *nontape programs* which contain only nontape rules, is denoted by P_i^N, thus, $P_i = P_i^T \cup P_i^N,\ P_i^T \cap P_i^N = \emptyset$.

The *computation* of a PCol automaton starts in the initial configuration, and the configurations are changed by the cells with the application of some of their programs. The programs either change the objects inside the cells (with rewriting rules) or exchange them for other objects with the environment (with communication rules). During the computation, the PCol automaton processes an input word. The leftmost symbol of the yet non-read part of the input word is read during a configuration change if at least one cell applies a tape program which introduces the same symbol inside the cell as the symbol to be read either by rewriting or by communication.

A *configuration* of PCol automaton is an $(n+2)$-tuple $(u; u_E, u_1, \ldots, u_n)$, where $u \in V^*$ is the unprocessed (unread) part of the input string, $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i, \in V^*,\ 1 \leq i \leq n$, represents the contents of i-th cell. The *initial configuration* is given by $(w; w_E, w_1, \ldots, w_n)$, the input word to be processed by the system and the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(\varepsilon; v_E, v_1, \ldots, v_n)$.

To describe the computation process formally, we introduce the following notation. For any rule r we define four mappings as follows. Let $X \in \{T, \varepsilon\}$, and if $r = a \overset{X}{\rightarrow} b$, then let $left(r) = a$, $right(r) = b$, $export(r) = \varepsilon$, and $import(r) = \varepsilon$; if $r = a \overset{X}{\leftrightarrow} b$, then let $left(r) = \varepsilon$, $right(r) = \varepsilon$, $export(r) = a$, and $import(r) = b$ for $b \neq e$, or $import(r) = \varepsilon$ for $b = e$. Let us extend this notation also for programs. For $\alpha \in \{left, right, export, import\}$ and for any program p, let $\alpha(p) = \bigcup_{r \in p} \alpha(r)$ where for a rule r and program $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some $j,\ 1 \leq j \leq q$. Moreover, for any tape program p containing the tape rule $r \in p$, we also define the mapping $read(p)$ as $read(p) = right(r)$ if r is a rewriting tape rule, or $read(p) = import(r)$ if r is a communication tape rule.

Let the programs of each P_i be labeled in a one-to-one manner by labels from the set $lab(P_i)$, $1 \leq i \leq n$, $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$. In the following, for the sake of brevity, if no confusion arises, we designate programs and their labels with the same letters, thus, for a label $p \in lab(P_i)$, we also write $p \in P_i$.

Let $c = (u; u_E, u_1, \ldots, u_n)$ be a configuration of a PCol automaton Π. We call a *set of programs*, P_c, *applicable in configuration* c, if the following conditions hold.

- If $p, p' \in P_c, p \neq p'$ and $p \in P_i, p' \in P_j$, then $i \neq j$;
- for each $p \in P_c$, if $p \in P_i$ then $left(p) \cup export(p) = u_i$;
- for each $p \in P_c$, if p is a tape rule, then $read(p) = a$ where $u = au'$;
- $\bigcup_{p \in P_c} import(p) \subseteq u_E$.

A configuration $c = (au; u_E, u_1, \ldots, u_n)$, $a \in V$, is *changed* to a configuration $c' = (u'; u'_E, u'_1, \ldots, u'_n)$ by applying the set P_c of applicable programs if the following properties hold:

- If there is a $p \in P_c$ such that $p \in P_i$, then $u'_i = right(p) \cup import(p)$, otherwise $u'_i = u_i$, $1 \le i \le n$;
- $u_E = u_E - \bigcup_{p \in P_c} import(p) \cup \bigcup_{p \in P_c} export(p)$; and
- if there is a tape program $p \in P_c$ with $read(p) = a$, then $u' = u$, otherwise $u' = au$.

We say that a set P_c of applicable programs is *maximal* (with respect to a certain additional property), if for any $p' \in \bigcup_{i=1}^{n} P_i$ (having the same additional property) such that $p' \notin P_c$, the set of programs $P_c \cup \{p'\}$ is not applicable.

Based on the properties of P_c, the set of programs applied to a configuration $c = (au; u_E, u_1, \ldots, u_n)$, $a \in V$, we distinguish the following types of transitions. Let the configuration obtained after the application of P_c be denoted by $c' = (u'; u'_E, u'_1, \ldots, u'_n)$. We have a

- *t-transition*, denoted by $\Rightarrow_t$, if $u' = u$ and P_c is maximal set of programs with respect to the property that every $p \in P_c$ is a tape program with $read(p) = a$;
- *tmin-transition*, denoted by $\Rightarrow_{tmin}$, if $u' = u$ and P_c is maximal set of programs with at least one $p \in P_c$, such that p is a tape program with $read(p) = a$;
- *tmax-transition*, denoted as $\Rightarrow_{tmax}$, if $u' = u$ and $P_c = P_T \cup P_N$ where P_T is a maximal set of applicable tape programs with $read(p) = a$ for all $p \in P_T$, the set P_N is a set of nontape programs, and $P_c = P_T \cup P_N$ is maximal;
- *n-transition*, denoted by $\Rightarrow_n$, if $u' = au$ and P_c is maximal set of nontape programs.

A PCol automaton works in the *t* (*tmax*, *tmin*) *mode of computation* if it uses only *t*- (tmax-, tmin-) transitions. It works in the *nt* (*ntmax* or *ntmin*) mode if at any computation step it may use a t- (tmax- or tmin-) transition or an n-transition,

A special case of the *nt* mode is called *initial*, denoted by *init*, if the computation of the automaton is divided in two phases: first it reads the input strings using t-transitions and after reading all the input symbols it uses n-transitions to finish the computation.

Let us designate $M = \{t, nt, tmax, ntmax, tmin, ntmin, init\}$. The *language accepted by a PCol automaton Π* as above is defined as the set of strings which can be read during a successful computation:

$$\begin{aligned} L(\Pi, mode) = \{w \in V^* | &(w; w_E, w_1, \ldots, w_n) \text{ can be} \\ &\text{transformed by } \Pi \text{ into } (\varepsilon; v_E, v_1, \ldots, v_n) \in F \\ &\text{with a computation in mode } mode \in M\}. \end{aligned}$$

Let $\mathcal{L}(PColA, mode)$ denote the class of languages accepted by PCol automata in the computational mode $mode \in M$, and let RE denote the class of recursively enumerable languages.

Now we demonstrate the above defined notions by an example.

Example 1. Let $\mathit{\Pi} = (\{a, b, c\}, e, w_E, (w_1, P_1), (w_2, P_2), (w_3, P_3), F)$ be a PCol automaton, where the sets of programs are defined as

P_1	P_2	P_3
$p_1 : \langle e \xrightarrow{T} b; e \leftrightarrow a\rangle,$	$p_A : \langle e \overset{T}{\leftrightarrow} a; e \leftrightarrow a\rangle,$	$p_I : \langle e \xrightarrow{T} a; e \leftrightarrow a\rangle,$
$p_2 : \langle e \xrightarrow{T} a; e \rightarrow a\rangle,$	$p_B : \langle e \overset{T}{\leftrightarrow} b; e \leftrightarrow a\rangle,$	$p_{II} : \langle e \xrightarrow{T} a; e \rightarrow b\rangle,$
$p_3 : \langle e \leftrightarrow a; e \leftrightarrow e\rangle,$	$p_C : \langle e \rightarrow a; e \leftrightarrow e\rangle,$	$p_{III} : \langle e \rightarrow a; e \leftrightarrow e\rangle,$

and $c = (w; aa, ee, ee, ee)$ is the current configuration of $\mathit{\Pi}$.

If $w = bw'$ for some $b \in V,\ w' \in V^*$, then $\mathit{\Pi}$ can execute a t-transition by applying the set of programs $P_c = \{p_1, p_B\}$ since the third cell has no applicable tape program. For a tmax-transition, $\mathit{\Pi}$ can apply the set $P_c = \{p_1, p_B, p_{III}\}$, in this case the third cell can use its applicable nontape program. For a tmin-transition, $\mathit{\Pi}$ has to choose one from three possible sets of programs $\{p_1, p_B, p_{III}\}$, $\{p_1, p_C, p_{III}\}$, or $\{p_3, p_B, p_{III}\}$. For an n-transition, $\mathit{\Pi}$ has to use the set $P_c = \{p_3, p_C, p_{III}\}$.

If $w = aw',\ a \in V,\ w' \in V^*$, then the sets of applicable programs for the different transition types are given in the following table.

transition types	applicable sets of programs
$t,\ tmax$	$\{p_2, p_A, p_{II}\}$
$tmin$	$\{p_2, p_A, p_{III}\}, \{p_2, p_C, p_I\}, \{p_2, p_C, p_{II}.\}, \{p_2, p_C, p_{III}\},$ $\{p_3, p_A, p_{II}\}, \{p_3, p_A, p_{III}\}, \{p_3, p_C, p_I\}, \{p_3, p_C, p_{II}\}$
n	$\{p_3, p_C, p_{III}\}$

If $w = cw',\ c \in V,\ w' \in V^*$, then there is no cell with an applicable tape program. The only set of applicable programs is the set $P_c = \{p_3, p_C, p_{III}\}$ for an n-transition.

Example 2. Let $L \subseteq \mathit{\Sigma}^*$ be a regular language, and let $M = (\mathit{\Sigma}, Q, \delta, q_0, F)$ be a finite automaton with $L(M) = L$, with alphabet $\mathit{\Sigma}$, set of states Q, initial state q_0, set of final states F, and transition function $\delta : \mathit{\Sigma} \times Q \rightarrow Q$.

It is not difficult to see that the PCol automaton $\mathit{\Pi} = (\mathit{\Sigma} \cup Q, e, (w, P), F')$ of capacity two simulates the computation of M, with initial cell contents $w = aq_0$ for some $a \in \mathit{\Sigma}$, set of rules

$$P = \{\langle x \xrightarrow{T} a,\ q \rightarrow q'\rangle \mid \text{for all } x \in \mathit{\Sigma} \text{ such that } \delta(x, q) = q' \text{ for some } q, q' \in Q\},$$

and set of final configurations

$$F' = \{(\varepsilon; \varepsilon, xq_f) | \text{for all } x \in \mathit{\Sigma}, \text{ and } q_f \in F\}.$$

Since P contains only tape programs, $\mathit{\Pi}$ cannot execute any n-transitions, and since it has only one cell, $L(\mathit{\Pi}, t) = L(\mathit{\Pi}, tmax) = L(\mathit{\Pi}, tmin) = L(M)$.

3 PCol automata computing in the modes with n-transitions

In this section we show that if we consider the functioning modes which allow n-transitions at arbitrary points of the computational process, then PCol automata characterize the class of recursively enumerable languages. First we recall the notion of a two-counter machine which will be used in the proof.

A *two-counter machine*, see [8], $M = (\Sigma \cup \{Z, B\}, E, R, q_0, q_F)$ is a 3-tape Turing machine where Σ is an *alphabet*, E is a set of *internal states* with $q_0, q_F \in E$ being the initial and the final states, and R is a set of *transition rules*. The machine has a read-only input tape and two semi-infinite storage tapes which are used as counters. The alphabet of the storage tapes contains only two symbols, Z and B (blank), while the alphabet of the input tape is $\Sigma \cup \{B\}$. The symbol Z is written on the first, leftmost cells of the storage tapes which are scanned initially by the tape heads. An integer t can be stored by moving a tape head t cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. The machine is capable of checking whether a stored value is *zero* or not by looking at the symbol scanned by the tape heads. If the scanned symbol is Z, then the value stored in the corresponding counter is *zero*.

Without the loss of generality, we assume that two-counter machines check and modify only one of their counters during any transition, thus, the rule set R contains transition rules of the form $(q, x, c_i) \to (q', e)$ where $x \in \Sigma \cup \{B\} \cup \{\lambda\}$ corresponds to the symbol scanned on the input tape in state $q \in E$, and $c_i \in \{Z, B\}$, $i \in \{1, 2\}$ correspond to the symbols scanned on the ith storage tape. By a rule of the above form, M enters state $q' \in E$, and the ith counter is modified according to $e \in \{-1, 0, +1\}$. If $x \in \Sigma \cup \{B\}$, then the machine was scanning x on the input tape, and the head moves one cell to the right; if $x = \varepsilon$, then the machine performs the transition irrespective of the scanned input symbol, and the reading head does not move.

A word $w \in \Sigma^*$ is accepted by the two-counter machine if starting in the initial state q_0, the input head reaches and reads the rightmost non-blank symbol on the input tape, and the machine is in the accepting state q_F. Two-counter machines are computationally complete; they are just as powerful as Turing machines.

Theorem 1.

$$\mathcal{L}(PColA, X) = RE, \ \textit{where } X \in \{nt, ntmax, ntmin\}.$$

Proof. Let $L \in \Sigma^*$ be an arbitrary recursively enumerable language, and let $M = (\Sigma, Q, q_0, q_f, Tr)$ be a two-counter machine as above with $L = L(M)$.

Let us construct the PCol automaton $\Pi = (V, e, w_E, (w_1, P_1), (w_2, P_2), F)$ where $V = \Sigma \cup Q \cup \{t, t', t'', t''' \mid t \in Tr\} \cup \{c_1, c_2, A\}$, $w_1 = q_0 e$, $w_2 = ee$, $F = \{(\varepsilon; u, q_f e, ee) \mid u \in V^*\}$, and the sets of programs are defined as follows.

For any $\alpha \in \{B, Z\}$, $\beta \in \{-1, 0, +1\}$, we define the disjoint sets of transitions $Tr_{\alpha,\beta} \subseteq Tr$ as follows: $t \in Tr_{\alpha,\beta}$, if and only if, $t : (q, x, i, \alpha) \to (w, \beta)$, $x \in \Sigma \cup \{\varepsilon\}$, $i \in \{1, 2\}$. Thus, $Tr = Tr_{B,-1} \cup Tr_{B,0} \cup Tr_{B,+1} \cup Tr_{Z,0} \cup Tr_{Z,+1}$.

Now we define the sets of programs as

$$P_1 = \bigcup_{t \in Tr} P_{1,t} \text{ and } P_2 = \bigcup_{t \in Tr} P_{2,t},$$

where for $t \in (Tr_{B,-1} \cup Tr_{B,0})$ we have

$$P_{1,t} = \{p_{t,1} : \langle q \to t; r_{t,1}\rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2}\rangle, p_{t,3} : \langle t' \to t''; c_i \to e\rangle,$$
$$p_{t,4} : \langle t'' \to e; e \leftrightarrow t'''\rangle, p_{t,5} : \langle t''' \to s; e \to e\rangle\},$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \xrightarrow{T} a$ and $a \to t'$, respectively, if $t \in Tr$ is such, that $x = a \in \Sigma$, otherwise, if $x = \varepsilon$, then $r_{t,1} = e \to e$ and $r_{t,2} = e \to t'$.

If $t \in Tr_{B,+1}$, then we have

$$P_{1,t} = \{p_{t,1} : \langle q \to t; r_{t,1}\rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2}\rangle, p_{t,3} : \langle t' \to t''; c_i \to c_i\rangle,$$
$$p_{t,4} : \langle t'' \to e; c_i \leftrightarrow t'''\rangle, p_{t,5} : \langle t''' \to s; e \to e\rangle\}$$

with $r_{t,1}$ and $r_{t,2}$ as above.

For these types of transitions, the set P_2 is defined as follows. If $t \in Tr_{B,-1}$, then we have

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to e\rangle, p_{t,7} : \langle t \to t'''; e \to e\rangle, p_{t,8} : \langle t''' \leftrightarrow e; e \to e\rangle\},$$

otherwise, if $t \in (Tr_{B,0} \cup Tr_{B,+1})$, then

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to c_i\rangle, p_{t,7} : \langle t \to t'''; c_i \leftrightarrow e\rangle\}.$$

Now, if $t \in Tr_{Z,0}$, then we have in P_1

$$P_{t,1} = \{p_{t,1} : \langle q \to t; r_{t,1}\rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2}\rangle, p_{t,3} : \langle e \leftrightarrow c_i; t \to A\rangle,$$
$$p_{t,4} : \langle t' \to e; e \leftrightarrow t''\rangle, p_{t,5} : \langle e \leftrightarrow e; t'' \to s\rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \xrightarrow{T} a$ and $a \to t'$, respectively, if the transition is such, that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t,1} = e \to e$ and $r_{t,2} = e \to t'$.

If $t \in Tr_{Z,+1}$, then

$$P_{t,1} = \{p_{t,1} : \langle q \to t; r_{t,1}\rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2}\rangle, p_{t,3} : \langle e \leftrightarrow c_i; t \to A\rangle,$$
$$p_{t,4} : \langle t' \to c_i; e \leftrightarrow t''\rangle, p_{t,5} : \langle c_i \leftrightarrow e; t'' \to s\rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are as above.

The set P_2 contains only two programs in both cases, these are defined as

$$P_{t,2} = \{p_{t,6} : \langle e \leftrightarrow t; e \to t''\rangle, p_{t,7} : \langle t \to e; t'' \leftrightarrow e\rangle\}$$

for all $t \in (Tr_{Z,0} \cup Tr_{Z,+1})$.

The PCol automaton Π simulates the work of the two-counter machine M by reading the input symbols with its tape programs and keeping track of the contents of the ith counter as the number of c_i, $i \in \{1,2\}$ objects present in the environment.

Each transition of M is simulated separately. One of the symbols inside the first cell of Π is from Q, it corresponds to the internal state of M during the simulation process. This symbols is changed through a series of programs into the symbols $s \in Q$ if and only if, M can also change its state from q to s while the counter contents are also checked and modified with an interplay of programs from the two cells of Π.

The reader may check that the PCol automaton Π may reach a final configuration after reading the whole input, if and only if the simulated two-counter machine is able to reach the internal state q_f after processing the same input string using its transitions from Tr.

4 The other computation modes

First we consider the power of the t, *tmax*, and *tmin* computation modes. Note that a PCol automaton working in these modes reads one input symbol in every computational step, thus, the length of the computation cannot be more than the length of the input string.

As we have seen in Example 2, any regular language can be accepted by a PCol automaton with one cell. Now we present an example showing that the class of languages characterized by PCol automata in the t, *tmax*, or *tmin* modes contains non-context-free languages.

Example 3. There exists PCol automaton accepting language $L = \{a^n b^n c^n \mid n \geq 0\}$ in any of the computation modes t, *tmax*, or *tmin*. To see this, we construct $\Pi = (\{a,b\}, e, \varepsilon, (w,P), F)$ where $w = ea$,

$$P = \{\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle, \langle a \xrightarrow{T} b; e \leftrightarrow a\rangle, \langle a \xrightarrow{T} b; b \leftrightarrow a\rangle,$$
$$\langle a \xrightarrow{T} c; b \rightarrow b\rangle, \langle b \xrightarrow{T} c; c \leftrightarrow b\rangle\},$$

and $F = \{(\varepsilon; u, cb), (\varepsilon; u, ea) \mid u \in \{c\}^*\}$.

To see how Π works, consider a computation for the input word *aabbcc*.

After the last step, the input tape is read and the automaton is in the final state $(\varepsilon; c, cb)$. It is not difficult to see that Π can only reach a final state if the input is of the form $\{a\}^*\{b\}^*\{c\}^*$ with an equal number of each type of symbols.

Since P contains only tape programs, Π cannot execute any n-transitions, and since it has only one cell, $L(\Pi,t) = L(\Pi, tmax) = L(\Pi, tmin) = L$

step	cell	environment	unread part of tape	applied program
1.	ea	ε	$\mathbf{a}abbcc$	$\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle$
2.	ae	a	$\mathbf{a}bbcc$	$\langle e \xrightarrow{T} a; a \leftrightarrow e\rangle$
3.	ae	aa	$\mathbf{b}bcc$	$\langle a \xrightarrow{T} b; e \leftrightarrow a\rangle$
4.	ba	a	$\mathbf{b}cc$	$\langle a \xrightarrow{T} b; b \leftrightarrow a\rangle$
5.	ba	b	$\mathbf{c}c$	$\langle a \xrightarrow{T} c; b \rightarrow b\rangle$
6.	cb	b	$\mathbf{c}$	$\langle b \xrightarrow{T} c; c \leftrightarrow b\rangle$
7.	cb	c	ε	

Now we consider the initial mode. This time, although the computations can be of arbitrary length, n-transitions can only be executed after the whole input string is processed. The next example demonstrates that the class of languages characterized by PCol automata in the initial mode contains non-semilinear languages.

Example 4. There exists a PCol automaton Π, such that $L(\Pi, init) = \{a^{2^n}\}$. To see this, consider the PCol automaton $\Pi = (V, e, \varepsilon, (ee, P_1), (ee, P_2), F)$ where $V = \{a, b, B, c, c', d, d', f, f', g, g', i, i', \underline{i}, \underline{\underline{i}}, \underline{i'}, \underline{\underline{i'}}, x, x', x'', x''', \overline{x}, \overline{\overline{x}}, x_h, y, y', y'', z, u, u', u'', u''', \overline{u}, \overline{\overline{u}}, v, v', v''\}$, $F = \{(\varepsilon; \varepsilon, x_h e, ee)\}$, and

$$P_1 = P_{1,in} \cup P_{1,div} \cup P_{1,b} \cup P_{1,B} \cup P_{1,tran} \cup P_{1,fin}, \text{ and } P_2 = P_{2,b} \cup P_{2,B}$$

where the set of programs are defined as follows.

$$P_{1,in} = \{p_1 : \langle e \xrightarrow{T} a; e \rightarrow b\rangle, p_2 : \langle a \xrightarrow{T} a; b \leftrightarrow e\rangle, p_3 : \langle a \xrightarrow{T} a; e \rightarrow b\rangle\}.$$

Using these programs, the first cell reads the input symbols and puts one object b into the environment after reading two as.

After reading the input, the cells may replace two bs by one B, or two Bs by one b. This is achieved by the programs:

$$\begin{aligned}P_{1,div} = \{&p_4 : \langle a \rightarrow c; e \leftrightarrow b\rangle, p_5 : \langle c \rightarrow d; b \rightarrow e\rangle, p_6 : \langle d \rightarrow f; e \leftrightarrow b\rangle,\\ &p_7 : \langle f \rightarrow g; b \rightarrow B\rangle, p_8 : \langle g \rightarrow i; B \leftrightarrow e\rangle, p_9 : \langle i \rightarrow c; e \leftrightarrow b\rangle,\\ &p_{10} : \langle i' \rightarrow c'; e \leftrightarrow B\rangle, p_{11} : \langle c' \rightarrow d'; B \rightarrow e\rangle,\\ &p_{12} : \langle d' \rightarrow f'; e \leftrightarrow B\rangle, p_{13} : \langle f' \rightarrow g'; B \rightarrow b\rangle,\\ &p_{14} : \langle g' \rightarrow i'; b \leftrightarrow e\rangle\}.\end{aligned}$$

After exchanging the bs to Bs or reversely, the cells have to control if there is any remaining bs or Bs and only in the negative case an the computation continue. This is done by the interplay of the programs

$$\begin{aligned}P_{1,b} = \{&p_{15} : \langle e \rightarrow x'; x \leftrightarrow e\rangle, p_{16} : \langle e \rightarrow x''; x' \leftrightarrow e\rangle p_{17} : \langle x'' \rightarrow x'''; e \leftrightarrow e\rangle,\\ &p_{18} : \langle x''' \rightarrow \overline{x}; e \leftrightarrow e\rangle, p_{19} : \langle \overline{x} \rightarrow \overline{\overline{x}}; e \leftrightarrow e\rangle, p_{20} : \langle \overline{\overline{x}} \rightarrow \underline{i}; e \leftrightarrow y''\rangle,\\ &p_{21} : \langle \overline{\overline{x}} \rightarrow \underline{i'}; e \leftrightarrow y\rangle, p_{22} : \langle y'' \rightarrow i; \underline{i} \leftrightarrow e\rangle, p_{23} : \langle y \rightarrow z; \underline{i'} \leftrightarrow e\rangle,\\ &p_{24} : \langle e \rightarrow \underline{\underline{i'}}; z \leftrightarrow \underline{i'}\rangle, p_{25} : \langle \underline{i'} \rightarrow i'; \underline{\underline{i'}} \leftrightarrow e\rangle\},\end{aligned}$$

$$\begin{aligned} P_{2,b} = \{ & p_{26} : \langle e \to y; e \leftrightarrow x \rangle, p_{27} : \langle x \to y'; y \leftrightarrow x' \rangle, p_{28} : \langle x' \to y''; y' \leftrightarrow b \rangle, \\ & p_{29} : \langle b \to b; y'' \leftrightarrow y \rangle, p_{30} : \langle y \to e; b \leftrightarrow e \rangle, p_{31} : \langle x' \to e; y' \leftrightarrow z \rangle, \\ & p_{32} : \langle z \to e; e \leftrightarrow e \rangle \}, \end{aligned}$$

for checking bs, and the programs

$$\begin{aligned} P_{1,B} = \{ & p_{33} : \langle e \to u'; u \leftrightarrow e \rangle, p_{34} : \langle e \to u''; u' \leftrightarrow e \rangle, p_{35} : \langle u'' \to u'''; e \leftrightarrow e \rangle, \\ & p_{36} : \langle u''' \to \overline{u}; e \leftrightarrow e \rangle, p_{37} : \langle \overline{u} \to \overline{\overline{u}}; e \leftrightarrow e \rangle, p_{38} : \langle \overline{\overline{u}} \to \underline{i'}; e \leftrightarrow v'' \rangle, \\ & p_{39} : \langle \overline{\overline{u}} \to \underline{i}; e \leftrightarrow v \rangle, p_{40} : \langle v'' \to i'; \underline{i'} \leftrightarrow e \rangle, p_{41} : \langle v \to w; \underline{i} \leftrightarrow e \rangle, \\ & p_{42} : \langle e \to \underline{\underline{i}}; w \leftrightarrow \underline{i} \rangle, p_{43} : \langle \underline{i} \to i; \underline{\underline{i}} \leftrightarrow e \rangle \}, \end{aligned}$$

$$\begin{aligned} P_2 = \{ & p_{44} : \langle e \to v; e \leftrightarrow u \rangle, p_{45} : \langle u \to v'; v \leftrightarrow u' \rangle, p_{46} : \langle u' \to v''; v' \leftrightarrow B \rangle, \\ & p_{47} : \langle B \to B; v'' \leftrightarrow v \rangle, p_{48} : \langle v \to e; B \leftrightarrow e \rangle, p_{49} : \langle u' \to e; v' \leftrightarrow w \rangle, \\ & p_{50} : \langle w \to e; e \leftrightarrow e \rangle \}, \end{aligned}$$

for checking Bs.

The following programs are used for connecting the different phases of the functioning of the system,

$$P_{1,tran} = \{ p_{51} : \langle i \to x; e \leftrightarrow e \rangle, p_{52} : \langle i' \to u; e \leftrightarrow e \rangle \},$$

and for finishing the computation

$$P_{1,fin} = \{ p_{53} : \langle d \to x_h; e \leftrightarrow e \rangle, p_{54} : \langle d' \to x_h; e \leftrightarrow e \rangle, p_{55} : \langle a \to x_h; b \to e \rangle \}.$$

5 Conclusion

P colony automata are very simple language recognizing devices, with strong formal resemblance to finite automata. Especially interesting are those cases when the function of these constructs is governed by the use of their tape rules, i.e., the computational modes t, $tmin$ and $tmax$. The description of the exact computational power of these variants of PCol automata is a challenging problem. We guess to obtain language classes of very low complexity.

References

1. L. Ciencialová, L. Cienciala, Variation on the theme: P colonies. In: Proc. 1st Intern. Workshop on Formal Models. (D. Kolăr, 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. Cienciala, L. Ciencialová, A. Kelemenová, Homogeneous P colonies. Computing and Informatics 27 (2008), 481–496.
4. 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.
5. 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.
6. 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.
7. E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil, P automata. Chapter 6, In: The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2010, 144–167.
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), 569–502.
11. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems 23 (1992), 621–633.
12. 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.
13. 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.
14. M. Minsky, Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
15. Gh. Păun, Membrane Computing – An Introduction. Springer-Verlag, Berlin, 2002.
16. The Oxford Handbook of Membrane Computing. (Gh. Păun, G. Rozenberg, A. Salomaa, eds.) Oxford University Press, 2010.