Deterministic Parsing with P Colony Automata

Erzsébet Csuhaj-Varjú[1], Kristóf Kántor[2], and György Vaszil[2](✉)

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

[2] Department of Computer Science, Faculty of Informatics,
University of Debrecen, Kassai út 26, Debrecen 4028, Hungary
{`kantor.kristof`,`vaszil.gyorgy`}`@inf.unideb.hu`

Abstract. We investigate the possibility of the deterministic parsing (that is, parsing without backtracking) of languages described by (generalized) P colony automata. We define a subclass of these computing devices satisfying a property which resembles the LL(k) property of context-free grammars, and study the possibility of parsing the characterized languages using a k symbol lookahead, as in the LL(k) parsing method for context-free languages.

1 Introduction

The computational model called P colony is similar to tissue-like membrane systems. In P colonies, multisets of objects are used to describe the contents of cells and the environment, and these multisets are processed by the cells in the corresponding colony using rules which enable the evolution of the objects present in the cells or the exchange of objects between the environment and the cells. These computing agents have a very confined functionality: they can store a restricted amount of objects at a given time (this is called the capacity of the cell; every cell has the same capacity) and they can process a restricted amount of information. The way of information processing is very simple: The rules are either of the form $a \rightarrow b$ (for changing an object a into an object b inside the cell), or $a \leftrightarrow b$ (for exchanging an object a inside a cell with an object b in the environment). A program is a rule set with exactly the same number of rules as the capacity of the cell. When a program is executed, the k rules (the capacity of the cell) that it contains are applied to the k objects simultaneously. A configuration of a P colony with n cells is a an n-tuple of multisets of objects, those which are present inside the cells. During a computational step, a maximal number of cells of the P colony execute one of their programs in parallel. A computation ends when the P colony reaches one of its final configurations (usually given as the set of halting configurations, that is, when no program can be applied by any of the cells).

C. Graciani et al. (Eds.): Pérez-Jiménez Festschrift, LNCS 11270, pp. 88–98, 2018.
https://doi.org/10.1007/978-3-030-00265-7_8

There are many theoretical results concerning P colonies. Despite the fact that they are extremely simple computing systems, they are computationally complete, even with very restricted size parameters and other syntactic or functioning restrictions. For these, and more topics, results, see [4,5,7–10,12,13,17,18] and for summaries consult [6,22].

P colony automata were introduced in [3]. They are called automata, because they accept string languages by assuming an initial input tape with an input string in the environment. The available types of rules are extended by so-called tape rules. These types of rules in addition to manipulating the objects as their non-tape counterparts, also read the processed objects from the input tape.

To overcome the difficulty that different tape rules can read different symbols in the same computational step, generalized P colony automata were introduced in [19] and studied further in [20,21]. The main idea of this computational model was to get the process of input reading closer to other kinds of membrane systems, especially to antiport P systems and P automata. The latter, introduced in [14] (see also [11]) are P systems using symport and antiport rules (see [23]), characterizing string languages.

This generality is used in the generalized P colony automata theory, that is, the idea of characterizing strings through the sequences of multisets processed during computations. A computation in this model defines accepted multiset sequences, which are transformed into accepted symbol sequences/strings. In this model there is no input string, but there are tape rules and non-tape rules equally for evolution and communication rules. In a single computational step, this system is able to read more than one symbol, thus reading a multiset. This way generalized P colony automata are able to avoid the conflicts present in P colony automata, where simultaneous usage of tape rules in a single computational step can arise problems. After getting the result of a computation, that is, the accepted sequence of multisets, it is possible to map them to strings in a similar way as shown in P automata.

In [19], some basic variants of the model were introduced and studied from the point of view of their computational power. In [20,21] we continued the investigations structuring our results around the capacity of the systems, and different types of restrictions imposed on the use of tape rules in the programs.

The concept of a P colony automata has been developed into another direction as well, namely, those variants have been introduced and studied where the environment consists of a string and the communication rules of the cells are for substituting (replacing a symbol with another symbol), inserting, or erasing symbols of the current environmental string [2]. These constructs are called APCol systems (Automaton-like P colonies). Notice that this model essentially differs from the generalized P colony automaton since the whole environment is given in advance and in the form of a string. APCol systems are also able to obtain the full computational power, i.e., they are computationally complete [4,5].

Since P colony automata variants accept languages, different types of characterizations of their language classes are of interest. One possible research direction could be investigating their parsing properties in terms of programs and rules of the (generalized) P colony automata. In this paper we study the possibility of deterministically parsing the languages characterized by these devices. We define the so-called LL(k) condition for these types of automata, which enables deterministic parsing with a k symbol lookahead, as in the case of context-free LL(k) languages, and present an initial result showing that using generalized P colony automata we can deterministically parse context-free languages that are not LL(k) in the "original" sense.

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. The number of occurrences of a symbol a in w where $a \in V$ is denoted by $|w|_a$.

A multiset over a set V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ denotes the set of non-negative integers. This mapping assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The set $supp(M) = \{a \mid M(a) \geq 1\}$ is the support of M. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $supp(M) = \emptyset$. The set of finite multisets over the alphabet V is denoted by $\mathcal{M}(V)$. A finite multiset M over V will also be represented by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, the empty multiset will be 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 $M_1, M_2 \in \mathcal{M}(V)$, $M_1 \subseteq M_2$ 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 $a \in V$.

A *genPCol 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 automaton, its elements are called *objects*;
- $e \in V$ is the *environmental object* of the automaton, the only object which is assumed to be available in an arbitrary, unbounded number of copies in the environment;
- $w_E \in (V - \{e\})^*$ is a string representing a multiset from $\mathcal{M}(V - \{e\})$, the multiset of objects different from e which is found in the environment initially;
- $(w_i, P_i), 1 \leq i \leq n$, specifies the i-th *cell* where w_i is (the representation of) 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*, each program is formed from k rules of the following types (where $a, b \in V$):
  - *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 $a \leftrightarrow b$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

  A program is called a *tape program* if it contains at least one tape rule.
- F is a set of *accepting configurations* of the automaton which we will specify in more detail below.

A genPCol automaton reads an input word during a computation. A part of the input (possibly consisting of more than one symbol) is read during each configuration change: the processed part of the input corresponds to the multiset of symbols introduced by the tape rules of the system.

A *configuration* of a genPCol automaton is an $(n+1)$-tuple $(u_E, u_1, \ldots, u_n)$, where $u_E \in \mathcal{M}(V - \{e\})$ is the multiset of objects different from e in the environment, and $u_i \in \mathcal{M}(V)$, $1 \leq i \leq n$, are the contents of the i-th cell. The *initial configuration* is given by $(w_E, w_1, \ldots, w_n)$, 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 $(v_E, v_1, \ldots, v_n)$, where

- $v_E \in \mathcal{M}(V - \{e\})$ denotes a multiset of objects different from e being in the environment, and
- $v_i \in \mathcal{M}(V)$, $1 \leq i \leq n$, is the contents of the i-th cell.

In order to describe the functioning of genPCol automata, let us define the following multisets. Let r be a rewriting or a communication rule (tape or nontape), and let us denote by $left(r)$ and $right(r)$ the objects on the left and on the right side of r, respectively. Let also, for $\alpha \in \{left, right\}$ and for any program p, $\alpha(p) = \bigcup_{r \in p} \alpha(r)$ where the union denotes multiset union (as defined above), and for a rule r and a program $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some j, $1 \leq j \leq k$.

Moreover, for any tape program p we also define the multiset of symbols $read(p) = \bigcup_{r \in p, r = a \xrightarrow{T} b, b \neq e} right(r) \cup \bigcup_{r \in p, r = a \overset{T}{\leftrightarrow} b, a \neq e} left(r)$, the multiset of symbols (different from e) on the right side of rewriting tape rules and on the left side of communication tape rules. If p is not a tape program, that is, p contains no tape rules, then $read(p) = \emptyset$.

For all communication rules r of the program p, let $export(p) = \bigcup_{r \in p} left(r)$ and $import(p) = \bigcup_{r \in p} right(r)$, the multiset of objects that are sent out to the environment and brought inside the cell when applying the program p, respectively.

Moreover, $create(p) = \bigcup_{r \in p} right(p)$ for the rewriting rules r of p, that is, $create(p)$ is the multiset of symbols produced by the rewriting rules of program p.

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of a genPCol automaton $\mathit{\Pi}$, and let $U_E = u_E \cup \{e, e, \ldots\}$, thus, the multiset of objects found in the environment (together with the infinite number of copies of e, denoted as $\{e, e, \ldots\}$, which are always present). The *sequence of programs*

$$(p_1, \ldots, p_n) \in (P_1 \cup \{\#\}) \times \ldots \times (P_n \cup \{\#\})$$

is *applicable in configuration* c, if the following conditions hold.

- The selected programs are applicable in the cells (the left sides of the rules contain the same symbols that are present in the cell), that is, for each $1 \leq i \leq n$, if $p_i \in P_i$ then $left(p_i) = u_i$;
- the symbols to be brought inside the cells by the programs are present in the environment, that is, $\bigcup_{p_i \neq \#, 1 \leq i \leq n} import(p_i) \subseteq U_E$;
- the set of selected programs is maximal, that is, if any $p_i = \#$ is replaced by some $p_i' \in P_i$, $1 \leq i \leq n$, then the above conditions are not satisfied any more.

The set of all applicable sequences of programs in the configuration $c = (u_E, u_1, \ldots, u_n)$ is denoted by App_c, that is,

$$App_c = \{P_c = (p_1, \ldots, p_n) \in (P_1 \cup \{\#\}) \times \ldots \times (P_n \cup \{\#\}) \mid \text{ where } P_c \\ \text{is a sequence of applicable programs in the configuration } c\}.$$

A configuration c is called *a halting configuration* if the set of applicable sequences of programs is the singleton set $App_c = \{(p_1, \ldots, p_n) \mid p_i = \# \text{ for all } 1 \leq i \leq n\}$.

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of the genPCol automaton. By applying a sequence of applicable programs $P_c \in App_c$, the configuration c is *changed* to a configuration $c' = (u_E', u_1', \ldots, u_n')$, denoted by $c \overset{P_c}{\Longrightarrow} c'$, if the following properties hold:

- If $(p_1, \ldots, p_n) = P_c \in App_c$ and $p_i \in P_i$, then $u_i' = create(p_i) \cup import(p_i)$, otherwise, if $p_i = \#$, then $u_i' = u_i$, $1 \leq i \leq n$. Moreover,
- $U_E' = U_E - \bigcup_{p_i \neq \#, 1 \leq i \leq n} import(p_i) \cup \bigcup_{p_i \neq \#, 1 \leq i \leq n} export(p_i)$ (where U_E' again denotes $u_E' \cup \{e, e, \ldots\}$ with an infinite number of copies of e).

Thus, in genPCol automata, we apply the programs in the maximally parallel way, that is, in each computational step, every component cell nondeterministically applies one of its applicable programs. Then we collect all the symbols that the tape rules "read" (these multisets are denoted by $read(p)$ for a program p above): this is the multiset read by the system in the given computational step.

For any P_c sequence of applicable programs in a configuration c, $read(P_c)$ denotes the multiset of objects read by the tape rules of the programs of P_c, that is,

$$read(P_c) = \bigcup_{p_i \neq \#,\ (p_1, \ldots, p_n) = P_c} read(p_i).$$

Then we can also define the set of multisets which can be read in any configuration of the genPCol automaton Π as

$$in(\Pi) = \{read(P_c) \mid P_c \in App_c\}.$$

Remark 1. Although the set of configurations of a genPCol automaton Π can be infinite (because the multiset corresponding to the contents of the environment is not necessarily finite), the set $in(\Pi)$ is always finite. To see this, note that the applicability of a program by a component cell also depends on the contents of the particular component. Since at most one program can be applied in a component in one computational step, and the number of programs associated to each component is finite, the number of different sequences of applicable programs in any configuration, that is, the cardinality of the set App_c is finite.

A successful computation defines this way an accepted sequence of multisets: $u_1 u_2 \dots u_s,\ u_i \in in(\Pi)$, for $1 \leq i \leq s$, that is, the sequence of multisets entering the system during the steps of the computation.

Let $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$ be a genPCol automaton. The *set of input sequences accepted by* Π is defined as

$$\begin{aligned} A(\Pi) = \{u_1 u_2 \dots u_s \mid u_i \in in(\Pi),\ 1 \leq i \leq s,\ &\text{and there is a configuration} \\ &\text{sequence } c_0, \dots, c_s, \text{ with } c_0 = (w_E, w_1, \dots, w_n),\ c_s \in F,\ c_s \text{ halting,} \\ &\text{and} \quad c_i \overset{P_{c_i}}{\Longrightarrow} c_{i+1} \text{ with } u_{i+1} = read(P_{c_i}) \text{ for all } 0 \leq i \leq s-1\}. \end{aligned}$$

Let Π be a genPCol automaton, and let $f : in(\Pi) \to 2^{\Sigma^*}$ be a mapping, such that $f(u) = \{\varepsilon\}$ if and only if u is the empty multiset.

The *language accepted by* Π with respect to f is defined as

$$L(\Pi, f) = \{f(u_1) f(u_2) \dots f(u_s) \in \Sigma^* \mid \ u_1 u_2 \dots u_s \in A(\Pi)\}.$$

The class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ is denoted

- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k))$ when all the communication rules are tape rules,
- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k))$ when all the programs must have at least one tape rule, and
- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ when programs with any kinds of rules are allowed.

Let V and Σ be two alphabets, and let $\mathcal{M}_{FIN}(V) \subseteq \mathcal{M}(V)$ denote the set of finite subsets of the set of finite multisets over an alphabet V. Consider a mapping $f : D \to 2^{\Sigma^*}$ for some $D \in \mathcal{M}_{FIN}(V)$. We say that $f \in \mathcal{F}_{\text{TRANS}}$, if for any $v \in D$, we have $|f(v)| = 1$, and we can obtain $f(v) = \{w\}$, $w \in \Sigma^*$ by applying a deterministic finite transducer to any string representation of the multiset v, (as w is unique, the transducer must be constructed in such a way that all string representations of the multiset v as input result in the same $w \in \Sigma^*$ as output, and moreover, as f should be nonerasing, the transducer produces a result with $w \neq \varepsilon$ for any nonempty input).

Besides the above defined class of mappings, we also use the so called permutation mapping. Let $f_{perm} : \mathcal{M}(V) \to 2^{\Sigma^*}$ where $V = \Sigma$ be defined as follows. For all $v \in \mathcal{M}(V)$, we have

$$\begin{aligned} f_{perm}(v) = \{a_{\sigma(1)} a_{\sigma(2)} \dots a_{\sigma(s)} \mid v = a_1 a_2 \dots a_s &\text{ for some permutation} \\ &\sigma \text{ of } \{1, \dots, s\}\ \}. \end{aligned}$$

We denote the language classes that can be characterized with these types of input mappings as $\mathcal{L}_X(\text{genPCol}, Y(k))$, where $X \in \{perm, \text{TRANS}\}$, $Y \in \{\text{com-tape}, \text{all-tape}, *\}$.

Now we recall an example from [20] to demonstrate the above defined notions.

Example 1. Let $\Pi = (\{a, b, c\}, e, \emptyset, (ea, P), F)$ be a genPCol automaton where

$$P = \{p_1 : \langle e \to a, a \overset{T}{\leftrightarrow} e\rangle,\ p_2 : \langle e \to b, a \overset{T}{\leftrightarrow} e\rangle,\ p_3 : \langle e \to b, b \overset{T}{\leftrightarrow} a\rangle,$$
$$p_4 : \langle e \to c, b \overset{T}{\leftrightarrow} a\rangle,\ p_5 : \langle a \to b, b \overset{T}{\leftrightarrow} a\rangle,\ p_6 : \langle a \to c, b \overset{T}{\leftrightarrow} a\rangle\}$$

with all the communication rules being tape rules. Let $F = \{(v, ca) \mid a \notin v\}$ be the set of final configurations.

The initial configuration of this systems is $c_0 = (\emptyset, ea)$. Since the number of components is just one, the sequences of applicable programs are one element "sequences". The set of sequences of programs applicable in the initial configuration contains two elements, $App_{c_0} = \{(p_1), (p_2)\}$.

A possible computation of this system is the following:

$$(\emptyset, ea) \Rightarrow (a, ea) \Rightarrow (aa, ea) \Rightarrow (aaa, eb) \Rightarrow (aab, ba) \Rightarrow (bba, ba) \Rightarrow (bbb, ac)$$

where the first three computational steps read the multiset containing an a, the last three steps read a multiset containing a b, thus the accepted multiset sequence of this computation is $(a)(a)(a)(b)(b)(b)$.

It is not difficult to see that similarly to the one above, the computations which end in a final configuration (a configuration which does not contain the object a in the environment) accept the set of multiset sequences

$$A(\Pi) = \{(a)^n(b)^n \mid n \geq 1\}.$$

The set of multisets which can be read by Π is $in(\Pi) = \{a, b\}$ (where a and b denote the multisets containing one copy of the object a and b, respectively).

If we consider f_{perm} as the input mapping, we have

$$L(\Pi, f_{perm}) = \{a^n b^n \mid n \geq 1\}.$$

On the other hand, if we consider the mapping $f_1 \in \mathcal{F}_{\text{TRANS}}$ where $f_1 : in(\Pi) \to 2^{\Sigma^*}$ with $\Sigma = \{c, d, e, f\}$ and $f_1(a) = \{cd\}$, $f_1(b) = \{ef\}$, we get the language

$$L(\Pi, f_1) = \{(cd)^n(ef)^n \mid n \geq 1\}.$$

The computational capacity of genPCol automata was investigated in [19–21]. It was shown that with unrestricted programs systems of capacity *one* generate any recursively enumerable language, that is,

$$\mathcal{L}_X(\text{genPCol}, *(k)) = \mathcal{L}(\text{RE}),\ k \geq 1,\ X \in \{perm, TRANS\}.$$

A similar result holds for all-tape systems with capacity at least two.

$$\mathcal{L}_X(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE}) \text{ for } k \geq 2,\ X \in \{perm, TRANS\}.$$

.

3 P Colony Automata and the LL(k) Condition

Let $U \subset \Sigma^*$ be a finite set of strings over some alphabet Σ. Let $\mathrm{FIRST}_k(U)$ denote the set of length k prefixes of the elements of U for some $k \geq 1$, that is, let

$$\mathrm{FIRST}_k(U) = \{pref_k(u) \in \Sigma^* \mid u \in U\}$$

where $pref_k(u)$ denotes the string of the first k symbols of u if $|u| \geq k$, or $pref_k(u) = u$ otherwise.

Definition 1. Let $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$ be a genPCol automaton, let $f : in(\Pi) \to 2^{\Sigma^*}$ be a mapping as above, and let $c_0, c_1, \dots, c_s$ be a sequence of configurations with $c_i \Longrightarrow c_{i+1}$ for all $0 \leq i \leq s-1$.

We say that the P colony Π is LL(k) for some $k \geq 1$ with respect to the mapping f, if for any two distinct sets of programs applicable in configuration c_s, $P_{c_s}, P'_{c_s} \in App_{c_s}$ with $P_{c_s} \neq P'_{c_s}$, the next k symbols of the input string that is being read determines which of the two sequences are to be applied in the next computational step, that is, the following holds.

Consider two computations

$$c_s \xRightarrow{P_{c_s}} c_{s+1} \xRightarrow{P_{c_{s+1}}} \dots \xRightarrow{P_{c_{s+m}}} c_{s+m+1}, \text{ and } c_s \xRightarrow{P'_{c_s}} c'_{s+1} \xRightarrow{P'_{c_{s+1}}} \dots \xRightarrow{P'_{c_{s+m'}}} c'_{s+m'+1}$$

where $u_{c_s} = read(P_{c_s})$ and $u_{c_{s+i}} = read(P_{c_{s+i}})$ for $1 \leq i \leq m$, and similarly $u'_{c_s} = read(P'_{c_s})$ and $u'_{c_{s+i}} = read(P'_{c_{s+i}})$ for $1 \leq i \leq m'$, thus, the two sequences of input multisets are

$$u_{c_s} u_{c_{s+1}} \dots u_{c_{s+m}} \text{ and } u'_{c_s} u'_{c_{s+1}} \dots u'_{c_{s+m'}}.$$

Assume that these sequences are long enough to "consume" the next k symbols of the input string, that is, for w and w' with

$$w \in f(u_{c_s}) f(u_{c_{s+1}}) \dots f(u_{c_{s+m}}) \text{ and } w' \in f(u'_{c_s}) f(u'_{c_{s+1}}) \dots f(u'_{c_{s+m'}}),$$

either $|w| \geq k$ and $|w'| \geq k$, or if $|w| < k$ (or $|w'| < k$), then c_{s+m+1} (or $c_{s+m'+1}$) is a halting configuration.

Now, the P colony Π is LL(k), if for any two computations as above,

$$\mathrm{FIRST}_k(w) \cap \mathrm{FIRST}_k(w') = \emptyset.$$

The class of context-free LL(k) languages will be denoted by $\mathcal{L}$(CF,LL(k)) (see for example the monograph [1] for more details), while the languages characterized by genPCol automata satisfying the above defined condition, with input mapping of type f_{perm} or $f \in TRANS$, will be denoted by $\mathcal{L}_X$(genPCol,LL(k)), $X \in \{perm, TRANS\}$.

Let us illustrate the above definition with an example.

Example 2. Let $\Pi = (\{a, b, c, d, f, g, e\}, e, \emptyset, (ea, P_1), F)$ where

$$\begin{aligned} P_1 = \{ & \langle e \to b, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to e, b \overset{T}{\leftrightarrow} a\rangle, \langle e \to c, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to f, a \overset{T}{\leftrightarrow} e\rangle, \\ & \langle e \to d, c \overset{T}{\leftrightarrow} b\rangle, \langle b \to c, d \overset{T}{\leftrightarrow} e\rangle, \langle e \to g, f \overset{T}{\leftrightarrow} b\rangle, \langle b \to f, g \overset{T}{\leftrightarrow} e\rangle\} \text{ and} \\ F = \{ & (v, ce), (v, fe) \mid v \in V^*, b \notin v\}. \end{aligned}$$

The language characterized by Π is

$$L(\Pi, f_{perm}) = \{a\} \cup \{(ab)^n a(cd)^n \mid n \geq 1\} \cup \{(ab)^n a(fg)^n \mid n \geq 1\}.$$

To see this, consider the possible computations of Π. The initial configuration is $(\emptyset, ea)$ and there are three possible configurations that can be reached, namely (we denote by $\Rightarrow_u$ a configuration change during which the multiset of symbols u was read by the automaton)

1. $(\emptyset, ea) \Rightarrow_a (a, ce)$,
2. $(\emptyset, ea) \Rightarrow_a (a, fe)$,
3. $(\emptyset, ea) \Rightarrow_a (a, be)$.

The first two cases are non-accepting states, but the derivations cannot be continued, so let us consider the third one.

$$(a, be) \Rightarrow_b (b, ea) \Rightarrow_a (ba, be) \Rightarrow_b (bb, ea) \Rightarrow_a \ldots \Rightarrow_b (b^i, ea).$$

At this point, the computation can follow two different paths again, either

$$(b^i, ae) \Rightarrow_a (b^i a, ec) \Rightarrow_c (b^{i-1} ac, db) \Rightarrow_d (b^{i-1} acd, ce) \Rightarrow_c \ldots \Rightarrow_d (ac^i d^i, ce),$$

or

$$(b^i, ae) \Rightarrow_a (b^i a, ef) \Rightarrow_f (b^{i-1} af, gb) \Rightarrow_g (b^{i-1} afg, fe) \Rightarrow_f \ldots \Rightarrow_g (af^i g^i, fe).$$

In the first phase of the computation, the system produces copies of b and sends them to the environment, then in the second phase these copies of b are exchanged to copies of cd or copies of fg. The system can reach an accepting state when all the copies of b are used, that is, when an equal number of copies of ab and either of cd or of fg were produced.

Note that the system satisfies the LL(1) property, the symbol that has to be read, in order to accept a desired input word, determines the set of programs that has to be used in the next computational step.

As a consequence of the above example, we can state the following.

Theorem 1. *There are context-free languages in $\mathcal{L}_X$(genPCol,LL(1)), $X \in \{perm, TRANS\}$, which are not in $\mathcal{L}$(CF,LL(k)) for any $k \geq 1$.*

Proof. The language $L(\Pi, f_{perm}) \in \mathcal{L}_{perm}$(genPCol,LL(1)) from Example 2 is not in $\mathcal{L}$(CF,LL(k)) for any $k \geq 1$. If we consider the mapping $f_1 \in TRANS$, $f_1 : \{a, b, c, d, f, g\} \to \{a, b, c, d, f, g\}$ with $f_1(x) = x$ for all $x \in \{a, b, c, d, f, g\}$, then $L(\Pi, f_1) = L(\Pi, f_{perm})$, thus, $\mathcal{L}_{TRANS}$(genPCol,LL(1)) also contains the non-LL(k) context-free language.

4 Conclusions

P systems and their variants are able to describe powerful language classes, thus their applicability in the theory of parsing or analyzing syntactic structures are of particular interest, see, for example [15,16]. For example, in [15], so-called active P automata (P automata with dynamically changing membrane structure) were used for parsing, utilizing the dynamically changing membrane structure of the P automaton for analyzing the string. In addition to studying the suitability of P system variants for parsing different types of languages, developing well-known notions like LL(k) property for these models are of interest as well, since the results demonstrate the boundaries of the original concepts. In this paper we have started investigations in this direction, namely we studied the possibility of deterministically parsing languages characterized by P colony automata. We provided the definition of an LL(k)-like property for (generalized) P colony automata, and showed that languages which are not LL(k) in the "original" context-free sense for any $k \geq 1$ can be characterized by LL(1) P colony automata with different types of input mappings. The properties of these language classes for different values of k and different types of input mappings are open to further investigations. Our investigations confirmed that concepts, questions related to parsing are of interest in P systems theory.

Acknowledgments. The work of E. Csuhaj-Varjú was supported in part by the National Research, Development and Innovation Office of Hungary, NKFIH, grant no. K 120558. The work of K. Kántor and Gy. Vaszil was supported in part by the National Research, Development and Innovation Office of Hungary, NKFIH, grant no. K 120558 and also by the construction EFOP-3.6.3-VEKOP-16-2017-00002, a project financed by the European Union, co-financed by the European Social Fund.

References

1. Aho, A.V., Ulmann, J.D.: The Theory of Parsing, Translation, and Compiling, vol. 1. Prentice-Hall, Englewood Cliffs (1973)
2. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: Towards P colonies processing strings. In: Macías Ramos, L.F., Martínez Del Amor, M.A., Păun, G., Pérez Hurtado, I., Riscos Nuñez, A., Valencia Cabrera, L. (eds.) Twelfth Brainstorming Week on Membrane Computing, pp. 103–118. Fénix Editora (2014)
3. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E., Vaszil, G.: PCol automata: recognizing strings with P colonies. In: Martínez Del Amor, M.A., Păun, G., Pérez Hurtado, I., Riscos Nuñez, A. (eds.) Eighth Brainstorming Week on Membrane Computing, Sevilla, 1–5 February 2010, pp. 65–76. Fénix Editora (2010)
4. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: P colonies processing strings. Fundam. Inform. **134**(1–2), 51–65 (2014)
5. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: A class of restricted P colonies with string environment. Nat. Comput. **15**(4), 541–549 (2016)
6. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E., Sosík, P.: P colonies. Bull. Int. Membr. Comput. Soc. **1**(2), 119–156 (2016)

7. Cienciala, L., Ciencialová, L., Kelemenová, A.: On the number of agents in P colonies. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 193–208. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77312-2_12
8. Cienciala, L., Ciencialová, L., Kelemenová, A.: Homogeneous P colonies. Comput. Inform. **27**(3), 481–496 (2008)
9. Ciencialová, L., Cienciala, L.: Variation on the theme: P colonies. In: Kolăr, D., Meduna, A. (eds.) Proceedings of the 1st International Workshop on Formal Models, Ostrava, pp. 27–34 (2006)
10. Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., Vaszil, G.: Variants of P colonies with very simple cell structure. Int. J. Comput., Commun. Control. **4**(3), 224–233 (2009)
11. Csuhaj-Varjú, E., Oswald, M., Vaszil, G.: P automata. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., Oxford (2010)
12. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A.: Computing with cells in environment: P colonies. Mult.-Valued Log. Soft Comput. **12**(3–4), 201–215 (2006)
13. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: P colonies with a bounded number of cells and programs. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 352–366. Springer, Heidelberg (2006). https://doi.org/10.1007/11963516_22
14. Csuhaj-Varjú, E., Vaszil, G.: P automata or purely communicating accepting P systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 219–233. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36490-0_14
15. Bel-Enguix, G., Gramatovici, R.: Parsing with active P automata. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 31–42. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24619-0_3
16. Bel Enguix, G., Nagy, B.: Modeling syntactic complexity with P systems: a preview. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 54–66. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08123-6_5
17. Freund, R., Oswald, M.: P colonies working in the maximally parallel and in the sequential mode. In: Zaharie, D., et al. (eds.) Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25–29 September 2005, Timisoara, Romania, pp. 419–426. IEEE Computer Society (2005)
18. Freund, R., Oswald, M.: P colonies and prescribed teams. Int. J. Comput. Math. **83**(7), 569–592 (2006)
19. Kántor, K., Vaszil, G.: Generalized P colony automata. J. Autom., Lang. Comb. **19**(1–4), 145–156 (2014)
20. Kántor, K., Vaszil, G.: Generalized P colony automata and their relation to P automata. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2017. LNCS, vol. 10725, pp. 167–182. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73359-3_11
21. Kántor, K., Vaszil, G.: On the classes of languages characterized by generalized P colony automata. Theor. Comput. Sci. **724**, 35–44 (2018)
22. Kelemenová, A.: P colonies. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 584–593. Oxford University Press, Inc., Oxford (2010)
23. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Gener. Comput. **20**(3), 295–306 (2002)