P Colony Automata with LL(k)-like Conditions *

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, 1117 Budapest, Hungary
csuhaj@inf.elte.hu
[2] Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
{kantor.kristof, vaszil.gyorgy}@inf.unideb.hu

Summary. We investigate the possibility of the deterministic parsing (that is, parsing without backtracking) of languages characterized by (generalized) P colony automata. We define a class of P colony automata 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, where multisets of objects are used to describe the contents of the cells and environment and then are processed by the cells in the corresponding colony using rules which enable the evolution of the objects present in the cells and 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 system) and they can process a restricted amount of information. The way the information processing is done is really simple: The rules are either of the form $a \to b$ (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 rule set is called a program, it consists of exactly the same number of rules as the capacity of the system. When a program is executed, the k (the capacity of the system) rules that it contains are applied to

* Supported in part by project no. K 120558, implemented by the National Research, Development and Innovation Fund of Hungary, financed under the K_16 funding scheme. Also supported 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.

the k objects simultaneously. During a computational step, every colony member cell execute one of their programs in parallel. A computation ends when the system reaches one of its the final configurations (usually given as the set of halting configurations, that is, those situations when no programs can be applied by any of the cells).

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 [5, 6, 4, 3, 8, 9, 11, 12].

P colony automata were introduced in [2]. They are called automata, because they accept string languages by assuming an initial input tape and 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 [13] and studied further in [15, 14]. 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 [10] (see also [7]) are P systems using symport and antiport rules (see [16]), 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 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 [13], some basic variants of the model were introduced and studied from the point of view of their computational power. In [15, 14] 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 of the systems. In the present 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 one symbol lookahead, as in the case of context-free LL(k) languages, and present an initial result showing that using P colony automta 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. We denote the number of occurrences of a symbol $a \in V$ in w 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 $a \to b$, 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 symbols) 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 program $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that r is one of the rules of the program, that is, $r = r_j$ for some $j,\ 1 \leq j \leq k$.

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

Let us also denote by $export(p)$ and $import(p)$ the multiset of objects that are sent out to the environment and brought inside the cell when applying the program p, respectively, that is, $export(p) = \bigcup_{r \in p} left(r)$, $import(p) = \bigcup_{r \in p} right(r)$ for all communication rules r of the program p. Moreover, by $create(p)$ we denote the multiset of symbols produced by the rewriting rules of program p, thus, $create(p) = \bigcup_{r \in p} right(p)$ for the rewriting rules r of p.

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of a genPCol automaton Π, 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 es which are always present). The *set 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.

Let us denote by App_c be the set of all applicable sets of programs in the configuration $c = (u_E, u_1, \ldots, u_n)$, that is,

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

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of the genPCol automaton. By applying a set 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 $c \stackrel{P_c}{\Longrightarrow} c'$, if the following properties hold:

- If $(p_1, \ldots, p_n) = P_c$ 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 es).

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, a set of applicable programs in a configuration c, let us denote by $read(P_c)$ 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 is infinite (because the multiset corresponding to the contents of the environment is not necessarily finite), the set $in(\Pi)$ is 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 sets of applicable programs in any configuration, that is, the set App_c.

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

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

$$A(\Pi) = \{u_1u_2 \ldots u_s \mid u_i \in in(\Pi),\ 1 \leq i \leq s,\ \text{and there is a configuration sequence } c_0, \ldots, c_s,\ \text{with } c_0 = (w_E, w_1, \ldots, w_n),\ c_s \in F,\ \text{and } c_i \overset{P_{c_i}}{\Longrightarrow} c_{i+1} \text{ with } u_{i+1} = read(P_{c_i}) \text{ for all } 0 \leq i \leq s-1\}.$$

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

$$L(\Pi, f) = \{f(u_1)f(u_2) \ldots f(u_s) \in \Sigma^* \mid\ u_1u_2 \ldots u_s \in A(\Pi)\}.$$

Let us denote the class of languages accepted by generalized PCol automata with capacity l and with mappings from the class $\mathcal{F}$

- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(l))$ when all the communication rules are tape rules,
- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(l))$ when all the programs must have at least one tape rule, and
- by $\mathcal{L}(\text{genPCol}, \mathcal{F}, *(l))$ 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

$$f(v) = \{a_1a_2 \ldots a_s \mid |v| = s, a_1a_2 \ldots a_s \text{ is a permutation of the elements of } v\}.$$

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

Now we recall an example from [14] to demonstrate the above defined notions.

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

$$P = \{\langle e \to a, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to b, a \overset{T}{\leftrightarrow} e\rangle, \langle e \to b, b \overset{T}{\leftrightarrow} a\rangle, \langle e \to c, b \overset{T}{\leftrightarrow} a\rangle, \langle a \to b, b \overset{T}{\leftrightarrow} a\rangle, \langle a \to c, b \overset{T}{\leftrightarrow} a\rangle\}$$

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

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 $f_1 : in(\Pi) \rightarrow 2^{\Sigma^*}$ 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 [13, 15, 14]. 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 us denote by $\text{FIRST}_k(U)$ for some $k \geq 1$, the set of length k prefixes of the elements of U, that is, let

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

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 $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton, let $f : in(\Pi) \to 2^{\Sigma^*}$ be a mapping as above, and let $c_0, c_1, \ldots, c_s$ 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, $P_1, P_2 \in Acc_{c_s}$ with $P_1 \neq P_2$, if $u_1 = read(P_1)$ and $u_2 = read(P_2)$, then $\mathrm{FIRST}_k(f(u_1)) \cap \mathrm{FIRST}_k(f(u_2)) = \emptyset$.

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 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

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

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 bs and sends them to the environment, then in the second phase these bs are exchanged to cds or fgs. The system can reach an accepting state when all the bs are used, that is, when an equal number of abs and either cds or fgs 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 $f_1 \in TRANS$, $f_1 : \{a, b, c, d, f, g\} \rightarrow \{a, b, c, d, f, g\}$ 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

We have investigated the possibility of deterministically parsing languages characterized by P colony automata. We have given the definition of an LL(k)-like property for (generalized) P colony automata, and shown 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 ks and different types of input mappings are open to further investigations.

References

1. Aho, A.V., Ulmann, J.D.: The Theory of Parsing, Translation, and Compiling, vol. 1. Prentice-Hall, Englewood Cliffs, N.J. (1973)
2. 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 Núñez, A. (eds.) Eighth Brainstorming Week on Membrane Computing, Sevilla, February 1-5, 2010, pp. 65–76. Fénix Editora (2010)
3. 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.) Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 4860, pp. 193–208. Springer (2007)
4. Cienciala, L., Ciencialová, L., Kelemenová, A.: Homogeneous P colonies. Computing and Informatics 27(3+), 481–496 (2008)

5. Ciencialová, L., Cienciala, L.: Variation on the theme: P colonies. In: Kolář, D., Meduna, A. (eds.) Proc. 1st Intern. Workshop on Formal Models. pp. 27–34. Ostrava (2006)
6. Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., Vaszil, G.: Variants of P colonies with very simple cell structure. International Journal of Computers, Communication and Control 4(3), 224–233 (2009)
7. 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. (2010)
8. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A.: Computing with cells in environment: P colonies. Multiple-Valued Logic and Soft Computing 12(3-4), 201–215 (2006)
9. 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.) Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers. Lecture Notes in Computer Science, vol. 4361, pp. 352–366. Springer (2006)
10. Csuhaj-Varjú, E., Vaszil, G.: P automata or purely communicating accepting P systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2597, pp. 219–233. Springer (2002)
11. Freund, R., Oswald, M.: P colonies working in the maximally parallel and in the sequential mode. In: Zaharie, D., Petcu, D., Negru, V., Jebelean, T., Ciobanu, G., Cicortas, A., Abraham, A., Paprzycki, M. (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)
12. Freund, R., Oswald, M.: P colonies and prescribed teams. Int. J. Comput. Math. 83(7), 569–592 (2006)
13. Kántor, K., Vaszil, G.: Generalized P colony automata. Journal of Automata, Languages and Combinatorics 19(1-4), 145–156 (2014)
14. Kántor, K., Vaszil, G.: Generalized P colony automata and their relation to P automata. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing - 18th International Conference, CMC 2017, Bradford, UK, July 25-28, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10725, pp. 167–182. Springer (2017)
15. Kántor, K., Vaszil, G.: On the classes of languages characterized by generalized P colony automata. Theor. Comput. Sci. 724, 35–44 (2018)
16. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Comput. 20(3), 295–306 (2002)