The Family of Languages Generated by Non-Cooperative Membrane Systems

Artiom Alhazov[1,2], Constantin Ciubotaru[1], Yurii Rogozhin[1], Sergiu Ivanov[1,3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {artiom,chebotar,rogozhin,sivanov}@math.md
[2] IEC, Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima 739-8527 Japan
[3] Technical University of Moldova, Faculty of Computers,
Informatics and Microelectronics,
Ştefan cel Mare 168, Chişinău MD-2004 Moldova

Abstract. The aim of this paper is to study the family of languages generated by the transitional membrane systems without cooperation and without additional ingredients. The fundamental nature of these basic systems makes it possible to also define the corresponding family of languages in terms of derivation trees of context-free grammars. We also compare this family to the well-known language families and discuss its properties. An example of a language is given which is considerably more "difficult" than those in the established lower bounds.

1 Introduction

Membrane computing is a theoretical framework of parallel distributed multiset processing. It has been introduced by Gheorghe Păun in 1998, and has been an active research area; see [13] for the comprehensive bibliography and [8],[10] for a systematic survey. Membrane systems are also called P systems.

The configurations of membrane systems (with symbol objects) consist of multisets over a finite alphabet, distributed across a tree structure. Therefore, even such a relatively simple structure as a word (i.e., a sequence of symbols) is not explicitly present in the system. To speak of languages as sets of words, one first needs to represent them in membrane systems, and there are a few ways to do it.

- Represent words by string objects. Rather many papers take this approach, see Chapter 7 of [10], but only few consider parallel operations on words. Moreover, a tuple of sets or multisets of words is already a quite complicated structure. The third drawback is that it is very difficult to define an elegant way of interactions between strings. Polarizations and splicing are examples of that; however, these are difficult to use in applications. In this paper we focus on symbol objects.

– Represent a word by a single symbol object, or by a few objects of the form (letter,position) as in, e.g., [2]. This only works for words of bounded length, i.e., one can speak about at most finite languages.
– Represent positions of the letters in a word by nested membranes. The corresponding letters can be then encoded by objects in the associated regions, membrane types or membrane labels. Working with such a representation, even implementing a rule $a \to bc$ requires sophisticated types of rules, like creating a membrane around existing membrane, as defined in [3].
– Consider letters as digits and then view words as numbers, or use some other encoding of words into numbers or multisets. Clearly, the concept of words ceases to be direct with such encoding. Moreover, implementing basic word operations in this way requires a lot of number processing, not to speak of parallel word operations.
– Work in accepting mode, see, e.g., [4]. It is necessary to remark that in this article we are mainly speaking of non-cooperative P systems, and working in accepting mode without cooperation yields rather trivial subregular results. More exactly, such systems would accept either the empty language or only the empty word, or all the words over some subalphabet.
– Consider traces of objects across membranes. This is an unusual approach in the sense that the result is not obtained from the final configuration, but rather from the behavior of a specific object during the computation. This makes it necessary to introduce an observer, complicating the model.
– Do all the processing by multisets, and regard the order of sending the objects in the environment as their order in the output word. In case of ejecting multiple symbols in the same step, the output word is formed from any of their permutations. One can say that this approach also needs an implicit observer, but at least this observer only inspects the environment and it is, in some sense, the simplest possible one. This paper is devoted to this concept.

Informally, the family of languages we are interested in is the family generated by systems with parallel applications of non-cooperative rules that rewrite symbol objects and/or send them between the regions. This model has been introduced already in 2000, [9]. Surprisingly, this language family did not yet receive enough attention of researchers. Almost all known characterizations and even bounds for generative power of different variants of membrane systems with various ingredients and different descriptional complexity bounds are expressed in terms of REG, MAT, $ET0L$ and RE, their length sets and Parikh sets (and much less often in terms of FIN or other subregular families, or CF or CS, or those accepted by log-tape bounded Turing machines, [5], [7]). The membrane systems language family presents interest since we show it lies between regular and context-sensitive families, being incomparable with well-studied intermediate ones. As we show in the paper, the nature of $LOP_*(ncoo, tar)$ is quite fundamental, and in the same time it is not characterized in terms of well-studied families. This is why we refer to it here as *the membrane systems language family.*

This paper is based on the ideas and initial work described in [1]. We also give an example of a language which is considerably more “difficult” than the currently established lower bounds. The word "difficult" informally refers to two kinds of non-context-freeness needed to describe the language. The internal one can be captured by permutations, while the external one can be captured by an intersection of linear languages.

2 Definitions

2.1 Formal language preliminaries

Consider a finite set V. The set of all words over V is denoted by V^*, the concatenation operation is denoted by $\bullet$ (which is written only when necessary) and the empty word is denoted by λ. Any set $L \subseteq V^*$ is called a language. For a word $w \in V^*$ and a symbol $a \in V$, the number of occurrences of a in w is written as $|w|_a$. The permutations of a word $w \in V^*$ are $\texttt{Perm}(w) = \{x \in V^* \mid |x|_a = |w|_a \forall a \in V\}$. We denote the set of all permutations of the words in L by $\texttt{Perm}(L)$, and we extend this notation to families of languages. We use FIN, REG, LIN, CF, MAT, CS, RE to denote finite, regular, linear, context-free, matrix without appearance checking and with erasing rules, context-sensitive and recursively enumerable families of languages, respectively. The family of languages generated by extended (tabled) interactionless L systems is denoted by $E(T)0L$. For more formal language preliminaries, we refer the reader to [11].

Throughout this paper we use string notation to denote the multisets. When speaking about membrane systems, keep in mind that the order in which symbols are written is irrelevant, unless we speak about the symbols sent to the environment. In particular, speaking about the contents of some membrane, when we write $a_1^{n_1} \cdots a_m^{n_m}$ (or any permutation of it), we mean a multiset consisting of n_i instances of symbol a_i, $1 \leq i \leq m$.

2.2 Transitional P systems

A membrane system is defined by a construct

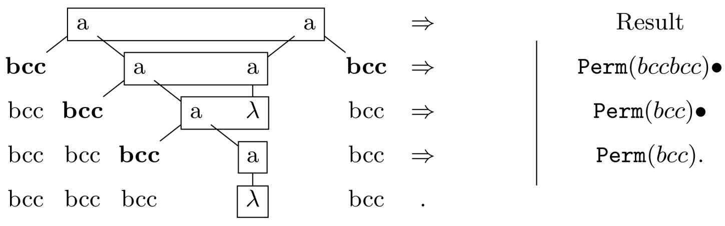
$\Pi = (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, i_0)$, where

O	is a finite set of objects,
μ	is a hierarchical structure of membranes, bijectively labeled by $1, \cdots, m$, the interior of each membrane defines a region; the environment is referred to as region 0,
w_i	is the initial multiset in region $i, 1 \leq i \leq m$,
R_i	is the set of rules of region $i, 1 \leq i \leq m$,
i_0	is the output region; when languages are considered, $i_0 = 0$ is assumed.

The rules of a membrane systems have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$ are written as a subscript, and target *here* is typically omitted. In case of non-cooperative rules, $u \in O$.

The rules are applied in maximally parallel way: no further rule should be applicable to the idle objects. In case of non-cooperative systems, the concept of maximal parallelism is the same as in L systems: all objects evolve by the associated rules in the corresponding regions (except objects a in regions i such that R_i does not contain any rule $a \to u$, but these objects do not contribute to the result). The choice of rules is non-deterministic.

A configuration of a P system is a construct which contains the information about the hierarchical structure of membranes as well as the contents of every membrane at a definite moment of time. The process of applying all rules which are applicable in the current configuration and thus obtaining a new configuration is called a transition. A sequence of transitions is called a computation. The computation halts when such a configuration is reached that no rules are applicable. The result of a (halting) computation is the *sequence* of objects sent to the environment (all the permutations of the symbols sent out in the same time are considered). The language $L(\Pi)$ generated by a P system Π is the union of the results of all computations. The family of languages generated by non-cooperative transitional P systems with at most m membranes is denoted by $LOP_m(ncoo, tar)$. If the number of membranes is not bounded, m is replaced by $*$ or omitted. If the target indications of the form in_j are not used, tar is replaced by out.



a a ⇒ Result
bcc a a **bcc** ⇒ Perm(bccbcc)•
bcc **bcc** a λ bcc ⇒ Perm(bcc)•
bcc bcc **bcc** a bcc ⇒ Perm(bcc).
bcc bcc bcc λ bcc .

Fig. 1. An example of a computation of a P system from Example 1. The lines are only used to hint how the rules are applied.

Example 1. To illustrate the concept of generating languages, consider the following P system:

$$\Pi = (\{a, b, c\}, [_1\ ]_1, a^2, \{a \to \lambda, a \to a\ b_{out}c_{out}^2\}, 0).$$

Each of the two symbols a has a non-deterministic choice whether to be erased or to reproduce itself while sending a copy of b and two copies of c into the environment. Therefore, the contents of region 1 can remain a^2 for an arbitrary number $m \geq 0$ of steps, and after that at least one copy of a is erased. The other copy of a can reproduce itself for another $n \geq 0$ steps before being erased. Each of the first m steps, two copies of b and four copies of c are sent out, while in each of the next n steps, only one copy of b and two copies of c are ejected. Therefore, $L(\Pi) = (\texttt{Perm}(bccbcc))^*(\texttt{Perm}(bcc))^*$.

3 Context-free grammars and time-yield

Consider a non-terminal A in a grammar $G = (N, T, S, P)$. We denote by G_A the grammar (N, T, A, P) obtained by considering A as axiom in G.

A derivation tree in a context-free grammar is an ordered rooted tree with leaves labeled by terminals and all other nodes labeled by non-terminals. Rules of the form $A \to \lambda$ cause a problem, which can be solved by allowing to also label leaves by λ, or by transformation of the corresponding grammar. Note: throughout this paper by derivation trees we only mean finite ones. Consider a derivation tree τ. The following notion describes the sequence of terminal symbols at a particular depth of a derivation tree:

The n-th level yield $\texttt{yield}_n$ of τ can be defined as follows:

We define $\texttt{yield}_0(\tau) = a$ if τ has a single node labeled by $a \in T$, and $\texttt{yield}_0(\tau) = \lambda$ otherwise.
Let k be the number of children nodes of the root of τ, and $\tau_1, \cdots, \tau_k$ be the subtrees of τ with these children as roots. We define $\texttt{yield}_{n+1}(\tau) = \texttt{yield}_n(\tau_1) \bullet \texttt{yield}_n(\tau_2) \bullet \cdots \bullet \texttt{yield}_n(\tau_k)$.

We now define the time yield L_t of a context-free grammar derivation tree τ, as the usual yield except the order of terminals is vertical from root instead of left-to-right, and the order of terminals at the same distance from root is arbitrary. We use $\prod$ to denote concatenation in the following formal definition:

$$L_t(\tau) = \prod_{n=0}^{\texttt{height}(\tau)} (\texttt{Perm}(\texttt{yield}_n(\tau))).$$

The time yield $L_t(G)$ of a grammar G is the union of time yields of all its derivation trees. The corresponding family of languages is

$$L_t(CF) = \{L_t(G) \mid G \text{ is a context-free grammar}\}.$$

Example 2. Consider a grammar $G_1 = (\{S, A, B, C\}, \{a, b, c\}, S, P)$, where

$$P = \{S \to SABC, S \to ABC, A \to A, B \to B, C \to C, A \to a, B \to b, C \to c\}.$$

We now show that $L_t(G_1) = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c > 0\} = L$. Indeed, all derivations of A are of the form $A \Rightarrow^* A \Rightarrow a$. Likewise, symbols B, C are also trivially rewritten an arbitrary number of times and then changes into a corresponding terminal. Hence, $L_t({G_1}_A) = \{a\}$, $L_t({G_1}_B) = \{b\}$, $L_t({G_1}_C) = \{c\}$. For inclusion $L_t(G) \subseteq L$ it suffices to note that S always generates the same number of symbols A, B, C.

The converse inclusion follows from the following simulation: given a word $w \in L$, generate $|w|/3$ copies of A, B, C, and then apply their trivial rewriting in such way that the timing when the terminal symbols appear corresponds to their order in w.

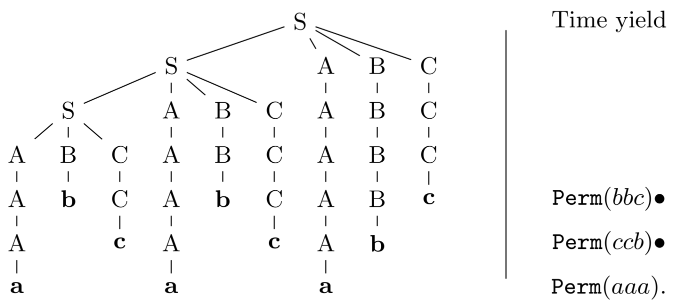


Fig. 2. An example of a derivation of the grammar from Example 2.

Corollary 1. *$L_t(CF) \not\subseteq CF$.*

4 The membrane family via the derivation trees of context-free grammars

We first show that for every membrane system without cooperation, there is a system from the same class with one membrane, generating the same language.

Lemma 1. *$LOP(ncoo, tar) = LOP_1(ncoo, out)$.*

Proof. Consider an arbitrary transitional membrane system Π (without cooperation and without additional ingredients). The known technique of flattening the structure (this is "folklore" in membrane computing; see, e.g., [12], [6]) consists of transforming Π in the following way. Object a in region associated to membrane i is transformed into object (a, i) in the region associated to the single membrane. The alphabet, initial configuration and rules are transformed accordingly. Clearly, the configurations of the original system and the new system are bisimilar, and the output in the environment is the same. $\square$

Theorem 1. *$L_t(CF) = LOP(ncoo, tar)$.*

Proof. By Lemma 1, the statement is equivalent to $L_t(CF) = LOP_1(ncoo, out)$. Consider a P system $\Pi = (O, [_1\ ]_1, w, R, 0)$. We construct a context-free grammar $G = (O' \cup \{S\}, O, S, P \cup \{S \to w\})$, where S is a new symbol, $'$ is a morphism from O into new symbols and

$$\begin{aligned} P = \{a' \to u'v \mid (a \to u\ v_{out}) \in R,\ a \in O,\ u, v \in O^*\} \\ \cup\ \{a' \to \lambda \mid \neg\exists(a \to u\ v_{out}) \in R\}. \end{aligned}$$

Here v_{out} are those symbols on the right-hand side of the rule in R which are sent out into the environment, and u are the remaining right-hand side symbols.

The computations of Π are identical to parallel derivations in G, except the following:

- Unlike G, Π does not keep track of the left-to-right order of symbols. This does not otherwise influence the derivation (since rules are context-free) or the result (since the order of non-terminals produced in the same step is arbitrary, and the timing is preserved).
- The initial configuration of Π is produced from the axiom of G in one additional step.
- The objects of Π that cannot evolve are erased in G, since they do not contribute to the result.

It follows that $L_t(CF) \supseteq LOP(ncoo, tar)$. To prove the converse inclusion, consider an arbitrary context-free grammar $G = (N, T, S, P)$. We construct a P system $\Pi = (N \cup T, [_1 \ ]_1, S, R, 0)$, where $R = \{a \to h(u) \mid (a \to u) \in R\}$, where h is a morphism defined by $h(a) = a$, $a \in N$ and $h(a) = a_{out}$, $a \in T$. The computations in Π correspond to parallel derivations in G, and the order of producing terminal symbols in G corresponds to the order of sending them to the environment by Π, hence the theorem statement holds. □

We now present a few normal forms for the context-free grammars in the context of the time yield. Note that these normal forms incorporate a number of similarities with both L systems and standard CF grammars, because level-by-level derivation in context-free grammars corresponds to the evolution in L systems. However, it is not possible to simply take existing normal forms, because the result must be preserved, and the result is defined in a different way.

Lemma 2. *(First normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

- *the axiom does not appear in the right-hand side of any rule, and*
- *if the left side is not the axiom, then the right-hand side is not empty.*

Proof. The technique is essentially the same as removing λ-productions in classical theory of context-free grammars. Let $G = (N, T, S, P)$. First, introduce the new axiom S' and add a rule $S' \to S$. Compute the set $E \subseteq N$ of non-terminals that can derive λ. This set has the following properties (read "$\longrightarrow$" as "then"):

$$(A \to \lambda) \longrightarrow (A \in E),$$
$$(A \to A_1 \cdots A_k),\ (A_1, \cdots, A_k \in E) \longrightarrow (A \in E).$$

Then replace productions $A \to u$ by $A \to h(u)$, where $h(a) = \{a, \lambda\}$ if $a \in E$ and $h(a) = a$ if $a \in N \cup T \setminus E$. Finally, remove λ-productions for all non-terminals except the axiom. Note that this transformation preserves not only the generated terminals, but also the order in which they are generated. □

The First normal form shows that erasing can be limited to the axiom.

Lemma 3. *(Binary normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

- *the First normal form holds,*

– *the right-hand side of any production is at most 2.*

Proof. The only concern in splitting the longer productions of $G = (N, T, S, P)$ in shorter ones is to preserve the order in which non-terminals are produced. The number

$$n = \lceil \log_2 \left(\max_{(A \to u) \in P} |u| \right) \rceil$$

is the number of steps sufficient to implement all productions of G by at most binary productions. Each production $p : A \to A_1 \cdots A_k$, $k \leq 2^n$, is replaced by

$$\begin{aligned} A &\to p_{0,0}, \\ p_{i,j} &\to p_{i+1,2j} p_{i+1,2j+1} \text{ for } 0 \leq i \leq n-1, 0 \leq j \leq 2^i - 1, \\ p_{n,i-1} &\to A_i \text{ for } 1 \leq i \leq k, \\ p_{n,i} &\to \lambda \text{ for } k \leq i \leq 2^n. \end{aligned}$$

these productions implement a full binary tree of depth n, rooted in A with new symbols in intermediate nodes, and leaves labeled $A_1, \cdots, A_k$, all remaining leaves labeled λ (the first and last chain productions are given for the simplicity of the presentation). It only remains to convert the grammar obtained to the First normal form. Indeed, the derivations in the obtained grammar correspond to the derivation of the original one, with the slowdown factor of $n + 2$, and the order of producing terminal symbols is preserved. Obviously, converting into the First normal form does not increase the right-hand side size of productions. $\square$

The Binary normal form shows that productions with right-hand side longer than two are not necessary.

Lemma 4. *(Third normal form) For a context-free grammar G there exists a context-free grammar G' such that $L_t(G) = L_t(G')$ and in G':*

– *the Binary normal form holds,*
– *$G' = (N, T, S, P')$ and every $A \in N$ is reachable,*
– *either $G' = (\{S\}, T, S, \{S \to S\})$, or $G' = (N, T, S, P')$ and for every $A \in N$, $L_t(G'_A) \neq \emptyset$.*

Proof. Consider a context-free grammar in the Binary normal form. First, compute the set $D \subseteq N$ of productive non-terminals as closure of

$$\begin{aligned} &(A \to u),\ (u \in T^*) \longrightarrow (A \in D) \\ &(A \to A_1 \cdots A_k),\ (A_1, \cdots, A_k \in D) \longrightarrow (A \in D). \end{aligned}$$

Remove all non-terminals that are not productive from N, and all productions containing them. If the axiom was also removed, then $L_t(G) = \emptyset$, hence we can take $G' = (\{S\}, T, S, \{S \to S\})$. Otherwise, compute the set $R \subseteq N$ of reachable non-terminals as closure of

$$\begin{aligned} &(S \in R), \\ &(A \in R),\ (A \to A_1 \cdots A_k) \longrightarrow (A_1, \cdots, A_k \in R). \end{aligned}$$

Remove all non-terminals not reachable from N, and all productions containing them. Note that all transformations preserve the generated terminals and the order in which they are produced, as well as the Binary normal form. □

The Third normal form shows that never ending derivations are only needed to generate the empty language.

5 Comparison with known families

Theorem 2. *[9] $LOP(ncoo, tar) \supseteq REG$.*

Proof. Consider an arbitrary regular language. Then there exists a complete finite automaton $M = (Q, \Sigma, q_0, F, \delta)$ accepting it. We construct a context-free grammar $G = (Q, \Sigma, q_0, P)$, where $P = \delta \cup \{q \to \lambda \mid q \in F\}$. The order of symbols accepted by M corresponds to the order of symbols generated by G, and the derivation can only finish when the final state is reached. Hence, $L_t(G) = L(M)$, and the theorem statement follows. □

Theorem 3. *$LOP(ncoo, tar) \subseteq CS$.*

Proof. Consider a context-free grammar $G = (N, T, S, P)$ in the First normal form. We construct a grammar $G' = (N \cup \{\#_1, L, R, F, \#_2\}, T, S', P')$, where

$$\begin{aligned} P' = \{&S' \to \#_1 LS\#_2, L\#_2 \to R\#_2, \#_1 R \to \#_1 L, \#_1 R \to F, F\#_2 \to \lambda\} \\ &\cup \{LA \to uL \mid (A \to u) \in P\} \cup \{La \to aL, Fa \to aF \mid a \in T\} \\ &\cup \{aR \to Ra \mid a \in N \cup T\}. \end{aligned}$$

The symbols $\#_1, \#_2$ mark the edges, the role of symbol L is to apply productions P to all non-terminals, left-to-right, while skipping the terminals. While reaching the end marker, symbol L changes into R and returns to the beginning marker, where it either changes back to L to iterate the process, or to F to check whether the derivation is finished.

Hence, $L(G') = L_t(G)$. Note that the length of sentential forms in any derivation (of some word with n symbols in G') is at most $n + 3$, because the only shortening productions are the ones removing $\#_1, \#_2$ and F, and each should be applied just once. Therefore, $L_t(G) \in CS$, and the theorem is proved. □

We now proceed to showing that the membrane systems language family does not contain the family of linear languages. To show this, we first define the notions of unbounded yield and unbounded time of a non-terminal.

Definition 1. *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has an unbounded yield if $L_t(G_A)$ is an infinite language, i.e., there is no upper bound on the length of words generated from A.*

It is easy to see that $L_t(G_A)$ is infinite if and only if $L(G_A)$ is infinite; decidability of this property is well-known from the theory of context-free grammars.

Definition 2. *Consider a grammar $G = (N, T, S, P)$. We say that $A \in N$ has unbounded time if the set of all derivation trees (for terminated derivations) in G_A is infinite, i.e., there is no upper bound on the number of parallel steps of terminated derivations in G_A.*

It is easy to see that A has unbounded time if $L(G_A) \neq \emptyset$ and $A \Rightarrow^+ A$, so decidability of this property is well-known from the theory of context-free grammars.

Lemma 5. *Let $G = (N, T, P, S)$ be a context-free grammar in the Third normal form. If for every rule $(A \to BC) \in P$, symbol B does not have unbounded time, than $L_t(G) \in REG$.*

Proof. Assume the premise of the lemma holds. Let F be the set of the first symbols in the right-hand sides of all binary productions. Then there exists a maximum m of time bounds for the symbols in F. For every such symbol $B \in F$ there also exists a finite set $t(B)$ of derivation trees in G_B. Let $t = \{\emptyset\} \cup \bigcup_{B \in F} t(B)$ be the set of all such derivation trees, also including the empty tree. We recall that t is finite.

We perform the following transformation of the grammar: we introduce non-terminals of the form $A[\tau_1, \cdots, \tau_{m-1}]$, $A \in N \cup \emptyset$, $\tau_i \in t$, $1 \leq i \leq m-1$. The new axiom is $S[\emptyset, \cdots, \emptyset]$. Every binary production $A \to BC$ is replaced by productions

$$\begin{aligned} A[\tau_1, \cdots, \tau_{m-1}] \to\ & \mathtt{yield}_0(\tau)\mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \\ & C[\tau, \tau_1, \cdots, \tau_{m-2}] \text{ for all } \tau \in t(B). \end{aligned}$$

Accordingly, productions $A \to C$, $C \in N$ are replaced by productions

$$A[\tau_1, \cdots, \tau_{m-1}] \to \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) C[\emptyset, \tau_1, \cdots, \tau_{m-2}],$$

and productions $A \to a$, $a \in T$ are replaced by productions

$$A[\tau_1, \cdots, \tau_{m-1}] \to a\ \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \emptyset[\emptyset, \tau_1, \cdots, \tau_{m-2}].$$

Finally, $\emptyset[\emptyset, \cdots, \emptyset] \to \lambda$ and

$$\emptyset[\tau_1, \cdots, \tau_{m-1}] \to \mathtt{yield}_1(\tau_1) \cdots \mathtt{yield}_{m-1}(\tau_{m-1}) \emptyset[\emptyset, \tau_1, \cdots, \tau_{m-2}].$$

In simple words, if the effect of one symbol is limited to m steps, then the choice of the corresponding derivation tree is memorized as an index in the other symbol, and needed terminals are produced in the right time. In total, m indexes suffice. It is easy to see that underlying grammar is regular, since only one non-terminal symbol is present. □

Lemma 6. $L = \{a^n b^n \mid n \geq 1\} \notin LOP(ncoo, tar)$.

Proof. Suppose there exists a context-free grammar $G = (N, T, S, P)$ in the Third normal form such that $L_t(G) = L$. Clearly, there must be a rule $A \to BC$ or $A \to CB \in P$ such that both B and C have unbounded time (by Lemma 5, since $L \notin REG$) and C has unbounded yield (since $L \notin FIN$).

Clearly, languages generated by any non-terminal from N must be scattered subwords of words from L, otherwise G would generate some language not in L. Thus, $L_t(G_B), L_t(G_C) \subseteq \{a^i b^j \mid i, j \geq 0\}$. It is not difficult to see that G_C must produce both symbols a and b. Indeed, since the language generated from C is infinite, substituting derivation trees for C with different numbers of one letter must preserve the balance of two letters. We now consider two cases, depending on whether $L_t(G_B) \subseteq a^*$.

If B only produces symbols a, then consider the shortest derivation tree τ in G_C. Since B has unbounded time, some symbol a can be generated after the first letter b appears in τ, so $L_t(G)$ generates some word not in L, which is a contradiction.

Now consider the case when B can produce a symbol b in some derivation tree τ in G_B. On one hand, a bounded number of letters a can be generated from B and C before the first letter b appears in τ; on the other hand, C has unbounded yield. Therefore, varying derivations under C we obtain a subset of $L_t(G)$ which is infinite, but the number of leading symbols a is bounded, so $L_t(G)$ contains words not in L, which is a contradiction. $\square$

Corollary 2. *$LIN \nsubseteq LOP(ncoo, tar)$.*

Lemma 7. *The family $LOP(ncoo, tar)$ is closed under permutations.*

Proof. For a given grammar $G = (N, T, S, P)$, consider a transformation where the terminal symbols a are replaced by non-terminals a_N throughout the description of G, and then the rules $a_N \to a_N$, $a_N \to a$, $a \in T$ are added to P. In a way similar to the first example, the order in which terminals are generated is arbitrary. $\square$

Corollary 3. *$\texttt{Perm}(REG) \subseteq LOP(ncoo, tar)$.*

Proof. Follows from regularity theorem 2 and permutation closure lemma 7. $\square$

The results of comparison of the membrane system family with the well-known language families can be summarized as follows:

Theorem 4. *$LOP(ncoo, tar)$ strictly contains REG and $\texttt{Perm}(REG)$, is strictly contained in CS, and is incomparable with LIN and CF.*

Proof. All inclusions and incomparabilities have been shown in or directly follow from Theorem 2, Corollary 3, Theorem 3, Corollary 2 and Corollary 1 with Theorem 1. The strictness of the first inclusions follows from the fact that REG and $\texttt{Perm}(REG)$ are incomparable, while the strictness of the latter inclusion holds since $LOP(ncoo, tar)$ only contains semilinear languages. $\square$

The lower bound can be strengthened as follows:

Theorem 5. *$LOP(ncoo, tar) \supseteq REG \bullet \mathtt{Perm}(REG)$.*

Proof. Indeed, consider the construction from the regularity theorem. Instead of erasing the symbol corresponding to the final state, rewrite it into the axiom of the grammar generating the second regular language, to which the permutation technique is applied. $\square$

Example 3. $LOP(ncoo, tar) \ni L_2 = \bigcup_{m,n\geq 1}(abc)^m\mathtt{Perm}((def)^n)$.

6 Closure properties

It has been shown above that the family of languages generated by basic membrane systems is closed under permutations. We now present a few other closure properties.

Lemma 8. *The family $LOP(ncoo, tar)$ is closed under erasing/renaming morphisms.*

Proof. Without restricting generality, we assume that the domain and range of a morphism h are disjoint (or rename the corresponding non-terminals). For a given grammar $G = (N, T, S, P)$, consider a transformation where every terminal symbol a becomes a non-terminal a' and the rules $a' \to h(a)$, $a \in T$ are added to P. It is easy to see that the new grammar generates exactly $h(L_t(G))$. $\square$

Corollary 4. $\{a^nb^nc^n \mid n \geq 1\} \notin LOP(ncoo, tar)$.

Proof. Assuming the contrary and applying the morphism defined by $h(a) = a$, $h(b) = b$, $h(c) = \lambda$ we obtain a contradiction with $L = \{a^nb^n \mid n \geq 1\} \notin LOP(ncoo, tar)$ from Lemma 6. $\square$

Corollary 5. *$LOP(ncoo, tar)$ is not closed under intersection with regular languages.*

Proof. By Example 2, $L = \{w \in T^* \mid |w|_a = |w|_b = |w|_c > 0\}$ belongs to the membrane systems language family. However, $L \cap a^*b^*c^* = \{a^nb^nc^n \mid n \geq 1\}$ does not, by Corollary 4. $\square$

Theorem 6. *$LOP(ncoo, tar)$ is closed under union and not closed under intersection or complement.*

Proof. The closure under union follows from adding a new axiom and productions of non-deterministic choice between multiple axioms. The family is not closed under intersection because it contains all regular languages (Theorem 2) and is not closed under intersection with them (Corollary 5). It follows that this family is not closed under complement, since intersection is the complement of union of complements. $\square$

Lemma 9. *$L = \bigcup_{m,n\geq 1} \mathtt{Perm}((ab)^m)c^n \notin LOP(ncoo, tar)$.*

Proof. Suppose there exists a context-free grammar $G = (N, T, S, P)$ in the Third normal form such that $L_t(G) = L$. Clearly, there must be a rule $A \to BC$ or $A \to CB \in P$ such that both B and C have unbounded time (by Lemma 5, since $L \notin REG$) and C has unbounded yield (since $L \notin FIN$). By choosing as $A \to BC$ or $A \to CB$ the rule satisfying above requirements which is first applied in some derivation of G, we make sure that all three letters a, b, c appear in words of $L_t(G_A)$.

Clearly, languages generated by any non-terminal from N must be scattered subwords of words from L, otherwise G would generate some language not in L. Thus, $L_t(G_B), L_t(G_C) \subseteq \{a, b\}^*c^*$. We now consider two cases, depending on whether $L_t(G_B) \subseteq \{a, b\}^*$.

If B only produces symbols a, b, then consider the shortest derivation tree τ in G_C. Since B has unbounded time, some symbol a or b can be generated after the first letter c appears in τ, so $L_t(G)$ generates some word not in L, which is a contradiction.

Now consider the case when B can produce a symbol c in some derivation tree τ in G_B. On one hand, a bounded number of letters a, b can be generated from B and C before the first letter c appears in τ; on the other hand, C has unbounded yield. Therefore, varying derivations under C we obtain a subset of $L_t(G)$ which is infinite, but the number of leading symbols a, b is bounded, so $L_t(G)$ contains words not in L, which is a contradiction. □

Corollary 6. *$LOP(ncoo, tar)$ is not closed under concatenation or taking the mirror image.*

Proof. Since $\bigcup_{m\geq 1} \mathtt{Perm}((ab)^m) \in \mathtt{Perm}(REG) \subseteq LOP(ncoo, tar)$ by Corollary 3 and $c^+ \in REG \subsetneq LOP(ncoo, tar)$ by Theorem 2, the first part of the statement follows from Lemma 9. Since $\bigcup_{m,n\geq 1} c^n\mathtt{Perm}((ab)^m) \in REG \bullet \mathtt{Perm}(REG) \subseteq LOP(ncoo, tar)$ by Theorem 5, the second part of the statement also follows from Lemma 9. □

7 A Difficult Language

In this section we give an example of a language in $LOP(ncoo, tar)$ which is considerably more "difficult" than languages in $REG \bullet \mathtt{Perm}(REG)$, in the sense informally explained below. Besides permutations of symbols sent out in the same time, it exhibits another kind of non-context-freeness. This second source of "difficulty" alone can, however, be captured as an intersection of two linear languages.

$$\begin{aligned} \Pi_D &= (\{D, D', a, b, c, a', b', c'\}, [_1\ \ ]_1, D^2, R), \\ R &= \{D \to (abc)_{out}D'D', D \to (abc)_{out}, \\ &\qquad D' \to (a'b'c')_{out}DD, D' \to (a'b'c')_{out}\}. \end{aligned}$$

The contents of region 1 is a population of objects D, initially 2, which are primed if the step is odd. Assume that there are k objects inside the system. At every step, every symbol D is either erased or doubled (and primed or de-primed), so the next step the number of objects inside the system will be any even number between 0 and $2k$. In addition to that, the output during that step is $\texttt{Perm}((abc)^k)$, primed if the step is odd. Hence, the generated language can be described as

$$\begin{aligned} L(\Pi_D) = \{ & \texttt{Perm}((abc)^{2k_0})\texttt{Perm}((a'b'c')^{2k_1}) \cdots \\ & \texttt{Perm}((abc)^{2k_{2t}})\texttt{Perm}((a'b'c')^{2k_{2t+1}}) \\ & \mid k_0 = 1,\ 0 \le k_i \le 2k_{i-1},\ 1 \le i \le 2t+1,\ t \ge 0\}. \end{aligned}$$

For an idea of how complex a language generated by some non-cooperative membrane system be, imagine that the skin may contain populations of multiple symbols that can (like D in the example above) be erased or multiplied (with different periods), and also rewritten into each other. The same, of course, happens in usual context-free grammars, but since the terminal symbols are collected from the derivation tree level by level instead of left to right, the effect is quite different.

8 Conclusions

In this paper we have reconsidered the family of languages generated by transitional P systems without cooperation and without additional control. It was shown that one membrane is enough, and a characterization of this family was given via derivation trees of context-free grammars. Next, three normal forms were given for the corresponding grammars. It was than shown that the membrane systems language family lies between regular and context-sensitive families of languages, and it is incomparable with linear and with context-free languages. Then, the lower bound was strengthened to $REG \bullet \texttt{Perm}(REG)$. An example of a considerably more "difficult" language was given than the lower bound mentioned above.

The membrane systems language family was shown to be closed under union, permutations, erasing/renaming morphisms. It is not closed under intersection, intersection with regular languages, complement, concatenation or taking the mirror image.

The following are examples of questions that are still not answered.

- Clearly, $LOP(ncoo, tar) \not\supseteq MAT$. What about $LOP(ncoo, tar) \subseteq MAT$?
- Is $LOP(ncoo, tar)$ closed under arbitrary morphisms? The difficulty is to handle $h(a) = bc$ if many symbols a can be produced in the same step.
- Look for sharper lower and upper bounds.

Acknowledgments Artiom Alhazov gratefully acknowleges the support of the Japan Society for the Promotion of Science and the Grant-in-Aid for Scientific Research, project 20·08364. All authors acknowledge the support by the Science and Technology Center in Ukraine, project 4032.

References

1. Alhazov, A., Ciubotaru, C., Rogozhin, Yu., Ivanov, S.: The Membrane Systems Language Class. Brainstorming Week on Membrane Computing, Sevilla, submitted (2010) and LA Symposium, RIMS Kôkyûroku, Kyoto, submitted (2010)
2. Alhazov, A., Sburlan, D.: Static Sorting P Systems. Applications of Membrane Computing, Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J. (eds.), Natural Computing Series, Springer-Verlag, 215–252 (2005)
3. Bernardini, F., Gheorghe, M.: Language Generating by means of P Systems with Active Membranes. Brainstorming Week on Membrane Computing, Technical Report, 26/03, Rovira i Virgili University, Tarragona, 46–60 (2003)
4. Csuhaj-Varjú, E.: P Automata. In: Membrane Computing, 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers, LNCS, vol. 3365, Springer, 19–35 (2005)
5. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, Gy.: On the Computational Complexity of P Automata. Natural Computing, vol. 5, 2, 109–126 (2006)
6. Freund, R., Verlan, S.: A Formal Framework for Static (Tissue) P systems. In: Eleftherakis, G.. Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.), Membrane Computing. 8th International Workshop, WMC 2007, Thessaloniki, Revised Selected and Invited Papers, LNCS, 4860, Springer, 271–284 (2007)
7. Ibarra, O.H., Păun, Gh.: Characterizations of Context-Sensitive Languages and Other Language Classes in Terms of Symport/Antiport P Systems. Theoretical Computer Science, vol. 358, 1, 88–103 (2006)
8. Păun, Gh.: Membrane Computing. An Introduction, Springer-Verlag, Berlin (2002)
9. Păun, Gh., Rozenberg, G., Salomaa, A.: Membrane Computing with an External Output. Fundamenta Informaticae, vol.41, 3, 313–340 (2000)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press (2010)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1-3, Springer (1997)
12. Qi, Z., You, J., Mao, H.: P Systems and Petri Nets. In: Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A.(eds.): Membrane Computing. International Workshop, WMC 2003, Tarragona, Revised Papers, LNCS, vol. 2933, Springer, 286–303 (2004)
13. P systems webpage. `http://ppage.psystems.eu/`