Polymorphic P Systems with Limited Depth

Anna Kuczik and György Vaszil

Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
kuczik.anna@inf.unideb.hu
vaszil.gyorgy@inf.unideb.hu

Summary. We investigate the computational power of non-cooperating polymorphic P systems with no additional ingredients and having a membrane structure of limited depth. We show that any ET0L language can be generated by such systems with a membrnae structure of depth three.

Key words: Program as data, P systems with dynamic rules, Polymorphic P systems, P systems with non-cooperating rules, P systems with limited depth

1 Introduction

Polymorphic P systems were introduced in [1] motivated by the idea that the program of a computing device could be viewed as data, therefore, it could also be changed during the course of the computation. In these types of P systems, rules are not statically defined, but are dynamically inferred from the contents of pairs of membranes: The contents of one member of the pair defines the multiset representing the left-hand side of the rule, the contents of the other member defines the right-hand side. As the membranes can contain further membranes, the contents of the pairs, and this way the left- and right-hand sides of rules may change dynamically during the computation.

The initial results presented in [1] show the power of the model. With cooperating rules (rules with left-hand sides with more than one objects) any recursively enumerable set of numbers can be generated, but non-cooperating systems (systems with rules with just one object on the left-hand side) can also generate several interesting languages, mainly based on the fact that an exponential, even super-exponential growth of the number of objects inside the system can be produced.

The study of non-cooperating variants of the model was continued further in [3] with considering the case of "no ingredients", that is, when no special features (not even target indicators) are added to the system. The equivalence of so called strong and weak polymorphism was shown, left polymorhism, right polymorphism,

and general polymorphism was defined. As its main contribution, [3] presented a hierarchy of computational power based on the depth of the membrane structure, but the computational power of the non-cooperating variant remained unclear.

In the present work, we intend to take some initial steps in this direction by showing that any ET0L language can be generated using non-cooperating polymorphic P systems (with no other ingredients) of depth three. In the following we first review the necessary definitions, then present an example wehere a simple ET0L system is simulated, then finally generalize the idea of the simulation to a method for generating any ET0L language.

2 Preliminaries

In the following we breiefly define the basic notions we will use. See [6] for more on formal language theory, and [4, 5] for details about membrane computing.

Multisets are sets with multiplicites associated with their elements. Let U be a set. A *multiset* over U is a mapping $M : U \to \mathbb{N}$, $M(a)$, for all $a \in U$, is the multiplicity of a in the multiset M. We can also use the form $(a, M(a))$. If U is finite, $U = \{a_1, a_2, \dots a_n\}$, then $\{(a_1, M(a_1)), (a_2, M(a_2)), \dots, (a_n, M(a_n))\}$ can also be represented by a string $w = a_1^{M(a_1)}, a_2^{M(a_2)}, \dots, a_n^{M(a_n)}$ (and all permutations of this string).

In formal language theory, an *alphabet* V is a finite non-empty set of symbols, its cardinality is denoted by $|V|$. A string generated by V under the operation of concatenation is denoted by V^*, and $V^+ = V^* \setminus \{\lambda\}$ where λ denotes the empty string.

Lindenmayer systems (or *L systems*) are parallel rewriting systems introduced in 1968 by A. Lindenmayer. Several variants of L systems have been developed since then, among these, we will use ET0L systems and languages.

A finite substitution τ over an alphabet V is a function mapping each symbol $a \in V$ into a non-empty finite language over: $V : \tau(a) \subseteq V^*$. We extend τ to words by $\tau(\lambda) = \{\lambda\}$, $\tau(w) = \tau(a_1)\tau(a_2)\dots\tau(a_n)$ for $w = a_1a_2\dots a_n$, and to languages by $\tau(L) = \{\tau(w) | w \in L\}$.

An *ET0L system* is a 4-tuple $G = (V, T, U, w)$ where V is an alphabet and $T \subseteq V$ is a terminal alphabet are finite sets, $w \in V^+$ is the initial word of G, and U is a finite set of finite substitutions over V (called the *tables* of U). In a computational step in G, all the symbols of the current sentential form (starting with the axiom) are substituted (or rewritten) using one of the tables of U. The language generated by G consists all terminal strings which can be generated in a series of computational steps (a derivation) from the initial word, that is, $L(G) = \{u \in T^* \mid w \Rightarrow^* u\}$ where $\Rightarrow$ denotes a comptutaional step, and $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. The family of languages generated by ET0L systems is denoted by $\mathcal{L}(ET0L)$.

It is known (see [2]) that for each ET0L system with an arbitrary number of tables, there exists an ET0L system with only two tables generating the very same

language. This means that every task which can be solved by an arbitrary ET0L system can also be solved by a system using two tables. Therefore, in the following example and model, we will assume that ET0L systems have two tables.

Moreover, since we are going to relate ET0L languages to the multiset langugaes of P systems, the most important thing is not the string generated by the ET0L system, but the multiplicities of different letters in the generated strings. We will denote by $N(G)$ the language of multisets corresponding the strings of $L(G)$, and by $\mathcal{L}(NET0L)$ the obtained class of multiset langugaes.

A membrane systms (or *P system*) is a tuple

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, h_o),$$

where O is an alphabet of objects, $T \subseteq O$ is the set of terminal objects, μ is the membrane structure, w_i are the multisets giving the initial contents of each membrane $1 \leq i \leq n$, R_i is a finite set of rules for each membrane $1 \leq i \leq n$, and h_o is the label of the output membranes, $h_o \in \{1, \ldots, n\}$.

The membrane structure μ is usually denoted by a string of matching parentheses labelled by the numbers $\{1, \ldots, n\}$, but it can also be represented by a tree with its root labelled by the label of the outermost membrane, and the descendant nodes of each node labelled by the labels of membranes enclosed by the region corresponding to the given node. In the following, the number of nodes encountered during the traversal of the longest path from the root to a leaf in such a tree representation will be called the *depth* of the membrane system. (For example, the membrane system which only has one membrane is of depth 1, the system with two nested membrane is of depth 2.)

The rules in R_i, $1 \leq i \leq n$, are given as multiset rewriting rules, of the form $u \to v$, where $u, v \in V^*$ are strings (understood as representations of multisets). If in such rules, the number of objects in u (the multiset on the left side of the rules) is greater than one, then we say that Π is a system with *cooperation*. Otherwise, it is a $non - cooperative$ system.

The rules in a given R_i are applied in the region enclosed by membrane i in a maximally parallel way, that is, as many rules have to be applied in parallel as possible with the restriction that each object can be rewritten by at most one rule.

2.1 The polymorphic P system model

In polymorphic membrane systems, unlike traditional membrane systems, the rules are not directly defined. The rules are represented by the membranes. The left and right sides of each rule are contained by a membrane. Consequently, the structure of the polymorphic membrane system will be different.

A polymorphic P system is a tuple

$$\Pi = (O, T, \mu, w_s, \langle w_{1L}, w_{1R} \rangle, \ldots, \langle w_{nL}, w_{nR} \rangle, h_o),$$

where O is the alphabet of objects, $T \subseteq O$ is the set of terminal objects, μ is the membrane structure consisting $2n + 1$ membranes labelled by $s, 1L, 1R, \ldots, nL,$

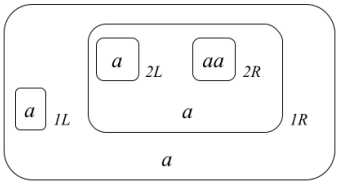

Fig. 1. The polymorphic P system Π_1 of Example 1.

nR, the multiset w_s is the initial contents of the skin membrane, $\langle w_{iL}, w_{iR} \rangle$ are pairs of multisets giving the contents of membranes iL and iR, $1 \leq i \leq n$, and $h_o \in \{1, \ldots, n\}$ is the label of the output membrane.

The *depth* is defined in the same way as conventional membrane systems, it is the height of μ seen as a tree. For every $1 \leq i \leq n$, the membranes iL and iR have the same parent membrane, so they are located at the same depth.

The rules of Π are not given statically in the description, but are dynamically deduced for each configuration based on the content of the membrane pairs iL and iR. Thus, if in the configuration of the system these membranes contain the u and v multisets, then in the next step their parent membrane is transformed as if the $u \to v$ multiset transcription rule were added to it. If iL is empty in some configuration, then the rule defined by the pair iL, iR is considered disabled, that is, no rule will be inferred from the contents of iL and iR.

Similarly to [1], polymorphic membrane systems and their languages are denoted as $NOP^k(polym, ncoo)$ and $\mathcal{L}(NOP^k(polym, ncoo))$ where k denotes the depth, *polym* means polymorphism, and *ncoo* means that the system is $non - cooperative$.

Now we recall an example of a simple polymorphic membrane system with superexponential growth from [1].

Example 1. Consider the polymorphic P system

$$\Pi_1 = (\{a\}, \{a\}, \mu, a, \langle a, a \rangle, \langle a, aa \rangle, S)$$

having a membrane structure as illustrated in Figure 2.1.

In the initial configuration, rule 1 looks like $a \to a$, because of the contents of $1L$ and $1R$, while rule 2 in membrane $1R$ is $a \to aa$. In the first step, rule 1 is applied in the skin, leaving the contents of the membrane intact, and rule 2 is applied in membrane $1R$ doubling the number of a's in 1R, so rule 1 will be changed to the form $a \to aa$. In the second step, rule 1 will transform the multiset a in the skin into aa. and rule 2 is applied in membrane 1R and double the contents again, so after that, rule 1 looks like $a \to a^4$. In general, after k derivation steps, 1R will be a^{2^k}, so rule 1 will have the form $a \to a^{2^k}$. As the number of a's in the skin will be $2^{\frac{k(k-1)}{2}}$, the rate of growth of the contents of the skin membrane is superexponential.

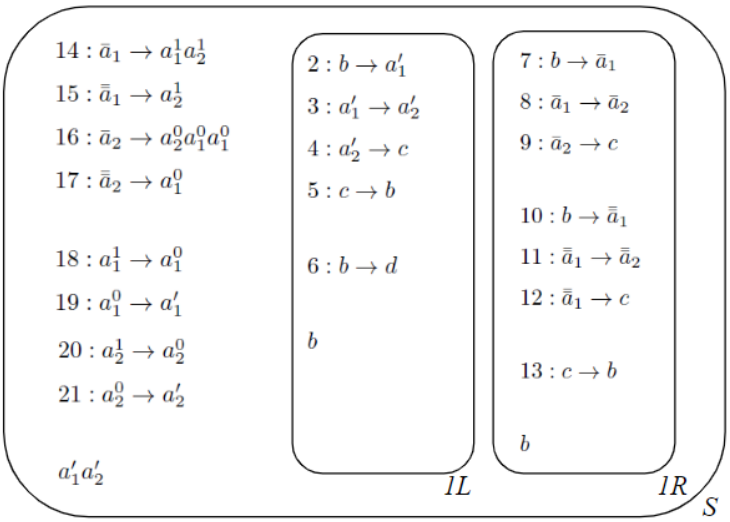

Fig. 2. The P system Π of Example 2.

3 Polymorphic P systems with limited depth

In this section we would like to examine the relationship of languages generated by ET0L systems and simple polymorphic P systems, where simplicity is captured by non-cooperation and limited depth. We look at an example first.

Example 2. Consider the following ET0L system $G = (V, T, U, w)$ with $V = T = \{a_1, a_2\}$, $w = a_1 a_2$, and two tables $U = (P_1, P_2)$, each containing two rules

$$P_1 = \{a_1 \to a_1 a_2,\ a_2 \to a_2 a_1 a_1\}, \text{ and}$$
$$P_2 = \{a_1 \to a_2,\ a_2 \to a_1\}.$$

We construct a non-cooperative polymorphic P system Π with depth 3 that can perform the choosing between rules of P_1 and P_2, and therefore simulates the operation of G.

Let $O = \{a_1, a_2, a_1', a_2', a_1^0, a_2^0, a_1^1, a_2^1, \bar{a}_1, \bar{a}_2, \bar{\bar{a}}_1, \bar{\bar{a}}_2, b, c, d, \}$, $T' = \{a_1', a_2'\}$ and

$$\Pi_2 = (O, T', \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{21L}, w_{21R}\rangle, s)$$

where the membrane structure of Π is defined as

$$\mu = [\ [\ldots]_{1L}\ [\ldots]_{1R}\ [\ ]_{14L}\ [\ ]_{14R}\ \cdots [\ ]_{21L}\ [\ ]_{21R}\ ]_s$$

with membrane $1L$ containing the inner membranes $[\ ]_{2L}\ [\ ]_{2R} \ldots [\ ]_{6L}\ [\ ]_{6R}$, and membrane $1R$ containing the inner membranes $[\ ]_{7L}\ [\ ]_{7R} \ldots [\ ]_{13L}\ [\ ]_{13R}$.

We use two types of rules in polymorphic membrane systems. The rules belonging to the first type do not change during the solution of the task, while the

rules belonging to the second type can change step by step. In this example we only have one rule that changes during the steps, rule 1. The other rules have the same form at every step during the process.

The graphical representation of μ can be seen in Figure 2 where also the initial membrane contents are depicted. Non-dynamical rules, that is, pairs of membranes $[\ w_{iL}\ ]_{iL},\ [\ w_{iR}\ ]_{iR}$ with constant contents (contents that never change during the computation) are given in a simplified notation as $w_{iL} \to w_{iR}$.

The initial contents of the regions effected by the polymorphic nature of Π, that is, the regions with non-constant contents are

$$w_s = a'_1 a'_2,\ w_{1L} = b,\ w_{1R} = b.$$

Step	Rule 1	Skin	1L	1R	Rules 14-21
1.	$b \to b$	$a'_1 a'_2$	2	7	
2.	$a'_1 \to \bar{a}_1$	$a'_1 a'_2$	3	8	
3.	$a'_2 \to \bar{a}_2$	$\bar{a}_1 a'_2$	4	9	14
4.	$c \to c$	$a^1_1 a^1_2 \bar{a}_2$	5	13	$16, 18, 20$
5.	$b \to b$	$a^0_1 a^0_2 a^0_2 a^0_1 a^0_1$	2	7 or 10	$19, 21$
6.	$a'_1 \to \bar{a}_1$ or $a'_1 \to \bar{\bar{a}}_1$	$a'_1 a'_2 a'_2 a'_1 a'_1$	...	...	...

Table 1. The polymorphic system Π of example 2

The functioning of Π is demonstrated in Table 1. The first column contains the step number. The second column contains how rule 1, defined by the membranes $1L$ and $1R$ looks like after every step. The third column contains the elements of the skin region. The fourth, fifth, and sixth columns contain the rules we (need to) use in the corresponding steps.

The general idea behind the functioning of Π_2 is as follows. Rules $14 - 17$ simulate the rewriting process of the tables of G. Those with lefthand side $\bar{a}_1$ or $\bar{a}_2$ simulate the first table, those with lefthand side $\bar{\bar{a}}_1, \bar{\bar{b}}_2$ simulate the second table. The objects of the skin region correspond to the sentential form of G. Rule 1 is "dynamic", it prepares the objects of the skin membrane for the application of the rules $14 - 17$ in the appropriate order. At the beginning of a "simulating cycle", rule 1 is used to rewrite a (more precisely, its variant, a'_1) to $\bar{a}_1$ or $\bar{\bar{a}}_2$, selecting this way the table to be simulated. Then, rule 1 changes to rewrite a'_2 according to the same selection, while rules $14 - 17$ proceed with the actual simulation of the chosen table. The rest of the rules are needed to synchronize the whole process.

Table 1 shows how the rewriting of $a_1 a_2$ to $a_1 a_2 a_2 a_1 a_1$ by the first table of G is simulated in Π. In the initial state, rule 1 looks like $b \to b$, which is not applicable, because we only have an a'_1 and a a'_2 in the skin.

So in the first step, we have to change rule 1. In $1L$ we can use rule 2 ($b \to a'_1$), which rewrites b in $1L$ to a'_1 making rule 1 is applicable, because we can use it to write a'_1 in the skin. In parallel, we have to use rule 7 ($b \to \bar{a}_1$) or rule 10 ($b \to \bar{\bar{a}}_2$) in $1R$. This decision depends on which table of the ET0L system we want to simulate. To simulate P_1, we must use rule 7, and to simulate P_2, we must use rule 10. As we would like to simulate P_1, we use rule 7.

As can be seen in the second row of Table 1, the form of rule 1 has changed, and now we can use it in the skin and rewrite a'_1 to $\bar{a}_1$.

At the same time, the rules used in $1L$ and $1R$ ($a'_1 \to a'_2$, $\bar{a}_1 \to \bar{\bar{a}}_2$, respectively), changed the shape of rule 1 to $a'_2 \to \bar{\bar{a}}_2$, in order to be able to start rewriting a'_2-s in the next step.

After we used rule 1, the object(s) in the skin changed. Now, we can use rule 14 ($\bar{a}_1 \to a_1^1 a_2^1$), which simulates the first rule from the P_1 ET0L table of G. The upper indexing of the symbols on the right-hand side starts from 1, so that they are written back into the primed form (after counting down with the indices to zero) at the appropriate step.

Meanwhile, in step 3, rule 1 ($a'_2 \to \bar{\bar{a}}_2$) is also applied to rewrite a'_2 (so the second rule of table P_1 of G can also be simulated), and rule 1 is changed to $c \to c$ (so it cannot be applied in the next step).

Now, with rule 16 ($\bar{\bar{a}}_2 \to a_2^0 a_1^0 a_1^0$), the rule $a_2 \to a_2 a_1 a_1$ of G is simulated, while rules 18 and 20 decrement the upper indices of the objects introduced by the simulation of the previous rule, and rule 1 is changed to $b \to b$.

Now, as can be seen in row 5. of Table 1, the system is ready to prepare the next simulating cycle by rewriting the objects corresponding to the sentential form of G to their primed versions, and changing rule 1 in the appropriate way. We returned to a state that was similar to the initial state, where $1L$ has b and $1R$ also has b, so we can choose between rule 7 and rule 10 again (to simulate another step from the ET0L system), and in parallel, rewrite a_1^0-s and a_2^0-s to a'_1-s and a'_2-s, with rules 19 and 21.

The simulation of the ET0L system can be completed when it returns to a state that is similar to the initial state, so before another table is selected for simulation. So, if $1L$ has b and $1R$ also has b, and we want to stop the mechanism, then we can choose rule 6 instead of rule 7 or rule 10. Rule 6 shuts down the system and the simulation ends. The reason for this is that after applying rule 6, the form of rule 1 is: $d \to \bar{a}_1$ or $d \to \bar{\bar{a}}_1$, and we can't use any of these rules in the Skin membrane, because we don't have a d object in Skin.

The result will be a string made from the letters $\{a'_1, a'_2, \ldots\}$ in the *Skin* membrane of the system.

Now we show that the basic idea of the example above can be generalized to arbitrary ET0L systems. Without the loss of generality, we assume that we deal with systems having two tables.

Theorem 1. $\mathcal{L}(NET0L) \subseteq \mathcal{L}(NOP^3(polym, ncoo))$.

Proof. Let $G = (N, T, U, w)$ be an ET0L system and let k denote the number of letters in the alphabet, $N = \{a_1, a_2, ..., a_k\}$. The two tables are $U = (P_1, P_2)$, and the rules are denoted by $r_{i,j}$, where index $i \in \{1, 2\}$ denotes the index of the table, and $1 \leq j \leq m$ is the index of the rule, with m being the maximum of $|P_1|$ and $|P_2|$, the number of rules in the two tables.

We denote the left- and righthand sides of the rules as $r_{i,j} : \alpha_{i,j} \to \beta_{i,j}$, where $\alpha_{i,j} \in N$ and $\beta_{i,j} \in N^*$, $1 \leq i \leq 2,\ 1 \leq j \leq m$.

Let

$$O = \{a_i, a'_i, a^n_i, \bar{a}_i, \bar{\bar{a}}_i \mid 1 \leq i, n \leq k\} \cup \{b, c, d, \},\ T' = \{a'_i \mid 1 \leq i \leq k\}$$

and let

$$\Pi = (O, T', \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{pL}, w_{pR}\rangle, s)$$

where $p = 2 + (3k + 6) + 2m + \frac{k(k+1)}{2}$, and the membrane structure of Π is defined as

$$\mu = [\ [\ldots]_{1L}\ [\ldots]_{1R}\ [\ ]_{(3k+7)L}\ [\ ]_{(3k+7)R}\ \cdots [\ ]_{pL}\ [\ ]_{pR}\ ]_s$$

with membrane $1L$ containing the inner membranes $[\ ]_{2L}\ [\ ]_{2R} \ldots [\ ]_{6L}\ [\ ]_{6R}$, and membrane $1R$ containing the inner membranes $[\ ]_{7L}\ [\ ]_{7R} \ldots [\ ]_{13L}\ [\ ]_{13R}$.

In 1L, the number of rules depends on the number of letters in the alphabet of the ET0L system, we have to apply $k + 2$ rules for each table simulation in succession, where $k = |N|$. In general, we specify the rules for k letters as

$$b \to a'_1,\ a'_i \to a'_{i+1},\ \text{for } 1 \leq i \leq k-1,\ \text{and } a'_k \to c,\ c \to b.$$

They perform the same task as the rules of 1L in Example 2 do for two letters.

Note that here we have used the simplified notation again for membranes with contents that remain constant for the whole computation. (Without this simplification we would have to write $\langle w_{2L}, w_{2R}\rangle$ and specify $w_{2L} = b$, $w_{2R} = a'_1$ instead of the rule $b \to a'_1$, and so on.)

Rules must be applied in $1R$ depending on the choice of the table, because those rules give the right side of rule 1, which modifies the objects in the skin. So we have to create rules for also P_1 and P_2, like in the example. We need

$$b \to \bar{a}_1,\ \bar{a}_i \to \bar{a}_{i+1} \text{ for } 1 \leq i \leq k-1,\ \bar{a}_k \to c,\ c \to b,$$

and

$$b \to \bar{\bar{a}}_1,\ \bar{\bar{a}}_i \to \bar{\bar{a}}_{i+1} \text{ for } 1 \leq i \leq k-1,\ \bar{\bar{a}}_k \to c,\ c \to b.$$

In the skin region we have to go through the rules of a table in order, and for this reason we do not rewrite the letters at the same time. We have to add extra rules, which help us get back to the form $a'_1, a'_2, \ldots$ for all objects at the same step, after rewriting the last letter.

We denote the jth rule of table 1 and table 2 with $r_{1,j}$ and $r_{2,j}$, respectively. In order to simplify the notation, we assume that the cardinality of the two tables are the same, $m = |P_1| = |P_2|$. If this is not the case, $|P_1| < |P_2|$ for example, then

we consider the "missing rules" $r_{1,j},\ P_1| < j \leq |P_2|$, to be $a_1 \to a_1$. Now we add the following rules to the skin region.

Rule set for the rules of P_1, the first table of $U = (P_1, P_2)$:

$$\{\bar{a}_i \to \beta_{1,j}^{k+1-i} \mid \alpha_{1,j} \to \beta_{1,j} \in P_1, \alpha_{1,j} = a_i \text{ for some } a_i \in \{a_1, a_2, \ldots a_k\}\}.$$

Rule set for the rules of P_2 of $U = (P_1, P_2)$:

$$\{\bar{\bar{a}}_i \to \beta_{2,j}^{k+1-i} \mid \alpha_{2,j} \to \beta_{2,j} \in P2, \alpha_{2,j} = a_i \text{ for some } a_i \in \{a_1, a_2, \ldots a_k\}\}.$$

After rewriting with the rules above, we have to use rules to count down a_i-s indexes until the last letter is transcribed, similarly as we count down in the example. Thus, every a_i gets $k + 1 - i$ indices as follows.

$$a_i^n \to a_i^{n-1}, a_i^1 \to a_i^{'}, \text{ where } 2 \leq n \leq k + 1 - i, 1 \leq i \leq k - 1.$$

To stop the system, we need to add another rule to $1L$. The rule must be one that can be applied at the end of a table simulation. For this reason, when $1L$ and $1R$ return to a state similar to the initial state, there should be an option to apply the stopping rule, which is $b \to d$, similar to the system of Example 2.

So we add the rule

$$b \to d$$

to $1L$.

After we used this rule the system shut down, because we never have d object in the skin region. So after this step we can't continue the rule application, the system halts. The result will be a string consisting of the letters $\{a_1', a_2', \ldots\}$ in the skin membrane of the system.

4 Conclusion

We have shown how a simple ET0L system can be simulated by a non-cooperating polymorphic P system of depth three, and then generalized the idea to produce any ET0L language. Our work is intended to be the initial step in the investigation of the computing power of non-cooperating systems with limited depth. The next topic of further study is looking for upper bounds on the computational power, and in particular, to relate left and right polymorphism and their depth-limited variants to the general case.

References

1. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94, Berlin, Heidelberg, 2011. Springer-Verlag.

2. Andrzej Ehrenfeucht, Grzegorz Rozenberg, and Sven Skyum. A relationship between ET0L and EDT0L languages. *Theoretical Computer Science*, 1(4):325–330, 1976.
3. Sergiu Ivanov. Polymorphic P systems with non-cooperative rules and no ingredients. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing*, volume 8961 of *Lecture Notes in Computer Science*, pages 258–273, Cham, 2014. Springer International Publishing.
4. Gheorghe Păun. *Membrane Computing: An Introduction*. Springer-Verlag, Berlin, Heidelberg, 2002.
5. Gheorghe Păun, Grzegorz Rozenberg, and Aarto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, 2010.
6. Grzegorz Rozenberg and Aarto Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, Berlin Heidelberg, 1997.