Computer Science Journal of Moldova, vol.**16**, no.3(48), 2008

About Precise Characterization of Languages Generated by Hybrid Networks of Evolutionary Processors with One Node *

Artiom Alhazov Yurii Rogozhin

Abstract

A hybrid network of evolutionary processors (an HNEP) is a graph where each node is associated with an evolutionary processor (a special rewriting system), a set of words, an input filter and an output filter. Every evolutionary processor is given with a finite set of one type of point mutations (an insertion, a deletion or a substitution of a symbol) which can be applied to certain positions of a string over the domain of the set of these rewriting rules. The HNEP functions by rewriting the words that can be found at the nodes and then re-distributing the resulting strings according to a communication protocol based on a filtering mechanism. The filters are defined by certain variants of random-context conditions. In this paper we complete investigation of HNEPs with one node and present a precise description of languages generated by them.

1 Introduction

Insertion, deletion, and substitution are fundamental operations in formal language theory, their power and limits have obtained much attention during the years. Due to their simplicity, language generating mechanisms based on these operations are of particular interest. *Networks of evolutionary processors* (NEPs, for short), introduced in [8],

*The authors gratefully acknowledge the Science and Technology Center in Ukraine, project 4032 and the second author acknowledges the support of European Commission, project MolCIP, MIF1-CT-2006-021666.

are proper examples for distributed variants of these constructs. In this case, an evolutionary processor (a rewriting system which is capable to perform an insertion, a deletion, and a substitution of a symbol) is located at every node of a virtual graph which may operate over sets or multisets of words. The system functions by rewriting the collections of words present at the nodes and then re-distributing the resulting strings according to a communication protocol defined by a filtering mechanism. The language determined by the network is defined as the set of words which appear at some distinguished node in the course of the computation. These architectures also belong to models inspired by cell biology, since each processor represents a cell performing point mutations of DNA and controlling its passage inside and outside the cell through a filtering mechanism. The evolutionary processor corresponds to the cell, the generated word to a DNA strand, and the operations insertion, deletion, and substitution of a symbol to the point mutations. It is known that, by using an appropriate filtering mechanism, NEPs with a very small number of nodes are computationally complete computational devices, i.e. they are as powerful as the Turing machines (see, for example [5, 6]).

Particularly interesting variants of these devices are the so-called *hybrid networks of evolutionary processors* (HNEPs), where each language processor performs only one of the above operations on a certain position of the words in that node. Furthermore, the filters are defined by some variants of random-context conditions, i.e., they check the presence/absence of certain symbols in the words. These constructs can be considered both language generating and accepting devices, i.e., generating HNEPs (GHNEPs) and accepting HNEPS (AHNEPs). The notion of an HNEP, as a language generating device, was introduced in [16] and the concept of an AHNEP was defined in [15].

In [9] it was shown that, for an alphabet V, GHNEPs with $27 + 3 \cdot card(V)$ nodes are computationally complete. A significant improvement of the result can be found in [1], where it was proved that GHNEPs with 10 nodes (irrespectively of the size of the alphabet) obtain the universal power. For accepting HNEPs, in [13] it was shown that for any recursively enumerable language there exists a recognizing

AHNEP with 31 nodes; the result was improved significantly in [14] where the number of necessary nodes was reduced to 24. Furthermore, in [14] the authors demonstrated a method to construct for any NP-language L an AHNEP with 24 nodes which decides L in polynomial time.

At last in [2] it was proved that any recursively enumerable language can be generated by a GHNEP having 7 nodes (thus, the result from [1] is improved) and in [3] the same authors showed that any recursively enumerable language can be accepted by an AHNEP with 7 nodes (thus, the result from [14] is improved significantly). In [3] also it was showed that the families of GHNEPs and AHNEPs with 2 nodes are not computationally complete.

In [9] it was demonstrated that a GHNEP with one node can generate only regular language, but now in this paper we present a precise form of the generated language and consider one case omitted in the previous proof. Tasks of characterization of languages generated by a GHNEP with two nodes and languages accepting by an AHNEP with two nodes are still opened.

2 Prerequisites

We first recall some basic notions from formal language theory that we shall use in the paper. An alphabet is a finite and non-empty set of symbols. The cardinality of a finite set A is denoted by $card(A)$. A sequence of symbols from an alphabet V is called a word (or a string) over V. The set of all words over V is denoted by V^*; the empty word is denoted by ε; and we define $V^+ = V^* \setminus \{\varepsilon\}$. The length of a word x is denoted by $|x|$, and we designate the number of occurrences of a letter a in a word x by $|x|_a$. For each non-empty word x, $alph(x)$ denotes the smallest alphabet Σ such that $x \in \Sigma^*$.

The shuffle operation defined on two words $x, y \in V^*$ by

$$\coprod(x,y) = \{x_1y_1x_2y_2\dots x_ny_n \mid n \geq 1,\ x_i, y_y \in V^*, \\ x = x_1x_2\dots x_n,\ y = y_1y_2\dots y_n\}.$$

Let $L_1, L_2 \in V^*$ are two languages. Then

$$\text{Ш}(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} \text{Ш}(x, y).$$

A *type-0 generative grammar* is a quadruple $G = (N, T, S, P)$, where N and T are disjoint alphabets, called the nonterminal and terminal alphabet, respectively, $S \in N$ is the start symbol or the axiom, and P is a finite set of productions or rewriting rules of the form $u \to v$, where $u \in (N \cup T)^* N (N \cup T)^*$ and $v \in (N \cup T)^*$. For two strings x and y in $(N \cup T)^*$, we say that x directly derives y in G, denoted by $x \Longrightarrow_G v$, if there is a production $u \to v$ in P such that $x = x_1 u x_2$ and $y = x_1 v x_2$, $x_1, x_2 \in (N \cup T)^*$ holds. The transitive and reflexive closure of $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^*$. The language $L(G)$ generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow_G^* w\}$.

We recall now a concept dual to a type-0 generative grammar, called a *type-0 analytic grammar* [17]. A type-0 analytic grammar $G = (N, T, S, P)$ is a quadruple where N, T, S are defined in the same way as for a generative grammar, and P is a finite set of productions of the form $u \to v$, where $u \in (N \cup T)^*$ and $v \in (N \cup T)^* N (N \cup T)^*$. The derivation relation is defined for a type-0 analytic grammar analogously to the derivation relation for a type-0 generative grammar. The language $L(G)$ recognized or accepted by a type-0 analytic grammar $G = (N, T, S, P)$ is defined as $L(G) = \{w \in T^* \mid w \Longrightarrow_G^* S\}$.

It is well-known that for the type-0 analytic grammar G' obtained from a type-0 generative grammar G with interchanging the left and the right hand sides of the productions in G, it holds that $L(G') = L(G)$.

In the sequel, following the terminology in [9], we recall the necessary notions concerning evolutionary processors and their hybrid networks. These language processors use so-called evolutionary operations, simple rewriting operations which abstract local gene mutations.

For an alphabet V, we say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are different from ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; and, it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution rules, deletion rules, and insertion rules over an alphabet V is denoted by Sub_V, Del_V, and Ins_V, respectively. Given such rules π, ρ, σ, and a word $w \in V^*$, we define the following *actions* of σ on w: If $\pi \equiv a \to b \in Sub_V$, $\rho \equiv a \to \varepsilon \in Del_V$, and

$\sigma \equiv \varepsilon \to a \in Ins_V$, then

$$\pi^*(w) = \begin{cases} \{ubv : \exists u, v \in V^*(w = uav)\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{1}$$

$$\rho^*(w) = \begin{cases} \{uv : \exists u, v \in V^*(w = uav)\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{2}$$

$$\rho^r(w) = \begin{cases} \{u : w = ua\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{3}$$

$$\rho^l(w) = \begin{cases} \{v : w = av\}, \\ \{w\}, \quad \text{otherwise} \end{cases} \tag{4}$$

$$\sigma^*(w) = \{uav : \exists u, v, \in V^*(w = uv)\}, \tag{5}$$

$$\sigma^r(w) = \{wa\},\ \sigma^l(w) = \{aw\}. \tag{6}$$

Symbol $\alpha \in \{*, l, r\}$ denotes the way of applying an insertion or a deletion rule to a word, namely, at any position ($a = *$), in the left-hand end ($a = l$), or in the right-hand end ($a = r$) of the word, respectively. Note that a substitution rule can be applied at any position. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the $\alpha - action$ of σ on L by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. For a given finite set of rules M, we define the $\alpha - action$ of M on a word w and on a language L by $M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

An evolutionary processor consists of a set of evolutionary operations and a filtering mechanism.

For two disjoint subsets P and F of an alphabet V and a word over V, predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ are defined as follows:

$$\varphi^{(1)}(w; P, F) \equiv P \subseteq alph(w) \wedge F \cap alph(w) = \emptyset$$

and

$$\varphi^{(2)}(w; P, F) \equiv alph(w) \cap P \neq \emptyset \wedge F \cap alph(w) = \emptyset.$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*).

For every language $L \subseteq V^*$ we define $\varphi^i(L,P,F) = \{w \in L \mid \varphi^i(w;P,F)\}$, $i = 1,2$.

An *evolutionary processor over* V is a 5-tuple (M, PI, FI, PO, FO) where:

- Either $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$. The set M represents the set of evolutionary rules of the processor. Notice that every processor is dedicated to only one type of the evolutionary operations.

- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor.

The set of evolutionary processors over V is denoted by EP_V.

Definition 1 *A* hybrid network of evolutionary processors *(an HNEP, shortly) is a 7-tuple* $\Gamma = (V, H, \mathcal{N}, C_0, \alpha, \beta, i_0)$*, where the following conditions hold:*

- *V is an alphabet, the alphabet of the network.*

- *$H = (X_H, E_H)$ is an undirected graph with set of vertices or nodes X_H and set of edges E_H. H is called the underlying graph of the network.*

- *$\mathcal{N} : X_H \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_H$ the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.*

- *$C_0 : X_H \longrightarrow 2^{V^*}$ is a mapping which identifies the initial configuration of the network. It associates a finite set of words with each node of the graph H.*

- *$\alpha : X_H \longrightarrow \{*, l, r\}$; $\alpha(x)$ defines the action mode of the rules performed in node x on the words occurring in that node.*

- *$\beta : X_H \longrightarrow \{(1),(2)\}$ defines the type of the input/output filters of a node. More precisely, for every node, $x \in X_H$, we define the following filters: the input filter is given as $\mu_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x)$, and the output filter is defined as $\tau_x(\cdot) = \varphi^{\beta(x)}(\cdot, PO_x, FO_x)$. That is, $\mu_x(w)$ (resp.τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. More generally, $\mu_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.*

- *$i_0 \in X_H$ is the output node of Γ.*

We say that $card(X_H)$ is the size of Γ. An HNEP is said to be a complete HNEP, if its underlying graph is a complete graph.

A configuration of an HNEP Γ, as above, is a mapping $C : X_H \longrightarrow 2^{V^*}$ which associates a set of words with each node of the graph. A component $C(x)$ of a configuration C is the set of words that can be found in the node x in this configuration, hence a configuration can be considered as the sets of words which are present in the nodes of the network at a given moment.

A configuration can change either by an evolutionary step or by a communication step. When it changes by an evolutionary step, then each component $C(x)$ of the configuration C is altered in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules, $\alpha(x)$. Formally, the configuration C' is obtained in one evolutionary step from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_H$.

When the configuration changes by a communication step, then each language processor $\mathcal{N}(x)$, where $x \in X_H$, sends a copy of its each word to every node processor where the node is connected with x provided that this word is able to pass the output filter of x, and receives all the words which are sent by processors of nodes connected with x provided that these words are able to pass the input filter of x. Those words which are not able to pass the respective output filter, remain at the node. Formally, we say that configuration C' is obtained in one communication step from configuration C, written as $C \vdash C'$, iff $C'(x) = (C(x) - \tau_x(C(x))) \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \mu_x(C(y)))$ holds for all $x \in X_H$.

For an HNEP Γ, the computation in Γ is a sequence of configurations C_0, C_1,$C_2, \ldots$, where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geq 0$.

HNEPs can be considered both language generating devices (generating hybrid networks of evolutionary processors or GHNEPs) and language accepting devices (accepting hybrid networks of evolutionary processors or AHNEPs).

In the case of GHNEPs we define the generated language as the set of all words which appear in the output node at some step of the

computation. Formally, the language generated by a generating hybrid network of evolutionary processors Γ is $L(\Gamma) = \bigcup_{s\geq 0} C_s(i_0)$.

In the case of AHNEPs, in addition to the components above, we distinguish an input alphabet and a network alphabet, V and U, where $V \subseteq U$, and instead of an initial configuration, we indicate an input node i_I. Thus, for an AHNEP, we use the notation $\Gamma = (V, U, H, \mathcal{N}, i_I, \alpha, \beta, i_0)$.

The computation by an AHNEP Γ for an input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}$, $C_1^{(w)}$,$C_2^{(w)}$,..., where $C_0^{(w)}$ is the initial configuration of Γ, with $C_0^{(w)}(i_I) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$, for $x \in G$, $x \neq i_I$, and $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$, $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i > 0$.

A computation as above is said to be accepting if there exists a configuration in which the set of words that can be found in the output node i_o is non-empty. The language accepted by Γ is defined by

$$L(\Gamma) = \{w \in V^* \mid \text{ the computation by } \Gamma \text{ on } w \text{ is an accepting one}\}.$$

3 Result

The following theorem states the regularity result for GHNEPs with one node. Although this has already been stated in [9], their proof is certainly incomplete. They stated that while GHNEPs without insertion only generate finite languages, GHNEPs with one insertion node only generate languages I^*C_0, C_0I^*, $C_0 \coprod I^*$, for the mode $l, r, *$, respectively. In the theorem below we present a precise characterization of languages generated by GHNEP with one node and consider the case omitted in [9] then the underlying graph G has a loop.

Theorem 1 *GHNEPs with one node only generate regular languages.*

Proof. As finite languages are regular, the statement holds for GHNEPs without insertion nodes. We now proceed with the case of one insertion node. Consider such a GHNEP $\Gamma = (V, G, N_1, C_0, \alpha, \beta, 1)$, where

$$N_1 = (M, PI, FI, PO, FO).$$

Let us introduce a few notations. Inserting a symbol from I in a language C yields a language $\mathsf{ins}_I(C)$. Depending on whether $\alpha = l$, $\alpha = r$ or $\alpha = *$, $\mathsf{ins}_I(C)$ is one of IC, CI, $C \coprod I$, respectively. For inserting an arbitrary number of symbols from a set I in a language C, $\mathsf{ins}_I^*(C)$ is one of I^*C, CI^*, $C \coprod I^*$. Clearly, ins_I^* preserves regularity.

We denote the set of symbols inserted in N_1 by $I = \{a \mid \lambda \to a \in M\}$. The configuration of N_1 after one step is $C_1 = \mathsf{ins}_I(C_0)$. Assume that $\beta = 2$ (a case then $\beta = 1$ can be considered analogously), then the conditions of passing permitting and forbidding output filter can be specified by regular languages $\pi = V^*POV^*$ and $\varphi = (V - FO)^*$, respectively. For instance, the set of words of C_1 that pass the forbidding output filter but do not pass the forbidding input filter is $C_1' = C_1 \cap \varphi \setminus \pi$. Notice that inserting symbols that belong to neither PO nor FO does not change the behavior of the filters; we denote the corresponding language by $B = \mathsf{ins}^*_{I\setminus(PO\cup FO)}(C_1)$.

Consider the case when the graph G consists of one node and no edges. Then, Γ generates the following language

$$\begin{aligned} L_1 = L_1(\Gamma) &= C_0 \cup C_1 \cup \mathsf{ins}_I^*(C_1 \setminus \varphi) \cup B \\ &\quad \cup \mathsf{ins}_{I\cap PO\setminus FO}(B) \cup \mathsf{ins}_I^*(\mathsf{ins}_{I\cap FO}(B)), \qquad (7) \\ B &= \mathsf{ins}^*_{I\setminus(PO\cup FO)}(C_1), \\ C_1 &= \mathsf{ins}_I(C_0). \end{aligned}$$

Indeed, this is a union of six languages:

1. initial configuration,
2. configuration after one insertion,
3. all words that can be obtained from a word from C_1 if it is trapped in N_1 by the forbidding filter,
4. B represents the words that pass the forbidding filter but not the permitting filter,
5. words obtained by inserting one permitting and not forbidden symbol into B, and

6. words obtained by inserting one forbidden symbol into B, and then by arbitrary insertions.

Consider the case when the graph G has a loop. The set of words leaving the node (for the first time) is $D = (C_1 \cap \varphi \cap \pi) \cap \mathsf{ins}_{I \cap PO \setminus FO}(B)$. The conditions of the permitting and forbidding input filters can be specified by regular languages $\pi' = V^*PIV^*$ and $\varphi' = (V - FI)^*$, respectively. Some of words from D return to N_1, namely $D \cap \pi' \cap \varphi'$. Notice that further insertion of symbols that belong neither to FO nor to FI causes the words to continuously exit and reenter N_1. The associated language is $B' = \mathsf{ins}^*_{I \setminus (FO \cup FI)}(D \cap \pi' \cap \varphi')$. Finally, we give the complete presentation of the language generated by Γ in this case:

$$\begin{aligned} L_1' = L_1(\Gamma) &= L_1 \cup B' \cup \mathsf{ins}^*_I(\mathsf{ins}_{I \cap FO}(B')) \cup \mathsf{ins}_{I \cap FI \setminus FO}(B'), \quad (8) \\ B' &= \mathsf{ins}^*_{I \setminus (FO \cup FI)}(D \cap \pi' \cap \varphi'), \\ D &= (C_1 \cap \varphi \cap \pi) \cap \mathsf{ins}_{I \cap PO \setminus FO}(B), \\ C_1 &= \mathsf{ins}_I(C_0). \end{aligned}$$

Indeed, this is a union of four languages:

1. words that never reenter N_1, as in the case when G has no edges,
2. B' represents the words that once leave and reenter N_1, and keep doing so after subsequent insertions,
3. words obtained by inserting a symbol from FO into B', and then by arbitrary insertions,
4. words obtained by inserting a symbol from $FI \setminus FO$ into B'.

□

4 Conclusion

In this paper we presented a precise form of the languages generated by an HNEP with one node and considered one case omitted in the previous proof. Thus we completed investigation of this class of HNEPs.

Tasks of characterization of languages generated by a GHNEP with two nodes and languages accepting by an AHNEP with two nodes are still opened.

References

[1] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. *About Universal Hybrid Networks of Evolutionary Processors of Small Size*, in: Pre-Proceedings of the 2nd International Conference on Language and Automata Theory and Applications, LATA 2008, GRLMC report 36/08, University Rovira i Virgili, Tarragona, 2008 pp. 43-54. Also in: Lecture Notes in Computer Science 5196, Springer, 2008, 28–39.

[2] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. *Computational Completeness of Hybrid Networks of Evolutionary Processors with Seven Nodes*, in: C. Campeanu, G. Pighizzini (Eds.), Descriptional Complexity of Formal Systems, DCFS 2008 Proceedings, University of Prince Edward Island, Charlottetown, 2008, pp. 38–47.

[3] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. *On the Size of Computationally Complete Hybrid Networks of Evolutionary Processors*, 2008 (submitted).

[4] A. Alhazov, M. Kudlek, Yu. Rogozhin. *Nine Universal Circular Post Machines*, Computer Science Journal of Moldova 10(3) (2002) 247–262.

[5] A. Alhazov, C. Martín-Vide, Yu. Rogozhin. *On the number of nodes in universal networks of evolutionary processors.* Acta Informatica 43(5) (2006) 331–339.

[6] A. Alhazov, C. Martín-Vide, Yu. Rogozhin. *Networks of Evolutionary Processors with Two Nodes Are Unpredictable*, in: Pre-Proceedings of the 1st International Conference on Language and

Automata Theory and Applications, LATA 2007, GRLMC report 35/07, Rovira i Virgili University, Tarragona, 2007, pp. 521–528.

[7] J. Castellanos, P. Leupold, V. Mitrana. *On the size complexity of hybrid networks of evolutionary processors*, Theoretical Computer Science 330(2) (2005) 205–220.

[8] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere. *Solving NP-complete problems with networks of evolutionary processors.* in: J. Mira, A. Prieto (Eds.), IWANN 2001, Lecture Notes in Computer Science 2084, Springer, 2001, pp. 621–628.

[9] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana. *Hybrid networks of evolutionary processors are computationally complete.* Acta Informatica 41(4-5) (2005) 257–272.

[10] J. Dassow, B. Truthe. *On the Power of Networks of Evolutionary Processors*, in: J. O. Durand-Lose, M. Margenstern (Eds.), MCU 2007, Lecture Notes in Computer Science 4667, Springer, 2007, pp. 158–169.

[11] M. Kudlek, Yu. Rogozhin. *Small Universal Circular Post Machines*, Computer Science Journal of Moldova 9(1) (2001) 34–52.

[12] M. Kudlek, Yu. Rogozhin. *New Small Universal Circular Post Machines*, in: R. Freivalds (Ed.), Proc. FCT 2001, Lecture Notes in Computer Science 2138, Springer, 2001, pp. 217–227.

[13] F. Manea, C. Martín-Vide, V. Mitrana. *On the size complexity of universal accepting hybrid networks of evolutionary processors*, Mathematical Structures in Computer Science 17(4) (2007) 753–771.

[14] F. Manea, C. Martín-Vide, V. Mitrana. *All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size*, Information Processing Letters 103(2007) 112–118.

[15] M. Margenstern, V. Mitrana, M.-J. Pérez-Jiménez. *Accepting Hybrid Networks of Evolutionary Processors*, in: C. Ferretti, G. Mauri, C. Zandron (Eds.), DNA 10, Lecture Notes in Computer Science 3384, Springer, 2005, pp. 235–246.

[16] C. Martín-Vide, V. Mitrana, M. Pérez-Jiménez, F. Sancho-Caparrini. *Hybrid networks of evolutionary processors*, in: E. Cantú-Paz et al. (Eds.), Proc. of GECCO 2003, Lecture Notes in Computer Science 2723, Springer, 2003, 401–412.

[17] A. Salomaa. *Formal Languages*, Academic Press, New York, 1973.

Artiom Alhazov, Yurii Rogozhin, Received October 31, 2008

Dr. Artiom Alhazov
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5 Academiei str., Chişinău, MD-2028, Moldova
email: *artiom@math.md*

Dr.hab. Yurii Rogozhin
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5 Academiei str., Chişinău, MD-2028, Moldova
email: *rogozhin@math.md*
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl.Imperial Tarraco, 1, Tarragona, Spain