Computer Science Journal of Moldova, vol.**21**, no.1(61), 2013

Developments in Networks of Evolutionary Processors *

Artiom Alhazov

Abstract

Networks of evolutionary processors (NEPs) are distributed word rewriting systems typically viewed as language generators. Each node contains a set of words, a set of operations (typically insertion, deletion or rewriting of one symbol with another one), an input filter and an output filter. The purpose of this paper is to overview existing models of NEPs, their variants and developments.

In particular, besides the basic model, hybrid networks of evolutionary processors (HNEPs) have been extensively studied. In HNEPs, operations application might be restricted to specific end of the string, but the filters are random-context conditions (they were regular in the basic model). We will also cover the literature on the so-called obligatory HNEPs, i.e., ones where the operations are obligatory: the string that cannot be rewritten is not preserved.

Some specific aspects that we pay attention to are: computational universality and completeness, the topology of the underlying graph, the number of nodes, the power of filters.

1 Introduction

Insertion, deletion, and substitution are fundamental operations in formal language theory, their power and limits have obtained much at-

*The author gratefully acknowledges project STCU-5384 "Models of high performance computations based on biological and quantum approaches", and project ref. nr. 12.819.18.09A "Development of IT support for interoperability of electronic linguistic resources" from Supreme Council for Science and Technological Development of the Republic of Moldova.

tention 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 [17], 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 [12, 13]).

1.1 Basic model

Motivated by some models of massively parallel computer architectures, networks of language processors have been introduced in [19]. Such a network can be considered as a graph, where the nodes are sets of productions and at any moment of time a language is associated with a node. In a derivation step, any node derives from its language all possible words as its new language. In a communication step, any node sends those words to other nodes that satisfy an output condition given as a regular language, and any node takes those words sent by the other nodes that satisfy an input condition also given by a regular language. The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, a special type of networks of language processors was introduced in [17], called networks with evolutionary processors, because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found e. g. in [17], [16], [15], [12]. In [16] it was shown that networks of evolutionary processors are universal in that sense that they can generate any recursively enumerable language, and that networks with six nodes are sufficient to get all recursively enumerable languages. In [12] the latter result has been improved by showing that networks with three nodes are sufficient.

In [12] one presents the proof of the computational completeness with two nodes, additionally employing a morphism. In [9] one shows that NEPs with two nodes (one insertion node and one deletion node) generate all recursively enumerable languages (in intersection with a monoid), avoiding the need for a morphism. The same paper shows that insertion and substitution characterize context-sensitive languages, while deletion and substitution characterize finite languages.

1.2 Hybrid model

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 [27] and the concept of an AHNEP was defined in [26].

In [18] 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 [6], where it was proved that GHNEPs with 10 nodes (irrespectively of the size of the alphabet) obtain the universal power. For accepting HNEPs, in [24] it was shown that for any recursively enumerable language there exists a recognizing AHNEP with 31 nodes; the result was improved in [25] where the number of necessary nodes was reduced to 24. Furthermore, in [25] 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 [7] it was proved that any recursively enumerable language can be generated by a GHNEP having 7 nodes (thus, the result from [6] is improved) and in [8] the same authors showed that any recursively enumerable language can be accepted by an AHNEP with 7 nodes (thus, the result from [25] is improved significantly). An improvement of the accepting result to 6 nodes has been obtained in [23], by simulating Tag systems. In [8] also it was showed that the families of GHNEPs and AHNEPs with 2 nodes are not computationally complete.

In [18] it was demonstrated that a GHNEP with one node can generate only regular language, while in [14] a precise form of the generated language was presented, also considering 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 open.

1.3 Obligatory operations

A variant of HNEPs, called Obligatory HNEPs (OHNEP for short) was introduced in [3]. The differences between HNEP and OHNEP are the following:

1. in deletion and substitution: a node discards a string if no operations in the node are applicable to the string (in HNEP case, this string remains in the node),

2. the underlying graph is a directed graph (in HNEP case, this graph is undirected); this second difference disappears when we consider complete networks.

These differences make OHNEPs universal [3] with 1 operation per node, no filters and only left insertion and right deletion.

In [5] complete OHNEPs were considered, i.e., OHNEPs with complete underlying graph. One may now regard complete OHNEP as a set of very simple evolutionary processors "swimming in the environment" (i.e., once a string leaves a node, it is not essential for the rest of the computation which node it left). In [5] it is proved that the complete OHNEPs with very simple evolutionary processors, i.e., evolutionary processors with only one operation (obligatory deletion, obligatory substitution and insertion) and filters containing not more than 3 symbols are computationally complete. We recall that the filters are either single symbols or empty sets, while the **sum** of weights has been counted.

In [4] one considers OHNEPs without substitution. It is not difficult to notice that in complete OHNEPs without substitution there is no control on the number of insertion or deletion of terminal symbols (i.e., those symbols which appear in output words). Therefore, the definition of OHNEPs needed to be modified in order to increase their computational power. In [4] one shows that it is possible to avoid substitution using modified operations of insertion and deletion in evolutionary processors similar to "matrix" rules in formal language theory. By using such techniques a small universal complete OHNEP with 182 nodes without substitution is constructed.

Several open questions were posed in [4], in particular the question about the minimal total complexity of filters of evolutionary processor in computationally complete OHNEPs and the question about universal complete OHNEP without substitution with the minimal number of nodes. In [1] one considers a model of OHNEP allowing the use of all three molecular operations: insertion, deletion and substitution, and provides a very unexpected result. OHNEPs are computationally complete even if the total power of the filters of each node does not exceed 1! This means that in any node, all four filters are empty, except possibly one, being a single symbol.

In the following we describe selected results in details.

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 $x \in \Sigma^*$.

For a word $u \in V^*$, we define the sets of proper prefixes, proper suffixes and non-empty suffixes of u by

$$\begin{aligned} PPref(u) &= \{x \mid u = xy, |y| \geq 1\}, \\ PSuf(u) &= \{y \mid u = xy, |x| \geq 1\}, \\ NSuf(u) &= \{y \mid u = xy, |y| \geq 1\}, \text{ respectively.} \end{aligned}$$

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

$$\begin{aligned} ⧢(x,y) = \{x_1y_1x_2y_2 \dots x_ny_n \mid n \geq 1,\ x_i, y_j \in V^*, \\ x = x_1x_2 \dots x_n,\ y = y_1y_2 \dots y_n\}. \end{aligned}$$

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

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

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 $x = x_1ux_2$ and $y = x_1vx_2$, $x_1, x_2 \in (N \cup T)^*$ 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* [28]. 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 $L(G') = L(G)$.

A type-0 generative grammar $G = (N, T, S, P)$ is in Kuroda normal form if every rule in P is one of the following forms: $A \longrightarrow a$, $A \longrightarrow \varepsilon$, $A \longrightarrow BC$, $AB \longrightarrow CD$, where $A, B, C, D \in N$ and $a \in T$.

Analogously, we can say that a type-0 analytic grammar $G = (N, T, S, P)$ is in Kuroda-like normal form if every production in P is one of the following forms: $a \longrightarrow A$, $\varepsilon \longrightarrow A$, $AB \longrightarrow C$, $AB \longrightarrow CD$, where $A, B, C, D \in N$ and $a \in T$.

It is well-known that the type-0 generative grammars in Kuroda normal form determine the class of recursively enumerable languages and it can immediately be seen that the same statement holds for the type-0 analytic grammars in Kuroda-like normal form.

In the sequel, following the terminology in [18], 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.

2.1 Circular Post machines

Circular Post Machines (CPMs) were introduced in [21], where it was shown that all introduced variants of CPMs are computationally complete, and moreover, the same statement holds for CPMs with two symbols. In [22], [10] several universal CPMs of variant 0 (CPM0) having small size were constructed[1], among them in [10] a universal CPM0 with 6 states and 6 symbols. In this article we use the deterministic variant of CPM0s.

A *Circular Post Machine* is a quintuple $(\Sigma, Q, \mathbf{q}_0, \mathbf{q}_f, R)$ with a finite alphabet Σ, where 0 is the blank, a finite set of states Q, the initial state $\mathbf{q}_0 \in Q$, the final state $\mathbf{q}_f \in Q$, and a finite set of instructions R with all instructions having one of the forms $\mathbf{p}x \to \mathbf{q}$ (erasing the symbol read by deleting a cell), $\mathbf{p}x \to y\mathbf{q}$ (overwriting and moving to the right), $\mathbf{p}0 \to y\mathbf{q}0$ (overwriting and creating a blank cell), where $x, y \in \Sigma$ and $\mathbf{p}, \mathbf{q} \in Q$, $\mathbf{p} \neq \mathbf{q}_f$. We also refer to all instructions with $\mathbf{q}_f$ in the left hand side as halt instructions.

The storage of this machine is a circular tape, the read and write head moves only in one direction (to the right), and with the possibility to delete a cell or to create and insert a new cell with a blank.

2.2 Evolutionary processors

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

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

[1]Other variants of CPMs use slightly different instruction sets, which may make a difference for the size of small universal machines.

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

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

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

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

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

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.

2.3 Hybrid networks

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 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 the evolutionary processor* $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$ *with each node* $x \in X_H$*.*

- $C_0 : X_H \longrightarrow 2^{V^*}$ *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 $C'(x) = M_x^{\alpha(x)}(C(x)) \quad \text{for all } x \in X_H$.

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 $C'(x) = (C(x) - \tau_x(C(x))) \bigcup_{\{x,y\} \in E_H} (\tau_y(C(y)) \cap \mu_x(C(y)))$ holds for all $x \in X_H$.

2.4 Computation and result

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 $C_0^{(w)}$, $C_1^{(w)}$,$C_2^{(w)}, \ldots$, 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

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

2.5 Obligatory networks

The model of OHNEPs is obtained from the model of HNEPs by excluding the second case, i.e., "$\{w\}$,otherwise", from (1)-(4). Hence, for a string to remain in a node, it is *obligatory* for it to evolve via some rule from Sub_V, Del_V or Ins_V.

Notice that the definition of OHNEPs is thus simpler and more uniform than that of HNEPs. In the same time, using the power of the underlying graph, it makes it possible to even reach the computational completeness with nodes only having one operation, and without filters, [3].

2.6 Basic model of NEPs

The concept of NEPs is simpler than that of HNEPs. We find it suitable here to define the former in terms of the latter, by modifying the following:

- Only the $*$ mode exists for evolutionary processors.
- A permitting filter and a forbidden filter are combined into one filter, which may be any regular language. A string passes the

filter iff it belongs to the corresponding regular language. The filtering mode β loses its meaning.

3 Selected results

We would like to point out that there is no interaction, direct or indirect, between the words of the network. Hence, the generated language is a union of languages, generated by the same system, but starting with only one word.

As for the replication, i.e., the possibility of applying multiple rules or the same rule in multiple ways, producing many words from one word, this could be viewed as a non-deterministic evolution of *one* word. In this case, distributivity simply means assigning a state to the word. Summing up, the language generated by a parallel deterministic word rewriting system may be viewed as a (union of) language(s) generated by a non-deterministic one-word rewriting system with states (without any other parallelism or distributivity).

Furthermore, the nature of the model (except the obligatory variant) leads to many cases of the "shadow" computations, in the following sense. If one carefully considers the definitions, and constructs a faithful simulation of the model, one would notice that a lot of computation in the system consists of repeatedly recomputing the same steps. This is due to the fact that if some operation $\pi \in Sub_V$ or $\rho \in Del_V$ of a node is not applicable to some word w in that node, the result is w. Taking the union over all operations of a node yields a set containing w, even if some other operation was applicable to w. Clearly, in the next step, the words obtainable from w in the same node will be recomputed. However, the system is deterministic, so nothing *new* is obtained in this way.

A careful examination reveals that, in some circumstances, the shadow computations can be avoided, modifying the definition while yielding the same generated language! Indeed, preserving w is useless (everything that is possible to derive from w in that node is derived immediately) unless w exits the node. However, at least in the case of complete networks, if w enters a node and exits it unchanged, this does not do anything new either (if w is an initial word, it can be copied to all nodes that it can reach unchanged in communication step).

The above reason lets us claim that, e.g., any result for complete OHNEPs holds also for the usual complete HNEPs, and the associated computational burden of the simulation may be greatly reduced. If the network is not complete, then a heuristic still may be used by a simulator, by preserving unchanged words only in case if they actually move from a node into some different node.

3.1 NEPs with two nodes

Theorem 1 *For any recursively enumerable language L, there are a set T and a network N of evolutionary processors with exactly one insertion node and exactly one deletion node such that $L = L(N) \cap T^*$.[9]*

Proof. (sketch) We consider a type-0 grammar $G = (N, T, P, S)$ with $L(G) = L$. Then all rules of P have the form $u \to v$ with $u \in N^+$ and $v \in (N \cup T)^*$. Let $X = N \cup T, X' = \{a, a' \mid a \in N \cup T\}$, $T' = \{a, a' \mid a \in T\}$ and $P' = \{p_i \mid p \in P, 1 \leq i \leq 4\}$. We define a morphism $\mu : X^* \to (X')^*$ by $\mu(a) = aa'$ for $a \in X$ and set $W = \{\mu(w) \mid w \in X^*\}$. We construct the following network $\mathcal{N} = (V, (M_1, A_1, I_1, O_1), (M_2, A_2, I_2, O_2), E, 2)$ of evolutionary processors with

$$\begin{aligned}
V &= P' \cup X', \\
M_1 &= \{\lambda \to p_i \mid p_i \in P',\ 1 \leq i \leq 4\} \cup \{\lambda \to a \mid a \in X'\}, \\
A_1 &= \{\mu(S)\}, \\
I_1 &= W \setminus (T')^*, \\
O_1 &= V * \setminus (W R_{1,1} W), \\
M_2 &= \{p_i \to \lambda \mid p_i \in P',\ 1 \leq i \leq 4\} \cup \{a \to \lambda \mid a \in X' \setminus T\}, \\
A_2 &= \emptyset, \\
I_2 &= W R_{1,2} W, \\
O_2 &= V^* \setminus (W R_{2,2} W \cup (T')^*),
\end{aligned}$$

$$E = \{(1,2),(2,1)\}, \text{ where}$$

$$\begin{aligned}
R_{1,1} &= \bigcup_{p:u\to v\in P} (\{p_1\mu(u), p_1\mu(u)p_3, p_1p_2\mu(u)p_3, p_1p_2\mu(u)p_3p_4\} \\
&\cup \{p_1p_2\mu(u)\}PPref(\mu(v))\{p_3p_4\}) \\
&\quad \setminus\{p_1p_2\mu(u)p_3p_4 \mid p : u \to v \in P\}, \\
R_{1,2} &= \{p_1p_2\mu(uv)p_3p_4 \mid p : u \to v \in P\}, \\
R_{2,2} &= \bigcup_{p:u\to v\in P} (\{p_1p_2\}PSuf(\mu(u))\{\mu(v)p_3p_4\} \\
&\cup \{p_2\mu(v)p_3p_4, p_2\mu(v)p_4, \mu(v)p_4\}).
\end{aligned}$$

The output and input filters are defined in order to remove the garbage and communicate the strings that should change the type of operation, keeping only the strings that should continue to evolve by operations of the same type. Since the morphism $\mu(a) = aa'$ is introduced, the strings obtained by applying rules to the left or to the right of the place of application of the current rule are no longer kept in the node by the filter, and are not accepted by either node (recall that $W = aa'|a \in \{N \cup T^*)$, so they leave the system. Claim: $L(\mathcal{N}) \setminus T^* = L$. The correct simulation of an application of a production $p : a_1 \cdots a_s \to b_1 \cdots b_t$ to a sentential form $\alpha a_1 \cdots a_s\beta$ and with $x = \mu(\alpha)$ and $y = \mu(\alpha)$ has the following form. In N_1 we have

$$\begin{aligned}
xa_1a_1' \cdots a_sa_s'y &\Rightarrow^{\lambda\to p_1} xp_1a_1a_1' \cdots a_sa_s'y \\
&\Rightarrow^{\lambda\to p_3} xp_1a_1a_1' \cdots a_sa_s'p_3y \\
&\Rightarrow^{\lambda\to p_2} xp_1p_2a_1a_1' \cdots a_sa_s'p_3y \\
&\Rightarrow^{\lambda\to p_4} xp_1p_2a_1a_1' \cdots a_sa_s'p_3p_4y \\
&\Rightarrow^{\lambda\to b_1} xp_1p_2a_1a_1' \cdots a_sa_s'b_1p_3p_4y \\
&\Rightarrow^{\lambda\to b_1'} xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1'p_3p_4y \\
&\Rightarrow^{*} xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1' \cdots b_tp_3p_4y \\
&\Rightarrow^{\lambda\to b_t'} xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y
\end{aligned}$$

and in N_2 we have

$$\begin{aligned}
& xp_1p_2a_1a_1' \cdots a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{a_1\to\lambda}\ & xp_1p_2a_1' \cdots a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{a_1'\to\lambda}\ & xp_1p_2a_2 \cdots a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{*}\ & xp_1p_2a_sa_s'b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{a_s\to\lambda}\ & xp_1p_2a_s'b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{a_s'\to\lambda}\ & xp_1p_2b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{p_1\to\lambda}\ & xp_2b_1b_1' \cdots b_tb_t'p_3p_4y \\
\Rightarrow^{p_3\to\lambda}\ & xp_2b_1b_1' \cdots b_tb_t'p_4y \\
\Rightarrow^{p_2\to\lambda}\ & xb_1b_1' \cdots b_tb_t'p_4y \\
\Rightarrow^{p_4\to\lambda}\ & xb_1b_1' \cdots b_tb_t'y.
\end{aligned}$$

Notice that if a production can be applied to the same sentential form in different ways (multiple productions and/or multiple places to apply them), then the corresponding number of strings is produced in the first step (inserting marker p_1 associated to the production, to the left of the application place). The rest of the simulation is deterministic in the following sense: starting from $xp_1a_1a_1' \cdots a_sa_s'y$, the result $xb_1b_1' \cdots b_tb_t'y$ is obtained according to the derivations above, while all other strings are discarded. The strings that leave one node and enter another one belong to the sets $O_1 \setminus I_2 = I_2$ and $O_2 \setminus I_1 = I_1$. All other strings that leave a node do not enter anywhere. With $p \in P$, $a \in X'$ and $A \in X' \setminus T$, the following tables illustrate the behaviour of a string (the numbers give the situation which is obtained by using the rule in question and n/a refers to nonapplicability of the rule).

n	Shape in $N1$	$\lambda \to p_1$	$\lambda \to p_3$	$\lambda \to p_2$	$\lambda \to p_4$	$\lambda \to a$
1	W	2	out	out	out	out
2	$Wp_1\mu(u)W$out	3	out	out	out	
3	$Wp_1\mu(u)p_3W$	out	out	4	out	out
4	$Wp_1p_2\mu(u)p_3W$	out	out	out	5	out
5	$Wp_1p_2\mu(u)\cdot$ $PPref(\mu(v))p_3p_4W$	out	out	out	out	5,6

n	Shape in $N1$	$p_1 \to \lambda$	$p_3 \to \lambda$	$p_2 \to \lambda$	$p_4 \to \lambda$	$A \to \lambda$
6	$Wp_1p_2\cdot$ $NSuf(\mu(u))\mu(v)p_3p_4W$	out	out	out	out	6,7
7	$Wp_1p_2\mu(v)p_3p_4W$	8	out	out	out	out
8	$Wp_2\mu(v)p_3p_4W$	n/a	9	out	out	out
9	$Wp_2\mu(v)p_4W$	n/a	n/a	10	out	out
10	$W\mu(v)p_4W$	n/a	n/a	n/a	1,11	out
11	$(T')^*$	n/a	n/a	n/a	n/a	11

These tables illustrate the fact that if a symbol is inserted or deleted in a way that does not follow the "correct" simulation, than the string leaves the system. Finally, consider $L(\mathcal{N}) \cap (T')^*$. It is the set of all strings obtained in N_2 without nonterminal symbols, without markers and without pre-terminals (i. e., primed versions of terminals). Hence, all of them are obtained from shape 5 of N_2 by deleting the marker p_4, reaching shape 6 if the string only has terminals and pre-terminals. It is easy to see that in several computation steps all pre-terminal symbols will be deleted. This exactly corresponds to the set of terminal strings w produced by the underlying grammar G, all letters being represented by a double repetition, i. e., encoded by μ. Such strings remain in N_2 and all pre-terminals are deleted, obtaining w from $\mu(w)$. □

3.2 HNEPs with one node

The following theorem states the regularity result for GHNEPs with one node. Although this has already been stated in [18], 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 [18], when the underlying graph G has a loop.

Theorem 2 *One-node GHNEPs only generate regular languages.[14]*

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, when $\beta = 1$, 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 reeenter 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'.

□

3.3 HNEPs with 7 nodes

Theorem 3 *Any recursively enumerable language can be generated by a complete HNEP of size 7. [7, 8]*

Proof. Let $L \subseteq T^*$ be a language generated by a type-0 grammar $G = (N, T, S, P)$ in Kuroda normal form.

We construct a complete HNEP $\Gamma = (V, H, \mathcal{N}, C_0, \alpha, \beta, 7)$ of size 7 which simulates the derivations in G and only that, by using the so-called rotate-and-simulate method. The rotate-and-simulate method means that the words in the nodes are involved in either the rotation of their leftmost symbol (the leftmost symbol of the word is moved to the end of the word) or the simulation of a rule of P. In order to indicate the end of the word when rotating its symbols and thus to guarantee the correct simulation, a marker symbol, #, different from any element of $(N \cup T)$ is introduced. Let $N \cup T \cup \{\#\} = A = \{A_1, A_2, \dots A_n\}$, $I = \{1, 2, \dots, n\}$, $I' = \{1, 2, \dots, n-1\}$, $I'' = \{2, 3 \dots, n\}$, $I_0 = \{0, 1, 2, \dots, n\}$, $I'_0 = \{0, 1, 2, \dots, n-1\}$, $B_0 = \{B_{j,0} \mid j \in I\}$, $B'_0 = \{B'_{j,0} \mid j \in I\}$, $\# = A_n$, $T' = T \cup \#$. Let us define the alphabet V of Γ as follows:

$$\begin{aligned}
V &= A \cup B \cup B' \cup C \cup C' \cup D \cup D' \cup E \cup E' \cup F \cup G \cup \{\varepsilon'\},\\
B &= \{B_{i,j} \mid i \in I,\ j \in I_0\}, B' = \{B'_{i,j} \mid i \in I, j \in I_0\},\\
C &= \{C_i \mid i \in I\}, C' = \{C'_i \mid i \in I\}, D = \{D_i \mid i \in I_0\},\\
D' &= \{D'_i \mid i \in I\},\\
E &= \{E_{i,j} \mid i, j \in I\}, E' = \{E'_{i,j} \mid i, j \in I\},\\
F &= \{F_j \mid j \in I\}, G = \{G_{i,j} \mid i, j \in I\}.
\end{aligned}$$

Let H be a complete graph with 7 nodes, let $\mathcal{N}, C_0, \alpha, \beta$ be presented in Table 1, and let node 7 be the output node of HNEP Γ.

A sentential form (a configuration) of grammar G is a word $w \in (N \cup T)^*$. When simulating the derivations in G, each sentential form w of G corresponds to a string of Γ in node 1 and having one of the forms $wB_{n,0}$ or $w''A_nw'B_{i,0}$, where $A_n = \#$, $w, w', w'' \in (N \cup T)^*$ and $w = w'A_iw''$. The start symbol $S = A_1$ of G corresponds to an initial word $A_1\#$, represented as $A_1B_{n,0}$ in node 1 of HNEP Γ, the other nodes do not contain any word. The simulation of the application of a rule of G to a substring of a sentential form of G is done in several evolution and communication steps in Γ, through rewriting the leftmost symbol and the two rightmost or the rightmost symbol of strings. This is the reason why we need the symbols to be rotated.

In the following we describe how the rotation of a symbol and the application of an arbitrary rule of grammar G are simulated in HNEP Γ.

Rotation.

Let $A_{i_1}A_{i_2}\dots A_{i_{k-1}}B_{i_k,0} = A_{i_1}wB_{i_k,0}$ be a word found at node 1, and let $w, w', w'' \in A^*$. Then, by applying rule `1.1` we obtain $A_{i_1}A_{i_2}\dots A_{i_{k-1}}B_{i_k,0} = A_{i_1}wB_{i_k,0} \xrightarrow{1.1} \{C'_{i_1}wB_{i_k,0},\ A_{i_1}w'C'_{i_t}w''B_{i_k,0}\}$.

We note that during the simulation symbols C'_i should be transformed to ε', and this symbol can only be deleted from the left-hand end of the string (node 6). So, the replacement of C_{i_t} by its primed version in a string of the form $A_{i_1}w'C_{i_t}w''B_{i_k,0}$ results in a word that will stay in node 6 forever; thus, in the sequel, we will not consider strings with C'_i not in the leftmost position. In the communication step following the above evolution step, string $C'_{i_1}wB_{i_k,0}$ cannot leave node 1 and stays there for the next evolution step:

$$C'_{i_1}wB_{i_k,0} \xrightarrow{1.5} C'_{i_1}wB'_{i_k,0}.$$

Observe that rules `1.1` and `1.5` may be applied in any order. After then, string $C'_{i_1}wB'_{i_k,0}$ can leave node 1 and can enter only node 2. In the following steps of the computation, in nodes 1 and 2, the string is involved in evolution steps followed by communication:

$$C_{i_1-t}wB_{i_k,t} \xrightarrow{1.4} C'_{i_1-(t+1)}wB_{i_k,t} \xrightarrow{1.6} C'_{i_1-(t+1)}wB'_{i_k,t+1} \text{ (in node 1)},$$

$$C'_{i_1-t}wB'_{i_k,t} \xrightarrow{2.1} C_{i_1-(t+1)}wB'_{i_k,t} \xrightarrow{2.2} C_{i_1-(t+1)}wB_{i_k,t+1} \text{ (in node 2)}.$$

$N, \alpha, \beta, C_0,$	M, PI, FI, PO, FO
$1, *, (2),$ $\{A_1 B_{n,0}\}$	$\{\mathbf{1.1}: A_i \to C'_i \mid i \in I\} \ \cup \ \{\mathbf{1.2}: A_i \to \varepsilon' \mid i \in I',\ A_i \to \varepsilon\} \cup$ $\{\mathbf{1.3}: B_{j,0} \to B_{s,0} \mid A_j \to A_s,\ j, s \in I'\}$ $\{\mathbf{1.4}: C_i \to C'_{i-1},\ \mathbf{1.5}: B_{j,0} \to B'_{j,0},$ $\mathbf{1.6}: B_{j,k} \to B'_{j,k+1} \mid i \in I'',\ j \in I,\ k \in I'\} \cup$ $\{\mathbf{1.7}: C_1 \to \varepsilon'\} \ \cup \ \{\mathbf{1.8}: E'_{j,k} \to E_{j,k-1},\ \mathbf{1.9}: D'_i \to D_{i+1},$ $\mathbf{1.10}: E'_{j,1} \to F_j \mid i \in I', j \in I, k \in I''\}$ $PI = \{A_n, B_{n,0}\} \cup C \cup E'$ $FI = C' \cup E \cup D \cup F \cup G \cup \{\varepsilon'\}$ $PO = C' \cup B' \cup D \cup F \cup \{\varepsilon'\}$ $FO = B \cup C \cup D' \cup E'$
$2, *, (2), \emptyset$	$\{\mathbf{2.1}: C'_i \to C_{i-1},\ \mathbf{2.2}: B'_{j,k} \to B_{j,k+1} \mid i \in I'',\ j \in I,\ k \in I'_0\} \cup$ $\{\mathbf{2.3}: C'_1 \to \varepsilon'\} \ \cup \ \{\mathbf{2.4}: E_{j,k} \to E'_{j,k-1},\ \mathbf{2.5}: D_i \to D'_{i+1},$ $\mathbf{2.6}: E_{j,1} \to F_j \mid i \in I'_0, j \in I, k \in I''\} \ \cup \ \{\mathbf{2.7}: A_n \to \varepsilon'\} \cup$ $\{\mathbf{2.8}: B_{j,0} \to A_j \mid A_j \in T\}$ $PI = \{B_{j,0} \mid A_j \in T\} \cup C' \cup E$ $FI = \{B \setminus B_{j,0} \mid A_j \in T\} \cup C \cup \ D' \cup E' \cup F \cup G \cup \{\varepsilon'\}$ $PO = C \cup D' \cup F \cup \{\varepsilon'\}$ $FO = \{B_{j,0} \mid A_j \in T\} \cup B' \cup C' \cup \ E \cup D$
$3, r, (2), \emptyset$	$\{\mathbf{3.1}: \varepsilon \to D_0\}$ $PI = B \setminus B_0 \cup B' \setminus B'_0 \cup G$ $FI = C \cup C' \cup B_0 \cup \{D_0\},\ PO = \{D_0\},\ FO = \emptyset$
$4, *, (2), \emptyset$	$\{\mathbf{4.1}: B_{j,k} \to E_{j,k},\ \mathbf{4.2}: B'_{j,k} \to E_{j,k} \mid j, k \in I\} \cup$ $\{\mathbf{4.3}: B_{j,k} \to E_{s,t},$ $\mathbf{4.4}: B'_{j,k} \to E_{s,t} \mid j, k, s, t \in I', A_j A_k \to A_s A_t\} \cup$ $\{\mathbf{4.5}: G_{j,k} \to E_{j,k} \mid j, k \in I'\}$ $PI = \{D_0\},\ FI = E,\ PO = E$ $FO = B \cup B' \cup G$
$5, *, (2), \emptyset$	$\{\mathbf{5.1}: D_j \to B_{j,0},\ \mathbf{5.2}: D'_j \to B_{j,0} \mid j \in I\} \cup$ $\{\mathbf{5.3}: F_j \to A_j \mid j \in I\} \ \cup \{\mathbf{5.4}: D_j \to G_{s,t},$ $\mathbf{5.5}: D'_j \to G_{s,t} \mid A_j \to A_s A_t, j, s, t \in I'\}$ $PI = D \setminus \{D_0\} \cup D'$ $FI = E \cup E' \cup \{D_0\} \ \cup C \cup C' \cup \{\varepsilon'\}$ $PO = \emptyset,\ FO = D \cup D' \cup F$
$6, l, (2), \emptyset$	$\{\mathbf{6.1}: \varepsilon' \to \varepsilon\}$ $PI = \{\varepsilon'\}$ $FI = B \setminus B_0 \cup B' \cup C \cup C' \cup F \cup (D \setminus \{D_0\})$ $PO = \emptyset,\ FO = \{\varepsilon'\}$
$7, *, (2), \emptyset$	$\emptyset$ $PI = T,\ FI = V \setminus T,\ PO = \emptyset,\ FO = T$

Table 1.

We note that during this phase of the computation rules `1.2`: $A_i \to \varepsilon'$ or `2.7`: $A_n \to \varepsilon'$ may be applied in nodes 1 and 2. In this case, the string leaves node 1 or 2, but cannot enter any node. So, this case will not be considered in the sequel.

The process continues in nodes 1 and 2 until subscript i of C_i or that of C'_i is decreased to 1. In this case, either rule `1.7` : $C_1 \to \varepsilon'$ in node 1 or rule `2.3` : $C'_1 \to \varepsilon'$ in node 2 will be applied and the obtained string $\varepsilon' w B'_{i_k,i_1}$ or $\varepsilon' w B_{i_k,i_1}$ is communicated to node 3. (Notice that the string is able to leave the node either if both C and B are primed or both of them are unprimed.) Then, in node 3, depending on the form of the string, either evolution step $\varepsilon' w B'_{i_k,i_1} \xrightarrow{3.1} \varepsilon' w B'_{i_k,i_1} D_0$ or evolution step $\varepsilon' w B_{i_k,i_1} \xrightarrow{3.1} \varepsilon' w B_{i_k,i_1} D_0$ is performed. Strings $\varepsilon' w B'_{i_k,i_1} D_0$ or $\varepsilon' w B_{i_k,i_1} D_0$ can enter only node 4, where (depending on the form of the string) either evolution step $\varepsilon' w B_{i_k,i_1} D_0 \xrightarrow{4.1} \varepsilon' w E_{i_k,i_1} D_0$ or evolution step $\varepsilon' w B'_{i_k,i_1} D_0 \xrightarrow{4.2} \varepsilon' w E_{i_k,i_1} D_0$ follows. The obtained word, $\varepsilon' w E_{i_k,i_1} D_0$, can enter only node 6, where evolution step $\varepsilon' w E_{i_k,i_1} D_0 \xrightarrow{6.1} w E_{i_k,i_1} D_0$ is performed. Then the string leaves the node and enters node 2.

Then, in nodes 2 and 1, a sequence of computation steps is performed, when the string is involved in evolution steps followed by communication as follows:

$$w E_{i_k,i_1-t} D_t \xrightarrow{2.4} w E'_{i_k,i_1-(t+1)} D_t \xrightarrow{2.5} w E'_{i_k,i_1-(t+1)} D'_{t+1} \text{ (in node 2).}$$

$$w E'_{i_k,i_1-t} D'_t \xrightarrow{1.8} w E_{i_k,i_1-(t+1)} D'_t \xrightarrow{1.9} w E_{i_k,i_1-(t+1)} D_{t+1} \text{ (in node 1),}$$

The process continues in nodes 1 and 2 until the second subscript of $E'_{i,j}$ or that of $E_{i,j}$ is decreased to 1. In this case, either rule `1.10` : $E'_{i_k,1} \to F_{i_k}$ in node 1 or rule `2.6` : $E_{i_k,1} \to F_{i_k}$ in node 2 is applied and the new string, $w F_{i_k} D_{i_1}$ or $w F_{i_k} D'_{i_1}$, will be present in node 5. Notice that applying rules `1.1, 1.2` and `2.7` we obtain strings that cannot enter nodes 3 – 7 and stay in nodes 1 or 2.

The next evolution steps that take place in node 5 are as follows:

$$w F_{i_k} D_{i_1} (w F_{i_k} D'_{i_1}) \xrightarrow{5.1(5.2)} w F_{i_k} B_{i_1,0} \xrightarrow{5.3} w A_{i_k} B_{i_1,0}.$$

In the following communication step, string $wA_{i_k}B_{i_1,0}$ can enter either node 1 or node 2 (if $A_{i_1} \in T$). In the first case, the rotation of symbol A_{i_1} has been successful. Let us consider the second case. Then string $wA_{i_k}B_{i_1,0}$ appears in node 2.

- Suppose that the word $wA_{i_k}B_{i_1,0}$ does not contain any nonterminal symbol except A_n. Let $wA_{i_k}B_{i_1,0} = A_nw'A_{i_k}B_{i_1,0}$, where $w = A_nw'$. So, $w'A_{i_k}A_{i_1}$ is a result and it appears in node 7. Notice that if $w = w'A_nw''$ and $w' \neq \varepsilon$, then word $w'A_nw''A_{i_k}B_{i_1,0}$ leads to a string which will stay in node 6 forever (if rule `2.7` was applied). So, we consider the following evolution of the word $wA_{i_k}B_{i_1,0} = A_nw'A_{i_k}B_{i_1,0}$: $A_nw'A_{i_k}B_{i_1,0} \xrightarrow{2.7} \varepsilon'w'A_{i_k}B_{i_1,0} \xrightarrow{2.8} \varepsilon'w'A_{i_k}A_{i_1}$. Then, string $\varepsilon'w'A_{i_k}A_{i_1}$ will appear in node 6, where symbol ε' will be deleted by rule `6.1`. Finally, the resulted word $w'A_{i_k}A_{i_1}$ will enter node 7. This is a result.

- Suppose now that the word $wA_{i_k}B_{i_1,0}$ contains at least one nonterminal symbol different from A_n and $A_{i_1} \in T$.

  Consider the evolution of the word $wA_{i_k}B_{i_1,0} = w'A_nw''A_{i_k}B_{i_1,0}$ in node 2:

  $$w'A_nw''A_{i_k}B_{i_1,0} \xrightarrow{2.8} w'A_nw''A_{i_k}A_{i_1} \xrightarrow{2.7} w'\varepsilon'w''A_{i_k}A_{i_1}.$$

  Now, string $w'\varepsilon'w''A_{i_k}A_{i_1}$ will enter node 6 and either it will not be able to leave it (if $w' \neq \varepsilon$) or it will not be able to enter any of the other nodes (if $w' = \varepsilon$).

In the following we will explain how the application of the rules of G are simulated in Γ.

Rule $A_i \longrightarrow \varepsilon$**.** Suppose that string $A_iwB_{j,0}$ is in node 1 and let $w, w', w'' \in A^*$. Then, by evolution, we obtain $A_iwB_{j,0} \xrightarrow{1.2} \varepsilon'wB_{j,0}$ or $A_tw'A_iw''B_{j,0} \xrightarrow{1.2} A_tw'\varepsilon'w''B_{j,0}$ which can enter only node 6. String $A_tw'\varepsilon'w''B_{j,0}$ will stay in node 6 forever. By evolution $\varepsilon'wB_{j,0} \xrightarrow{6.1} wB_{j,0}$ and the resulting string, $wB_{j,0}$, enters in node 1 (and node 2, if $A_j \in T$). Thus, the application of rule $A_i \longrightarrow \varepsilon$ in G was correctly simulated.

Rule $A_i \longrightarrow A_j$**.** The evolution step performed at node 1 is $wB_{i,0} \xrightarrow{1.3} wB_{j,0}$. Since string $wB_{j,0}$ is in node 1, the simulation of the rule $A_i \longrightarrow A_j$ of grammar G was done in a correct manner.

Rule $A_j \longrightarrow A_sA_t$**.** At the end of the simulation of the rotation of a symbol, in node 5 instead of applying rule $D_j \rightarrow B_{j,0}$ $(D'_j \rightarrow B_{j,0})$ a rule $D_j \rightarrow G_{s,t}$ $(D'_j \rightarrow G_{s,t})$ is applied. That is, in node 5, either evolution step $wD_j \xrightarrow{5.4} wG_{s,t}$ or evolution step $wD'_j \xrightarrow{5.5} wG_{s,t}$ is performed. The new string $wG_{s,t}$ can enter only node 3, where, by evolution, $wG_{s,t} \xrightarrow{3.1} wG_{s,t}D_0$. String $wG_{s,t}D_0$ can enter only node 4, where evolution step $wG_{s,t}D_0 \xrightarrow{4.5} wE_{s,t}D_0$ follows. The process continues as above, in the case of simulating rotation, and in several computation steps the string wF_sD_t or $wF_sD'_t$ will enter node 5. After evolution in this node, the resulting string $wA_sB_{t,0}$ will enter node 1 (and node 2, if $A_t \in T$). Thus, the application of rule $A_j \longrightarrow A_sA_t$ of G is correctly simulated.
Rule $A_iA_j \longrightarrow A_sA_t$**.** The evolutionary processor in node 4 has rules `4.3` : $B_{i,j} \rightarrow E_{s,t}$ or `4.4` : $B'_{i,j} \rightarrow E_{s,t}$. As in the case of simulating rotation, above, we will obtain string $wA_sB_{t,0}$ in node 1 (and in node 2, if $A_t \in T$).

We have demonstrated how the rotation of a symbol and the application of rules of G are simulated by Γ. By the constructions, the reader can verify that G and Γ generate the same language. $\square$

3.4 Obligatory HNEPs

As described in the second section, obligatory operations were considered in HNEPs. The result of the evolution step now consists of all strings produced from the current ones by the operations of insertion, deletion and substitution (the current strings are no longer preserved, even if some operation is not applicable to them). Not only this yields a simpler and a more uniform definition, but also the following result is obtained.

Theorem 4 *Any CPM0 P can be simulated by an OHNEP P', where* *obligatory evolutionary processors are with empty input and output filters and only insertion and obligatory deletion operations in right and left modes are used (without obligatory substitution operations). [3, 2]*

Proof. Let us consider a CPM0 P with symbols $a_j \in \Sigma$, $j \in J = \{0, 1 \dots, n\}$, $a_0 = 0$ is the blank symbol, and states, $q_i \in Q$, $i \in I = \{1, 2, \dots, f\}$, where q_1 is the initial state and the only terminal state is $q_f \in Q$. We suppose that P stops in the terminal state q_f on every symbol, i.e., there are instructions $q_f a_j \rightarrow Halt$, $a_j \in J$. (Notice that it is easy to transform any CPM0 P into a CPM0 P' that stops on every symbol in the final state.)

So, we consider CPM0 P with the set R of instructions of the forms $q_i a_j \longrightarrow q_l$, $q_i a_j \longrightarrow a_k q_l$, $q_i 0 \longrightarrow a_k q_m 0$, $q_f a_j \longrightarrow Halt$, where $q_i \in Q \setminus \{q_f\}$, $q_l, q_m \in Q$, $a_j, a_k \in \Sigma$. We do not consider case $q_m = q_f$ in instruction $q_i 0 \longrightarrow a_k q_m 0$. Notice that it is easy to modify the program of P such that it only halts by instructions of other types.

A configuration $w = q_i a_j W$ of CPM0 P describes that P in state $q_i \in Q$ considers symbol $a_j \in \Sigma$ to the left of $W \in \Sigma^*$.

Now we construct an OHNEP P' simulating P. To simplify the description of P', we use $\langle q_f a_j \rangle$ and $\langle q_f a_j \rangle_1$, $j \in J$ as aliases of $\langle out \rangle$. Let v take values from Q and let u take values from $\in Q \cup Q \cdot \{0\}$.

$$
\begin{aligned}
P' &= (V, G, N, C_0, \alpha, \beta, i_0),\\
V &= \{q_1\} \cup \Sigma,\\
G &= (X_G, E_G),\\
X_G &= \{\langle init \rangle, \langle out \rangle\}\\
&\cup\ \{\langle q_i a_j \rangle \mid (q_i a_j \rightarrow v) \in R\}\\
&\cup\ \{\langle q_i a_j \rangle_1 \mid (q_i a_j \rightarrow a_k u) \in R\},\\
E_G &= \{(\langle init \rangle, \langle q_1 a_j \rangle) \mid j \in J\}\\
&\cup\ \{(\langle q_i a_j \rangle, \langle q_l a_k \rangle) \mid (q_i a_j \rightarrow q_l) \in R,\ k \in J\}\\
&\cup\ \{(\langle q_i a_j \rangle, \langle q_i a_j \rangle_1) \mid (q_i a_j \rightarrow a_k u) \in R\}\\
&\cup\ \{(\langle q_i a_j \rangle_1, \langle q_l a_s \rangle) \mid (q_i a_j \rightarrow a_k q_l) \in R,\ s \in J\}\\
&\cup\ \{(\langle q_i 0 \rangle_1, \langle q_l 0 \rangle_1) \mid (q_i 0 \rightarrow a_k q_l 0), (q_l 0 \rightarrow a_s u) \in R, s \in J\},
\end{aligned}
$$

$$\cup \quad \{(\langle q_i 0\rangle_1, \langle q_p a_s\rangle) \mid (q_i 0 \to a_k q_l 0), (q_l 0 \to q_p) \in R, s \in J\},$$

$$\begin{array}{rcl}
C_0(x) & = & \{q_1 W\},\ \text{if } x = \langle init\rangle, \\
 & & \ \text{where } W \text{ is the input of } P, \\
C_0(x) & = & \emptyset,\ x \in X_G \setminus \{\langle init\rangle\}, \\
\beta(x) & = & 2,\ x \in X_G, \\
N(x) & = & (M_x, \emptyset, \emptyset, \emptyset, \emptyset),\ x \in X_G, \\
M_x & = & \{q_1 \to \varepsilon\}, x = \langle init\rangle, \\
M_x & = & \{a_j \to \varepsilon\}, x = \langle q_i a_j\rangle, \\
M_x & = & \{\varepsilon \to a_k\}, x = \langle q_i a_j\rangle_1, \\
 & & \text{where } (q_i a_j \to a_k u) \in R, \\
\alpha(x) & = & l,\ \text{if } M_x = \{a \to \varepsilon\}, \\
\alpha(x) & = & r,\ \text{if } M_x = \{\varepsilon \to a\},\ \text{or } M_x = \emptyset.
\end{array}$$

OHNEP P' will simulate every computation step performed by CPM0 P by a sequence of computation steps in P'.

Let $q_1 a_j W_0$ be the initial configuration of CPM0 P. We represent this configuration in node $\langle init\rangle$ of OHNEP P' as a word $q_1 a_j W_0$. Obligatory evolutionary processor associated with this node is $N(\langle init\rangle) = (\{(q_1 \to \varepsilon)^l\}, \emptyset, \emptyset, \emptyset, \emptyset)$. Since all other nodes also have empty filters, in the following we will skip the complete description of obligatory evolutionary processors, and will present only their obligatory evolutionary operations. The word $a_j W_0$ will be passed from node $\langle init\rangle$ to nodes $\langle q_1 a_j\rangle,\ j \in J$.

If the computation in P is finite, then the final configuration $q_f W$ of P will appear at node $\langle out\rangle$ of P' as a string W, moreover, any string W that can appear at node $\langle out\rangle$ corresponds to a final configuration $q_f W$ of P. In the case of an infinite computation in P, no string will appear in node $\langle out\rangle$ of P' and the computation in P' will never stop.

Now we describe nodes of OHNEP P', connections between them and obligatory evolutionary operations, associated with these nodes. Let $I' = I \setminus \{f\}$.

1. Node $\langle q_i a_j \rangle$ with operation $(a_j \to \varepsilon)^l,\ i \in I',\ j \in J$.
   Let word $a_t W,\ t \in J,\ W \in \Sigma^*$ appear in this node. If $j \neq t$, then this word $a_t W$ will be discarded, and in the next communication step node $\langle q_i a_j \rangle$ will send nothing. If $j = t$, then the node sends W to nodes $\{\langle q_l a_k \rangle \mid k \in J\}$ or $\langle q_i a_j \rangle_1$.
   - Instruction of P is $q_i a_j \longrightarrow q_l,\ i \in I',\ j \in J,\ l \in I$. Node $\langle q_i a_j \rangle$ is connected with nodes $\{\langle q_l a_k \rangle \mid k \in J\}$.
   - Instructions of P are $q_i a_j \longrightarrow a_k q_l$ or $q_i 0 \longrightarrow a_k q_l 0,\ i \in I',\ j,k \in J,\ l \in I$. Node $\langle q_i a_j \rangle$ is connected with node $\langle q_i a_j \rangle_1$.
2. Node $\langle q_i a_j \rangle_1,\ i \in I',\ j \in J$ with operation $(\varepsilon \to a_k)^r$ receives word W and sends word $W a_k$ to nodes $\{\langle q_l a_s \rangle \mid s \in J\}$ or $\langle q_l 0 \rangle_1$.
   - Instructions of P are $q_i a_j \longrightarrow a_k q_l,\ i \in I',\ j,k \in J,\ l \in I$. Node $\langle q_i a_j \rangle_1$ is connected with nodes $\{\langle q_l a_s \rangle \mid s \in J\}$.
   - Instruction of P is $q_i 0 \longrightarrow a_k q_l 0,\ i \in I',\ k \in J,\ l \in I$. Node $\langle q_i 0 \rangle_1$ is connected with nodes $\{\langle q_p a_s \rangle \mid s \in J\}$ if there exists an instruction of P $q_l 0 \longrightarrow q_p,\ p \in I$; and with node $\langle q_l 0 \rangle_1$ in other cases.

We repeat that in all cases, we mean $\langle out \rangle$ whenever we write $\langle q_f a_j \rangle$ or $\langle q_f a_j \rangle_1,\ j \in J$.

Now we describe simulation of instructions of CPM0 P by OHNEP P'.

Instruction $q_i a_j \longrightarrow q_l$**:** $q_i a_j W \xrightarrow{P} q_l W$**.**

Let word $a_t W$, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_i a_j \rangle$. If $t \neq j$, string $a_t W$ will be discarded; if $t = j$, string W will be passed to nodes $\{\langle q_l a_s \rangle \mid s \in J\}$. If $l = f$, the final configuration $q_f W$ of P will appear in the output node $\langle out \rangle$ as W. This is the result. So, we simulated instruction $q_i a_j \longrightarrow q_l$ in a correct manner.

Instruction $q_i a_j \longrightarrow a_k q_l$**:** $q_i a_j W \xrightarrow{P} q_l W a_k$**.**

Let word $a_t W$, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_i a_j \rangle$. If $t \neq j$, string $a_t W$ will be discarded; if $t = j$ string W will be passed to node $\langle q_i a_j \rangle_1$. Node $\langle q_i a_j \rangle_1$ receives this word and sends word Wa_k to nodes $\langle q_l a_s \rangle,\ s \in J$. If $l = f$, the final configuration $q_f W a_k$ of P will appear in the output node $\langle out \rangle$ as Wa_k. This is the result. So, we simulated instruction $q_i a_j \longrightarrow a_k q_l$ in a correct manner.

Instruction $q_i 0 \longrightarrow a_k q_l 0$**:** $q_i 0 W \xrightarrow{P} q_l 0 W a_k$**.**

Let word $a_t W$, where $t \in J,\ W \in \Sigma^*,\ i \in I'$ appear in node $\langle q_i 0 \rangle$. If $a_t \neq 0$, string $a_t W$ will be discarded; if $a_t = 0$, string W will be passed to node $\langle q_i 0 \rangle_1$. It receives this word and sends word Wa_k to nodes $\langle q_p a_s \rangle, s \in J$ if there is instruction of P $q_l 0 \longrightarrow q_p, p \in I'$. If there are instructions $q_l 0 \longrightarrow a_s q_p$ or $q_l 0 \longrightarrow a_s q_p 0$, then node $\langle q_i 0 \rangle_1$ is connected with node $\langle q_l 0 \rangle_1$. Thus, word Wa_k will be passed to node $\langle q_l 0 \rangle_1$, which corresponds to the configuration of P which has "just read" symbol 0 in state q_l. So, we simulated instruction $q_i 0 \longrightarrow a_k q_l 0$ in a correct manner.

So, CPM0 P is correctly modeled. We have demonstrated that the rules of P are simulated in P'. The proof that P' simulates only P comes from the construction of the rules in P', we leave the details to the reader. □

4 Conclusion

We have described the networks of evolutionary processors, their models and variants, together with the associated results. A few selected results were presented in more details. For instance, NEPs with two nodes are already computationally complete modulo the terminal alphabet. HNEPs with one node are given the precise regular characterization, HNEPs with two nodes are not computationally complete, while seven nodes are enough to reach the computational completeness of HNEPs, even with a complete graph. We should mention that a network over a complete graph (with loops, although it is not important for the last proof) may be viewed as number of agents in a common environment, acting "independently" without explicitly enforcing any transition protocol, where a computationally complete behavior still emerges.

A particularly interesting variant is obligatory HNEPs (OHNEPs). Using a power of the underlying graph, computational completeness is obtained even without the filters. In case of a complete graph, OHNEPs are still computationally complete. Moreover, it suffices that the sum of numbers of symbols in filters of each node does not exceed one. The last proof has been obtained in [1], using a variant of circular Post machines, CPM5, introduced in [11].

References

[1] A. Alhazov, G. Bel-Enguix, I. Epifanova, Yu. Rogozhin. *About Two Models of Complete Obligatory Hybrid Networks of Evolutionary Processors.* In preparation.

[2] A. Alhazov, G. Bel-Enguix, Yu. Rogozhin. *About a New Variant of HNEPs: Obligatory Hybrid Networks of Evolutionary Processors.* In: G. Bel-Enguix, M. D. Jiménez-López (Eds), Bio-Inspired Models for Natural and Formal Languages, Cambridge Scholars Publishing, 2011, pp.191–204.

[3] A. Alhazov, G. Bel-Enguix, Yu. Rogozhin. *Obligatory Hybrid Networks of Evolutionary Processors.* International Conference on Agents and Artificial Intelligence, Porto, 2009, INSTICC Press, pp.613–618.

[4] A. Alhazov, G. Bel-Enguix, A. Krassovitskiy, Yu. Rogozhin. *About Complete Obligatory Hybrid Networks of Evolutionary Processors without Substitution.* Advances in Computational Intelligence, 11th International Work-Conference on Artificial Neural Networks, IWANN 2011, Málaga, Lecture Notes in Computer Science, **6691**, 2011, pp.441–448.

[5] A. Alhazov, G. Bel-Enguix, A. Krassovitskiy, Yu. Rogozhin. *Complete Obligatory Hybrid Networks of Evolutionary Processors.*

Highlights in Practical Applications of Agents and Multiagent Systems, Salamanca, Advances in Intelligent and Soft Computing, **89**, 2011, pp.275–282.

[6] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. *About Universal Hybrid Networks of Evolutionary Processors of Small Size.* 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, pp.28–39.

[7] 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.

[8] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. *On the Size of Computationally Complete Hybrid Networks of Evolutionary Processors.* Theoretical Computer Science, 410, 35, 2009, pp.3188–3197.

[9] A. Alhazov, J. Dassow, C. Martín-Vide, Yu. Rogozhin, B. Truthe. *On Networks of Evolutionary Processors with Nodes of Two Types.* Fundamenta Informaticae, **91**, 1, 2009, pp.1–15.

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

[11] A. Alhazov, A.Krassovitsiy, Yu.Rogozhin. *Circular Post Machines and P Systems with Exo-insertion and Deletion.* Lecture Notes in Computer Science, **7184**, Springer, 2012, pp.73–86.

[12] A. Alhazov, C. Martín-Vide, Yu. Rogozhin. *On the Number of Nodes in Universal Networks of Evolutionary Processors.* Acta Informatica, **43(5)**, 2006, pp.331–339.

[13] A. Alhazov, C. Martín-Vide, Yu. Rogozhin. *Networks of Evolutionary Processors with Two Nodes Are Unpredictable.* 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. Also in: Technical Report, **818**, Turku Centre for Computer Science, Turku, 2007.

[14] A. Alhazov, Yu. Rogozhin. *About Precise Characterization of Languages Generated by Hybrid Networks of Evolutionary Processors with One Node.* The Computer Science Journal of Moldova, **16(3)**, 2008, pp.364–376.

[15] J. Castellanos, P. Leupold, V. Mitrana. *On the Size Complexity of Hybrid Networks of Evolutionary Processors.* Theoretical Computer Science, **330(2)**, 2005, pp.205–220.

[16] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere. *Networks of Evolutionary processors.* Acta Informatica, **38**, 2003, pp.517-529.

[17] 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.

[18] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana. *Hybrid Networks of Evolutionary Processors are Computationally Complete.* Acta Informatica, **41(4-5)**, 2005, 257–272.

[19] E. Csuhaj-Varjú, A. Salomaa. *Networks of Parallel Language Processors.* In: Gh. Păun, A. Salomaa, (Eds.), New Trends in formal Language Theory Lecture Notes in Computer Science, **1218**, Springer, 1997, pp.299-318.

[20] 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.

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

[22] 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.

[23] R. Loos, F. Manea, V. Mitrana. *Small Universal Accepting Hybrid Networks of Evolutionary Processors.* Acta Informatica, **47**, 2, 2010, pp.133–146.

[24] 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, pp.753–771.

[25] 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, pp.112–118.

[26] 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.

[27] 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, pp.401-412.

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

Artiom Alhazov

Received April 9, 2013

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