OBLIGATORY HYBRID NETWORKS OF EVOLUTIONARY PROCESSORS

Artiom Alhazov
Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Str. Academiei 5, MD-2028, Chişinău, Moldova

Gemma Bel-Enguix, Yurii Rogozhin
Rovira i Virgili University, Research Group on Mathematical Linguistics, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

Keywords: Formal languages, Networks of evolutionary processors, Point mutations, Distributed computing, Circular Post machines.

Abstract: In this paper obligatory hybrid networks of evolutionary processors (a variant of hybrid networks of evolutionary processors model) are proposed. In the obligatory hybrid network of evolutionary processors a node discards the strings to which no operations are applicable. We show that such networks have the same computability power as Turing machines only using one operation per node (deletion on the left end and insertion on the right end of the string) no rewriting and no filters.

1 INTRODUCTION

Insertion, deletion, and substitution are fundamental operations in formal language theory, their power and limits have obtained much attention. Due to their simplicity, language generating mechanisms based on these operations are of particular interest. *Networks of evolutionary processors* (NEPs, for short), introduced in (Castellanos et al., 2001), are proper examples for distributed variants of these constructs. In this case, evolutionary processors (language processors performing insertion, deletion, and substitution of a symbol) are located at nodes of a virtual graph and operate over sets or multisets of words. During the functioning of the system, they rewrite the corresponding collections of words and then re-distribute the resulting strings according to a communication protocol assigned to the system. The language determined by the network is usually 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. 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 (Alhazov et al., 2006; Alhazov et al., 2007)).

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. The notion was introduced in (Martín-Vide et al., 2003). In (Csuhaj-Varjú et al., 2005) it was shown that, for an alphabet V, HNEPs with $27+3\cdot card(V)$ nodes are computationally complete. A significant improvement of the result can be found in (Alhazov et al., 2008a), where it was proved that HNEPs with 10 nodes (irrespectively of the size of the alphabet) reach the universal power and at last in (Alhazov et al., 2008b) it was showed that HNEPs with 7 nodes can reach the universal power. Notice, that the family of HNEPs with 2 nodes is not computationally complete (Alhazov et al., 2008b).

In this paper, we consider new variant of HNEP, so called Obligatory Network of Evolutionary Processors (OHNEP shortly). The differences between

Alhazov A., Bel-Enguix G. and Rogozhin Y. (2009).
OBLIGATORY HYBRID NETWORKS OF EVOLUTIONARY PROCESSORS.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 613-618
DOI: 10.5220/0001809106130618

HNEP and OHNEP are:

1. in using deletion and substitution operations: a node discards a string if no operations in node are applicable to string (in HNEP case this string remains in the node),
2. an underlying graph is directed graph (in HNEP case this graph is undirected).

We underline that both differences are natural. The first one allows us to have the uniform definitions of the operations on a string, as opposed to considering two cases as in HNEPs (it is the set of results of the applications of the operation to all possible positions; the case when there are no such position yields the empty set by definition). The second difference, that of generalization of the underlying graph to be directed, is natural from the computational point of view; moreover, since the loops are typically not considered, it also seems relevant from the viewpoint of the biological motivation that the communicating channels are directed.

These differences allow to proof universality of OHNEP with nodes with only one operation, without input and output filters and using only insertion operation at the left end and deletion operation at the right end of a string. This interesting fact stresses the importance of structure of HNEP in order to reach universality. On the other hand we can avoid substitution operation. Notice that this feature of OHNEP to discard a string if this string does not participate at the operations has counterpart in DNA computing area, TVDH systems also discard strings if they do not participate at splicing operations (Margenstern et al., 2004).

A task to find a minimal number of nodes of universal OHNEP is open. A variant of OHNEP with underlying complete graph is not considered yet. An implementation of HNEPs and OHNEPs in mathematical linguistics is also interesting task to investigate.

2 DEFINITIONS

We recall some notions we shall use throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written as $card(A)$. A sequence of symbols from an alphabet V is called a word over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε; we use $V^+ = V^* \setminus \{\varepsilon\}$. The length of a word x is denoted by $|x|$, while we denote the number of occurrences of a letter a in a word x by $|x|_a$. For each nonempty word x, $alph(x)$ is the minimal alphabet W such that $x \in W^*$.

Circular Post Machines (CPMs) were introduced in (Kudlek and Rogozhin, 2001b), 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 (Kudlek and Rogozhin, 2001a; Alhazov et al., 2002) several universal CPMs of variant 0 (CPM0) having small size were constructed, among them in (Alhazov et al., 2002) 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, an initial state $\mathbf{q}_0 \in Q$, a terminal 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 and cut off 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 creation of a blank), where $x, y \in \Sigma$ and $\mathbf{p}, \mathbf{q} \in Q$.

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

In the following, we summarize the necessary notions concerning *obligatory evolutionary operations*.

For an alphabet V, we say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *obligatory substitution operation* if both a and b are different from ε; it is a *obligatory deletion operation* if $a \neq \varepsilon$ and $b = \varepsilon$; and, it is an *(obligatory) insertion operation* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all obligatory substitution, deletion, and insertion operations over an alphabet V are 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 *obligatory evolutionary actions* of π, ρ, 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

$$\pi^*(w) = \{ubv \mid w = uav,\ u, v \in V^*\} \quad (1)$$
$$\rho^*(w) = \{uv \mid w = uav,\ u, v \in V^*\} \quad (2)$$
$$\rho^r(w) = \{u \mid w = ua\} \quad (3)$$
$$\rho^l(w) = \{v \mid w = av\} \quad (4)$$
$$\sigma^*(w) = \{uav \mid w = uv,\ u, v \in V^*\} \quad (5)$$
$$\sigma^r(w) = \{wa\} \quad (6)$$
$$\sigma^l(w) = \{aw\} \quad (7)$$

Notice, that in (1) – (4) a result of obligatory evolution operation may be empty set (this is the main difference between obligatory hybrid network of evolutionary processors and hybrid network of evolutionary processors).

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.

Before turning to the notion of an evolutionary processor, we define the filtering mechanism.

For disjoint subsets $P, F \subseteq V$ and a word $w \in V^*$, we define the predicate φ ($\varphi^{(2)}$ in terminology of (Csuhaj-Varjú et al., 2005)) as $\varphi(w; P, F) \equiv alph(w) \cap P \neq \emptyset \ \wedge \ F \cap alph(w) = \emptyset$. The construction of this predicate 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 $\varphi(L, P, F) = \{w \in L \mid \varphi(w; P, F)\}$.

An *obligatory 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 obligatory evolutionary operations of the processor. Note that every processor is dedicated to only one type of the above obligatory 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.

We denote the set of obligatory evolutionary processors over V by OEP_V.

Definition 2.1 *An obligatory hybrid network of evolutionary processors (an OHNEP, shortly) is a 7-tuple* $\Gamma = (V, G, N, C_0, \alpha, \beta, i_0)$, *where the following conditions hold:*

- *V is an alphabet.*

- *$G = (X_G, E_G)$ is a directed graph with set of vertices X_G and set of edges E_G. G is called the* underlying graph *of the network.*

- *$N : X_G \longrightarrow OEP_V$ is a mapping which associates with each node $x \in X_G$ the obligatory evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.*

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

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

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

- *$i_0 \in X_G$ is the* output node *of the OHNEP.*

Notice, that in the definition of OHNEP above $G = (X_G, E_G)$ is a directed graph, but in the definition of HNEP (see, for example, (Csuhaj-Varjú et al., 2005)), underlying graph is an undirected graph. This is the second difference between HNEP and OHNEP.

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

A configuration of an OHNEP Γ, as above, is a mapping $C : X_G \longrightarrow 2^{V^*}$ 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 changed 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 $x \in X_G$.

When it changes by a communication step, then each language processor $N(x)$, where $x \in X_G$, sends a copy of each of its words 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, providing that these words are able to pass the input filter of x. Formally, we say that configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff $C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{(y,x) \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y)))$ for all $x \in X_G$.

For an OHNEP Γ, a 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 $i > 0$. If we use OHNEPs as language generating devices, then the generated language is the set of all words which appear in the output node at some step of the computation. Formally, the language generated by Γ is $L(\Gamma) = \bigcup_{s \geq 0} C_s(i_0)$.

3 MAIN RESULT

Theorem 1 *Any CPM0 P can be simulated by an OHNEP P', where obligatory evolution 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).*

Proof. Let us consider a CPM0 P with symbols $a_j \in \Sigma,\ j \in J = \{0,1\ldots,n\}$, $a_0 = 0$ is a blank symbol, and states, $q_i \in Q,\ i \in I = \{1,2,\ldots,f\}$, 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 $q_f a_j \to Halt$, $a_j \in J$. (Notice, that it is easy to transform any CPM0 P into a CPM0 P' that stops on every symbol in terminal state.)

So, we consider CPM0 P with the set R of instructions of the forms $q_i a_j \longrightarrow q_l$, $q_i a_j \longrightarrow a_k q_l$, $q_i 0 \longrightarrow a_k q_l 0$, $q_f a_j \longrightarrow Halt$, where $q_i \in Q \setminus \{q_f\},\ q_l \in Q$, $a_j, a_k \in \Sigma$. A configuration $w = q_i a_j W$ of CPM0 P describes that P in state $q_i \in Q$ considers symbol $a_j \in \Sigma$ on the left-hand end 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 $\langle out \rangle$.

$$
\begin{aligned}
P' &= (V, G, N, C_0, \alpha, \beta, i_0),\\
V &= \{q_1\} \cup \Sigma,\\
G &= (X_G, E_G),\\
X_G &= \{\langle init \rangle, \langle out \rangle\}\\
&\cup \{\langle q_i a_j \rangle \mid (q_i a_j \to u) \in R\}\\
&\cup \{\langle q_i a_j \rangle_1 \mid (q_i a_j \to a_k u) \in R\},\\
E_G &= \{(\langle init \rangle, \langle q_1 a_j \rangle) \mid j \in J\}\\
&\cup \{(\langle q_i a_j \rangle, \langle q_l a_k \rangle)\\
&\quad \mid (q_i a_j \to q_l) \in R,\ k \in J\}\\
&\cup \{(\langle q_i a_j \rangle, \langle q_i a_j \rangle_1) \mid (q_i a_j \to a_k u) \in R\}\\
&\cup \{(\langle q_i a_j \rangle_1, \langle q_l a_s \rangle)\\
&\quad \mid (q_i a_j \to a_k q_l) \in R,\ s \in J\}\\
&\cup \{(\langle q_i 0 \rangle_1, \langle q_l 0 \rangle_1) \mid (q_i 0 \to a_k q_l 0) \in R\},\\
C_0(x) &= \{q_1 W\}, \text{ if } x = \langle init \rangle,\\
&\quad \text{where } W \text{ is the input of } P,\\
C_0(x) &= \emptyset,\ x \in X_G \setminus \{\langle init \rangle\},\\
\beta(x) &= 2,\ x \in X_G,\\
N(x) &= (M_x, \emptyset, \emptyset, \emptyset, \emptyset),\ x \in X_G,\\
M_x &= \{q_1 \to \varepsilon\}, x = \langle init \rangle,\\
M_x &= \{a_j \to \varepsilon\}, x = \langle q_i a_j \rangle,\\
M_x &= \{\varepsilon \to a_k\}, x = \langle q_i a_j \rangle_1,\\
&\quad \text{where } (q_i a_j \to a_k u) \in R,\\
\alpha(x) &= l, \text{ if } M_x = \{a \to \varepsilon\},\\
\alpha(x) &= r, \text{ if } M_x = \{\varepsilon \to a\}, \text{ or } M_x = \emptyset.
\end{aligned}
$$

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

Let $q_1 a_j W_0$ be the initial configuration of CPM0 P. We present this configuration in node $\langle init \rangle$ of OHNEP P' as word $q_1 a_j W_0$. Obligatory evolution processor, associated with this node is $N(\langle init \rangle) = (\{(q_1 \to \varepsilon)^l\}, \emptyset, \emptyset, \emptyset, \emptyset)$. In the following we will omit complete description of obligatory evolution processor, and will present only obligatory evolution operation. Further word $a_j W_0$ from node $\langle init \rangle$ will be passed 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 on 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 is $q_i a_j \longrightarrow a_k q_l$ 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 is $q_i a_j \longrightarrow a_k q_l,\ i \in I',\ j,k \in J,\ l \in I$. 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 node $\langle q_l 0 \rangle_1$.

Again 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 $t \in J$, $W \in \Sigma^*$, $i \in I'$ appears 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 $\{\langle q_l a_j \rangle \mid j \in J\}$. 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 $t \in J$, $W \in \Sigma^*$, $i \in I'$ appears 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 $W a_k$ 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 $W a_k$. 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 $t \in J$, $W \in \Sigma^*$, $i \in I'$ appears 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 $W a_k$ to node $\langle q_l 0 \rangle_1$. If $l = f$, the final configuration $q_f 0 W a_k$ of P will appear in the output node $\langle out \rangle$ as a word $W a_k$. This is the result (we can avoid the case of missing symbol 0 if the simulated CPM0 is modified to only halt by instructions of previous types). In case $l \neq f$, word $W a_k$ will be passed to the node $\langle q_l 0 \rangle_1$, which correspond 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.

□

Example 3.1 *Consider the following CPM0:* $M = (\{0,1\}, \{q_1, q_2, q_3, q_f\}, q_1, q_f, P)$ *with the following commands (we do not list a command with* $q_3 1$ *on the left because it is not reachable).*

$$q_1 0 \to 1 q_3 0 \quad q_2 0 \to q_f \quad q_3 0 \to 1 q_2$$
$$q_1 1 \to 0 q_1 \quad q_2 1 \to q_1$$

We illustrate the work of M as follows. It transforms 101 *into* 011 *in 13 steps:* $q_1 101 \Rightarrow q_1 010 \Rightarrow q_3 0101 \Rightarrow q_2 1011 \Rightarrow q_1 011 \Rightarrow q_3 0111 \Rightarrow q_2 1111 \Rightarrow q_1 111 \Rightarrow q_1 110 \Rightarrow q_1 100 \Rightarrow q_1 000 \Rightarrow q_3 0001 \Rightarrow q_2 0011 \Rightarrow q_f 011$.

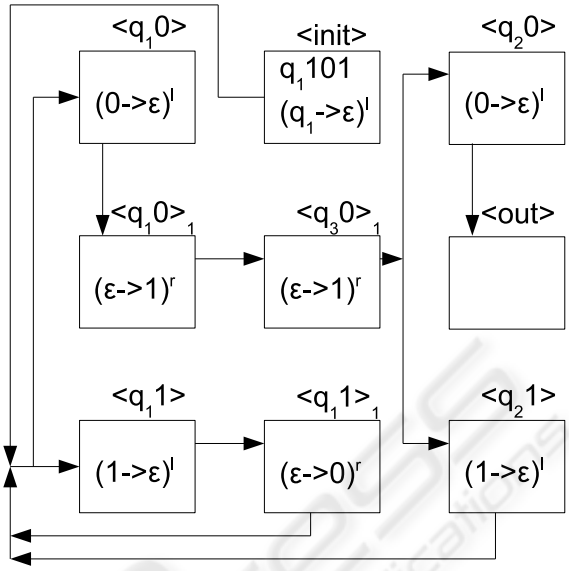

Figure 1: The OHNEP constructed from M.

We now present an OHNEP Γ *constructed from M.*

$$\begin{aligned}
\Gamma &= (V, G, N, C_0, \alpha, \beta, i_0),\\
V &= \{q_1, 0, 1\},\\
C_0(\langle init \rangle) &= \{q_1 W\},\ \textit{where } W \textit{ is the input of } M,\\
C_0(x) &= \emptyset,\ x \in X_G \setminus \{\langle init \rangle\},\\
\beta(x) &= 2,\ x \in X_G,\\
N(x) &= (M_x, \emptyset, \emptyset, \emptyset, \emptyset),\ x \in X_G,
\end{aligned}$$

and G is given in Figure 1, together with $(M_x)^{\alpha(x)}$ *for all nodes* $x \in X_G$ *(we omitted the node* $\langle q_3 0 \rangle$ *because it is not reachable).*

Corollary 3.1 *A family of OHNEPs with obligatory evolutionary processors with empty filters and evolutionary insertion in the right mode and deletion in the left mode (without substitution) is computationally complete.*

Corollary 3.2 *There exists a universal OHNEP with obligatory evolutionary processors with empty filters and evolutionary insertion in the right mode and deletion in the left mode and with 65 nodes.*

Proof. Let us consider the smallest known universal CPM0 P with 6 states and 6 symbol (Alhazov et al., 2002). We add special halt state to the program of this machine in order to stop on every symbol of the machine. So, CPM0 P will be with 7 states and 6 symbols. Now we construct OHNEP P' according algorithm in the theorem above and we get 65 nodes. □

4 CONCLUSIONS

We have considered new variant of Hybrid Network of Evolutionary Processors - Obligatory Hybrid Network of Evolutionary Processors. The differences between them are in underlying graph (undirected graph in HNEP case and directed graph in OHNEP case) and using of operations, OHNEP discards a string if operations at the node are not applicable to the string (in HNEP case the string remains in the node). We showed that OHNEPs with empty input and output filters and insertion operation at the right end and obligatory deletion operation on the left end of the string (without substitution operation) can carry out an universal computation, and there exists universal OHNEP with 65 nodes. Notice, that *structure* of OHNEP (underlying directed graph) and *obligatory* of operation deletion allows to avoid filters and substitution operation and to reach universality. Several questions are opened, in particularly question about computational power of OHNEPs with underlying complete graph and question about universal OHNEP with minimal number of nodes.

ACKNOWLEDGEMENTS

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

REFERENCES

Alhazov, A., Csuhaj-Varjú, E., Martín-Vide, C., and Rogozhin, Y. (2008a). About universal hybrid networks of evolutionary processors of small size. In *Proceedings of the 2nd International Conference on Language and Automata Theory and Applications, LATA 2008*, pages 28–39. Lecture notes in Computer Science 5196, Springer.

Alhazov, A., Csuhaj-Varjú, E., Martín-Vide, C., and Rogozhin, Y. (2008b). Computational completeness of hybrid networks of evolutionary processors with seven nodes. In *Proceedings of DCFS 2008 (10th International Workshop on Descriptional Complexity of Formal Systems)*. Charlottetown, Canada.

Alhazov, A., Kudlek, M., and Rogozhin, Y. (2002). Nine universal circular post machines. *Computer Science Journal of Moldova*, 10 (3):247–262.

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

Alhazov, A., Martín-Vide, C., and Rogozhin, Y. (2007). 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*, pages 521–528. GRLMC report 35/07, Rovira i Virgili University, Tarragona.

Castellanos, J., Martín-Vide, C., Mitrana, V., and Sempere, J. (2001). Solving np-complete problems with networks of evolutionary processors. In *Proceedings of IWANN 2001*, pages 521–528. Lecture Notes in Computer Science 2084, Springer.

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

Kudlek, M. and Rogozhin, Y. (2001a). New small universal circular post machines. In *Proceedings of FCT 2001*, pages 217–227. Lecture Notes in Computer Science 2138, Springer.

Kudlek, M. and Rogozhin, Y. (2001b). Small universal circular post machines. *Computer Science Journal of Moldova*, 9 (1):34–52.

Margenstern, M., Rogozhin, Y., and Verlan, S. (2004). Time-varying distributed h systems with parallel computations: The problem is solved. In *Proceedings of DNA 9*, pages 48–54. Lecture Notes in Computer Science 2943, Springer.

Martín-Vide, C., Mitrana, V., Pérez-Jiménez, M., and Sancho-Caparrini, F. (2003). Hybrid networks of evolutionary processors. In *Proceedings of GECCO 2003*, pages 401–412. Lecture Notes in Computer Science 2723, Springer.