About Universal Hybrid Networks of Evolutionary Processors of Small Size[⋆]

Artiom Alhazov[1,2], Erzsébet Csuhaj-Varjú[3,4], Carlos Martín-Vide[5], and Yurii Rogozhin[5,2]

[1] Åbo Akademi University, Department of Information Technologies, Turku Center for Computer Science, FIN-20520 Turku, Finland
aalhazov@abo.fi

[2] Academy of Sciences of Moldova, Institute of Mathematics and Computer Science, Academiei 5, MD-2028, Chişinău, Moldova
{artiom,rogozhin}@math.md

[3] Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
csuhaj@sztaki.hu

[4] Eötvös Loránd University, Faculty of Informatics, Department of Algorithms and Their Applications, Pázmány Péter sétány 1/c, H-1117 Budapest, Hungary

[5] Rovira i Virgili University, Research Group on Mathematical Linguistics, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
carlos.martin@urv.cat

Abstract. A hybrid network of evolutionary processors (an HNEP) is a graph with a language processor, input and output filters associated to each node. A language processor performs one type of point mutations (insertion, deletion or substitution) on the words in that node. The filters are defined by certain variants of random-context conditions. In this paper, we present a universal complete HNEP with 10 nodes simulating circular Post machines and show that every recursively enumerable language can be generated by a complete HNEP with 10 nodes. Thus, we positively answer the question posed in [5] about the possibility to generate an arbitrary recursively enumerable language over an alphabet V with a complete HNEP of a size smaller than $27 + 3 \cdot card(V)$.

Keywords: Bio-inspired computing, Hybrid networks of evolutionary processors, Small universal systems, Descriptional complexity, Circular Post machines.

⋆ The first author gratefully acknowledges the support by Academy of Finland, project 203667. The fourth author gratefully acknowledges the support of European Commission, project MolCIP, MIF1-CT-2006-021666. The first and the fourth authors acknowledge the Science and Technology Center in Ukraine, project 4032.

1 Introduction

Networks of language processors are finite collections of rewriting systems (language processors) organized in a communicating system [6]. The language processors are located at nodes of a virtual graph and operate over sets or multisets of words. During the functioning of the network, 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 system is usually defined as the set of words which appear at some distinguished node in the course of the computation. One of the main questions related to networks of language processors is how much extent their generative power depends on the used operations and the size of the system. Particularly important variants are those ones where the language processors are based on elementary string manipulating rules, since these constructs give *insight into the limits of the power of the simplicity of basic language theoretic operations and that of distributed architectures.*

Networks of evolutionary processors (NEPs), introduced in [4], and also inspired by cell biology, are proper examples for these types of constructs. In this case, each processor represents a cell performing point mutations of DNA and controlling its passage inside and outside it through a filtering mechanism. The language processor corresponds to the cell, the generated word to a DNA strand, and operations insertion, deletion, or substitution of a symbol to the point mutations. It is known that a computationally universal behaviour emerges as a result of interaction of such simple components (see, for example [1,2]).

In the case of so-called *hybrid networks of evolutionary processors* (HNEPs), each language processor performs only one of the above operations on a certain position of the words in that node. The filters are defined by some variants of random-context conditions. The concept was introduced in [9], and proved computationally complete in [5], with $27 + 3 \cdot card(V)$ nodes for alphabet V.

In this paper, we present a *universal complete HNEP with 10 nodes* and prove that *every recursively enumerable language can be generated by a complete NHEP with the same number of nodes.* Although these bounds are not shown sharp, we significantly improve the previous result. The constructions demonstrate that distributed architectures of very small size, with uniform structure and with components based on very simple language theoretic operations are sufficient to obtain computational completeness.

2 Preliminaries

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 $x \in W^*$.

In our constructions, HNEPs simulate type-0 grammars in Kuroda normal form and Circular Post Machines.

A type-0 grammar in *Kuroda normal form* is a construct $\Gamma = (N, T, S, P)$, where N is the set of nonterminal symbols, T is the set of terminal symbols, N and T are disjoint sets, $S \in N$ is the start symbol, and P is the set of rules of the forms $A \longrightarrow a$, $A \longrightarrow BC$, $A \longrightarrow \varepsilon$, $AB \longrightarrow CD$, where $A, B, C, D \in N$ and $a \in T$. These grammars are known to generate all recursively enumerable languages.

Circular Post Machines (CPMs) were introduced in [7], 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 [8,3] several universal CPMs of variant 0 (CPM0) having small size were constructed, among them in [3] a universal CPM0 with 34 states and 2 symbols. In this article we use the deterministic variant of CPM0s.

A *Circular Post Machine* is a quintuple $(\Sigma, Q, \mathbf{q}_0, \mathbf{q}_f, P)$ 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 of P with all instructions having one of the forms $\mathbf{p}x \rightarrow \mathbf{q}$ (erasing the symbol read), $\mathbf{p}x \rightarrow y\mathbf{q}$ (overwriting and moving to the right), $\mathbf{p}0 \rightarrow y\mathbf{q}0$ (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 so-called *evolutionary operations*, simple rewriting operations abstract local gene mutations.

For an alphabet V, we say that a rule $a \rightarrow b$, 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, deletion, and insertion rules 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 *actions* of σ on w: If $\pi \equiv a \rightarrow b \in Sub_V$, $\rho \equiv a \rightarrow \varepsilon \in Del_V$, and $\sigma \equiv \varepsilon \rightarrow a \in Ins_V$, then

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

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

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

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

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

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

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.

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 [5]) 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 *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. Note that every processor is dedicated to only one type of the above 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 evolutionary processors over V by EP_V.

Definition 1. *A hybrid network of evolutionary processors (an HNEP, 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 an undirected graph with set of vertices X_G and set of edges E_G. G is called the* underlying graph *of the network.*

- *$N : X_G \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the 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 HNEP.*

We say that $card(X_G)$ 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 $C : X_G \longrightarrow 2^{V^*}$ 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 $C'(x) = M_x^{\alpha(x)}(C(x)) \quad \text{for all } x \in X_G$.

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 $C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y) \cap \rho_x(C(y)))$ for all $x \in X_G$.

For an HNEP 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 HNEPs 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 Results

3.1 Universality

Theorem 1. *Any CPM0 P with 2 symbols can be simulated by an HNEP P' with 10 nodes.*

Proof. Let us consider a CPM0 P with two symbols, 0 and 1, and f 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 $q_f \in Q$. Suppose that P stops in the terminal state q_f on every symbol, i.e., there are two instructions $q_f 0 \rightarrow Halt$ and $q_f 1 \rightarrow Halt$. (Notice, that it is easy to transform any CPM0 with n states into a CPM0 with $n+1$ states that stops on every symbol in terminal state.)

So, we consider CPM0 P with instructions of the forms $q_i x \longrightarrow q_j$, $q_i x \longrightarrow y q_j$, $q_i 0 \longrightarrow y q_j 0$, $q_f 0 \longrightarrow Halt$, $q_f 1 \longrightarrow Halt$, where $q_i, q_j \in Q$, $x, y \in \{0,1\}$. A configuration $w = xWq_i$ of CPM0 P describes that P in state $q_i \in Q$ considers symbol $x \in \{0,1\}$ on the left-hand end of $W \in \{0,1\}^*$. Let $I' = I \setminus \{f\}$ and $x, y \in \{0,1\}$. In the following, we construct an HNEP P' simulating P. Starting with the initial configuration W_0 of CPM0 P in node 1 of HNEP P', we simulate every computation step performed by P with a sequence of computation steps in P'. If the computation in P is finite, then the final configuration W_f of P will be found at node 10 of P', moreover, any string that can be found at node 10 is a final configuration of P. In the case of an infinite computation in P, no string will appear in node 10 of P' and the computation in P' will never stop. In the Table 1 below a complete HNEP $P' = (V, G, N, C_0, \alpha, \beta, 10)$ with 10 nodes is described, where (for $i \in I', y \in X$)

$$V = Q \cup Q' \cup T \cup T' \cup S \cup S' \cup R \cup R' \cup X \cup X' \cup X'' \cup \{\hat{0}\} \text{ and}$$
$$Q' = \{q_i'\},\ T = \{t_{i,y}\},\ T' = \{t'_{i,y}\},\ S = \{s_{i,y}\},\ S' = \{s'_{i,y}\},$$
$$R = \{r_i \mid i \in I' \cup \{0\}\},\ R' = \{r_i'\}, X = \{0,1\},\ X' = \{0', 1'\},\ X'' = \{0'', 1''\}.$$

G is a complete graph with 10 nodes, N, C_0, α, β are presented in the Table 1 and node 10 is the output node of HNEP P'. We explain how P' simulates the instructions of CPM0 P. Due to the lack of space, we present only the necessary details.

Instruction $q_i x \longrightarrow q_j$**:** $xWq_i \xrightarrow{P} Wq_j$**.**
The simulation starts with xWq_i in node 1 of P'. By performing evolution steps on this string at node 1, we obtain $xWq_i \xrightarrow{1.1} xWq_i' \xrightarrow{1.2} \{x'Wq_i',\ xW'q_i'\}$, where $W \in \{0,1\}^*$ and $W' \in \{0, 1, 0', 1'\}$. In the following communication step, only strings with q_i' and x' can leave node 1. Notice that strings $xW'q_i'$ do not contain symbols x' on the left-hand end. It is easy to see that during the next transformations it is not possible to delete x' if it is not on the left-hand end of the strings, so these strings will stay forever in node 8. Thus, we will not further consider strings that contain symbols x' not in the correct position. String $x'Wq_i'$ can enter nodes 2 or 3. Let us consider, for example, node 2 (the case for node 3 can be treated analogously). If the string enters node 2, then there exists an instruction $q_i 0 \longrightarrow q_j$ in CPM0 P and $x' = 0'$, so $0'Wq_i' \xrightarrow{2.1} 0'Wq_j$. In the following communication step, string $0'Wq_j$ can enter only node 8, where $0'Wq_j \xrightarrow{8.1} Wq_j$, and then the obtained string, Wq_j, can enter only node 1. So, we simulated instruction $q_i x \longrightarrow q_j$ of P in a correct manner.

In the Table 1 below $i, j \in I',\ x, y \in X,\ x', y' \in X',\ y'' \in X''$.

Table 1.

N, C_0, α, β	M	PI, FI, PO, FO
1, $\{W_0\}$, $*, (2)$	$\{\mathbf{1.1} : q_i \to q_i'\} \cup$ $\{\mathbf{1.2} : x \to x'\} \cup$ $\{\mathbf{1.3} : y'' \to y\} \cup$ $\{\mathbf{1.4} : \hat{0} \to 0\}$	$PI = \emptyset$, $FI = Q' \cup X'' \cup X' \cup \{q_f\}$, $PO = X'$, $FO = Q \cup X'' \cup \{\hat{0}\}$
$2, \emptyset$, $*, (2)$	$\{\mathbf{2.1} : q_i' \to q_j \mid q_i 0 \to q_j\} \cup$ $\{\mathbf{2.2} : q_i' \to t_{j,y} \mid q_i 0 \to yq_j\} \cup$ $\{\mathbf{2.3} : q_i' \to s_{j,y} \mid q_i 0 \to yq_j 0\} \cup$ $\{\mathbf{2.4} : q_i' \to q_f \mid q_i 0 \to q_f\}$	$PI = \{q_i' \mid q_i 0 \to q_j\} \cup$ $\{q_i' \mid q_i 0 \to yq_j\} \cup$ $\{q_i' \mid q_i 0 \to yq_j 0\} \cup$ $\{q_i' \mid q_i 0 \to q_f\}$ $FI = \{1'\}, PO = \{0'\}, FO = Q'$
$3, \emptyset$, $*, (2)$	$\{\mathbf{3.1} : q_i' \to q_j \mid q_i 1 \to q_j\} \cup$ $\{\mathbf{3.2} : q_i' \to t_{j,y} \mid q_i 1 \to yq_j\} \cup$ $\{\mathbf{3.3} : q_i' \to q_f \mid q_i 1 \to q_f\}$	$PI = \{q_i' \mid q_i 1 \to q_j\} \cup$ $\{q_i' \mid q_i 1 \to yq_j\} \cup$ $\{q_i' \mid q_i 1 \to q_f\}$ $FI = \{0'\}, PO = \{1'\}, FO = Q'$

Table 1. (*continued*)

N, C_0, α, β	M	PI, FI, PO, FO
$4, \emptyset, r, (2)$	$\{\mathbf{4.1}: \varepsilon \to r_0\}$	$PI = T \cup S, FI = R \cup R'$, $PO = \{r_0\}, FO = \emptyset$
$5, \emptyset, *, (2)$	$\{\mathbf{5.1}: t_{i,y} \to t'_{i-1,y}$, $\mathbf{5.2}: s_{i,y} \to s'_{i-1,y} \mid 2 \le i \le f-1\} \cup$ $\{\mathbf{5.3}: r_i \to r'_{i+1} \mid 0 \le i \le f-2\} \cup$ $\{\mathbf{5.4}: t_{1,y} \to y\} \cup$ $\{\mathbf{5.5}: s_{1,y} \to y''\}$	$PI = T$, $FI = T' \cup S' \cup R'$, $PO = R'$, $FO = T \cup S \cup\ R$
$6, \emptyset, *, (2)$	$\{\mathbf{6.1}: t'_{i,y} \to t_{i-1,y}$, $\mathbf{6.2}: s'_{i,y} \to s_{i-1,y} \mid 2 \le i \le f-2\} \cup$ $\{\mathbf{6.3}: r'_i \to r_{i+1} \mid 1 \le i \le f-2\} \cup$ $\{\mathbf{6.4}: t'_{1,y} \to y\} \cup$ $\{\mathbf{6.5}: s'_{1,y} \to y''\}$	$PI = T'$, $FI = T \cup S \cup R$, $PO = R$, $FO = T' \cup S' \cup R'$
$7, \emptyset, *, (2)$	$\{\mathbf{7.1}: r_i \to q_i$, $\mathbf{7.2}: r'_i \to q_i \mid i \in I'\}$	$PI = R \cup R', FI = Q' \cup T \cup T' \cup$ $S \cup S', PO = X', FO = R \cup R'$
$8, \emptyset, l, (2)$	$\{\mathbf{8.1}: x' \to \varepsilon\}$	$PI = X', FI = Q' \cup T \cup T' \cup$ $S \cup S' \cup R \cup R', PO = \emptyset, FO = X'$
$9, \emptyset, l, (2)$	$\{\mathbf{9.1}: \varepsilon \to \hat{0}\}$	$PI = X'', FI = R \cup R' \cup X' \cup \{\hat{0}\}$, $PO = \{\hat{0}\}, FO = \emptyset$
$10, \emptyset, *, (2)$	$\emptyset$	$PI = \{q_f\}, FI = V \setminus \{X \cup \{q_f\}\}$, $PO = \emptyset, FO = \{q_f\}$

Instruction $q_i x \longrightarrow y q_j :\ xWq_i \xrightarrow{P} Wyq_j$.
HNEP P' starts the simulation with xWq_i in node 1. Then, two evolution steps follow, $xWq_i \xrightarrow{1.1} xWq'_i \xrightarrow{1.2} \{x'Wq'_i,\ xW'q'_i\}$, where $W \in \{0,1\}^*$ and $W' \in \{0, 1, 0', 1'\}$. Similarly to the previous case, we will consider only string of the form $x'Wq'_i$. This string can enter nodes 2 or 3. Consider, for example, node 2 (the case for node 3 can be treated analogously). If the string enters node 2, then there is an instruction $q_i 0 \longrightarrow yq_j$ in CPM0 P and $x' = 0'$. So, $0'Wq'_i \xrightarrow{2.2} 0'Wt_{j,y}$. String $0'Wt_{j,y}$ can enter nodes 4 and 5. In the latter case the string will stay in node 5 forever, as it does not contain any symbol from R'. Suppose that the string enters node 4. Then, an evolution step, $0'Wt_{j,y} \xrightarrow{4.1} 0'Wt_{j,y}r_0$, follows. Now, string $0'Wt_{j,y}r_0$ can successfully be communicated only to node 5. Then, in nodes 5 and 6, the string is involved in the following evolution steps: For $t \in I'$,

$$0'Wt_{j-t,y}r_t \xrightarrow{5.1} 0'Wt'_{j-(t+1),y}r_t \xrightarrow{5.3} 0'Wt'_{j-(t+1),y}r'_{t+1}$$

$$0'Wt'_{j-(t+1),y}r'_{t+1} \xrightarrow{6.1} 0'Wt_{j-(t+2),y}r'_{t+1} \xrightarrow{6.3} 0'Wt_{j-(t+2),y}r_{t+2}.$$

The string enters in node 5 and 6 in circle, until the first index of t or t' will be decreased to 1. At that moment in node 5 (node 6) index of r (r') will be exactly $j-1$, and it becomes j by rule `5.3` (`6.3`), i.e. the same, as the first index of t in string $0'Wt_{j,y}r_0$ before entering node 5. After that, $0'Wt_{1,y}r'_j \xrightarrow{5.4} 0'Wyr'_j$ or $0'Wt'_{1,y}r_j \xrightarrow{6.4} 0'Wyr_j$. In the following communication step, string $0'Wyr'_j$ or

$0'Wyr_j$ can enter only node 7, where the following evolution steps are performed: $0'Wyr_j \xrightarrow{7.1} 0'Wyq_j$, $0'Wyr'_j \xrightarrow{7.2} 0'Wyq_j$. String $0'Wyq_j$ is communicated to node 8, and this is the only node the string is able to enter. At node 8, evolution step $0'Wyq_j \xrightarrow{8.1} Wyq_j$ can be performed. Now, string Wyq_j can enter only node 1. So, instruction $q_ix \longrightarrow yq_j$ of P is correctly simulated.

Instruction $q_i0 \longrightarrow yq_j0 :\ 0Wq_i \xrightarrow{P} 0Wyq_j$**.**
The beginning of the simulation of instruction $q_i0 \longrightarrow yq_j0$ is the same as that of instruction $q_i0 \longrightarrow yq_j$. The difference appears when rule `2.3` : $q'_i \rightarrow s_{j,y}$ is applied in node 2 instead of rule `2.2` : $q'_i \rightarrow t_{j,y}$ and at the end of the circle process in nodes 5 and 6, $s_{1,y}$ or $s'_{1,y}$ becomes y'' (rules `5.5` or `6.5`) instead of y (rules `5.4` or `6.4`). Strings $0'Wy''r'_j$ or $0'Wy''r_j$ can enter only node 7. Then, either evolution step $0'Wy''r_j \xrightarrow{7.1} 0'Wy''q_j$ or evolution step $0'Wy''r'_j \xrightarrow{7.2} 0'Wy''q_j$ follows. String $0'Wy''q_j$ can enter only node 8, where evolution step $0'Wy''q_j \xrightarrow{8.1} Wy''q_j$ is performed. The new string, $Wy''q_j$, can enter only node 9, where evolution step $Wy''q_j \xrightarrow{9.1} \hat{0}Wy''q_j$ follows. Then string $\hat{0}Wy''q_j$ can enter only node 1, where evolution steps $\hat{0}Wy''q_j \xrightarrow{1.3} \hat{0}Wyq_j \xrightarrow{1.4} 0Wyq_j$ are performed. Thus, instruction $q_i0 \longrightarrow yq_j0$ of P is correctly simulated.

Instruction $q_ix \longrightarrow q_f :\ xWq_i \xrightarrow{P} Wq_f$**.**
In nodes 2 or 3 we have rules $q'_i \rightarrow q_f$ (rules `2.4` or `3.3`) and string $x'Wq'_i$ will be transformed to string $x'Wq_f$. After that it enters node 8 and changes to Wq_f. Now it enters node 10 as a result. 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.

Corollary 1. *There exists a universal HNEP with 10 nodes.*

3.2 Computational Completeness

Theorem 2. *Any recursively enumerable language can be generated by a complete HNEP of size 10.*

Proof. Let $\Gamma = (N, T, S, R)$ be a type-0 grammar in Kuroda normal form.

We construct a complete HNEP $\Gamma' = (V, G, N, C_0, \alpha, \beta, 10)$ of size 10 that simulates the derivations in Γ by the so-called *rotate-and-simulate* method. The rotate-and-simulate method means that the words found in the nodes are involved into either the rotation of the leftmost symbol (the leftmost symbol of the word is moved to the end of the word) or the simulation of a rule of R. To guarantee the correct simulation, a marker symbol, #, is introduced for indicating the end of the simulated word under the rotation. Assume that the symbols $N \cup T \cup \{\#\}$ are labeled in a one-to-one manner by $1, 2, \ldots, n$. More precisely let $N \cup T \cup \{\#\} = A = \{A_1, A_2, \ldots A_n\}$, $I = \{1, 2, \ldots, n\}$, $I' = \{1, 2, \ldots, n-1\}$, $I'' = \{2, 3 \ldots, n\}$, $I_0 = \{0, 1, 2, \ldots, n\}$, $I'_0 = \{0, 1, 2, \ldots, n-1\}$, $B_0 = \{B_{j,0} \mid j \in I\}$, $\# = A_n$, $T' = T \cup \#$. The alphabet V of the network is defined as follows:

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

G is a complete graph with 10 nodes, N, C_0, α, β are presented in Table 2 below and node 10 is the output node of HNEP Γ'.

A configuration of grammar Γ is a word $w \in \{N \cup T\}^*$. Each configuration w of Γ corresponds to a configuration $wB_{n,0}$ and configurations $w''A_nw'B_{i,0}$ of HNEP Γ', where $A_n = \#$, $w, w', w'' \in (N \cup T)^*$ and $w = w'A_iw''$.

The axiom $S = A_1$ of Γ corresponds to an initial word $A_1\#$, represented as $A_1B_{n,0}$ in node 1 of HNEP Γ'. Now we describe the how the rotation of a symbol and the application of an arbitrary rule of grammar Γ are simulated in Γ'. As above, due to the lack of space, we present only the necessary details.

Rotation
Let $A_{i_1}A_{i_2}\dots A_{i_{k-1}}B_{i_k,0}$ be found at node 1, and let $w, w', w'' \in A^*$. Then, by evolution, $A_{i_1}A_{i_2}\dots A_{i_{k-1}}B_{i_k,0} = A_{i_1}wB_{i_k,0} \xrightarrow{1.1} \{C_{i_1}wB_{i_k,0},\ A_{i_1}w'C_{i_t}w''B_{i_k,0}\}$ follows. Notice that during the simulation symbols C_i should be transformed to ε', and this symbol should be deleted from the left-hand end of the string (node 9). So, transformation of string $A_{i_1}w'C_{i_t}w''B_{i_k,0}$ leads to a string that will stay in node 9 forever; thus, in the sequel, we will not consider strings with C_i not in the leftmost position. In the following communication step, string $C_{i_1}wB_{i_k,0}$ can enter only node 2. Then, in nodes 2 and 3 the string is involved in evolution steps followed by communication as follows:

$$C_{i_1-t}wB_{i_k,t} \xrightarrow{2.1} C'_{i_1-(t+1)}wB_{i_k,t} \xrightarrow{2.2} C'_{i_1-(t+1)}wB'_{i_k,t+1} \text{ (in node 2)},$$
$$C'_{i_1-t}wB'_{i_k,t} \xrightarrow{3.1} C_{i_1-(t+1)}wB'_{i_k,t} \xrightarrow{3.2} C_{i_1-(t+1)}wB_{i_k,t+1} \text{(in node 3)}.$$

The process continues in nodes 2 and 3 until index of C_i or C'_i will be decreased to 1. In this case rule `2.3` : $C_1 \to \varepsilon'$ in node 2 or `3.3` : $C'_1 \to \varepsilon'$ in node 3 will be applied and string $\varepsilon'wB'_{i_k,i_1}$ or $\varepsilon'wB_{i_k,i_1}$ appears in node 4. Then, in node 4, either evolution step $\varepsilon'wB'_{i_k,i_1} \xrightarrow{4.1} \varepsilon'wB'_{i_k,i_1}D_0$ or evolution step $\varepsilon'wB_{i_k,i_1} \xrightarrow{4.1} \varepsilon'wB_{i_k,i_1}D_0$ is performed. Strings $wB'_{i_k,i_1}D_0$ or $wB_{i_k,i_1}D_0$ can enter only node 5, where either evolution step $\varepsilon'wB'_{i_k,i_1}D_0 \xrightarrow{5.1} \varepsilon'wE_{i_k,i_1}D_0$ or evolution step $\varepsilon'wB_{i_k,i_1}D_0 \xrightarrow{5.2} \varepsilon'wE_{i_k,i_1}D_0$ follows. String $\varepsilon'wE_{i_k,i_1}D_0$ can enter only node 6. Then, in nodes 6 and 7 the string is involved in evolution steps followed by communication as follows:

$$\varepsilon'wE_{i_k,i_1-t}D_t \xrightarrow{6.1} \varepsilon'wE'_{i_k,i_1-(t+1)}D_t \xrightarrow{6.2} \varepsilon'wE'_{i_k,i_1-(t+1)}D'_{t+1} \text{ (in node 6)},$$
$$\varepsilon'wE'_{i_k,i_1-t}D'_t \xrightarrow{7.1} \varepsilon'wE_{i_k,i_1-(t+1)}D'_t \xrightarrow{7.2} \varepsilon'wE_{i_k,i_1-(t+1)}D_{t+1} \text{ (in node 7)}.$$

The process continues in nodes 6 and 7 until second index of $E_{i,j}$ or that of $E'_{i,j}$ will be decreased to 1. In this case, rule `6.3` : $E_{i_k,1} \to A_{i_k}$ in node 6 or `7.3` : $E'_{i_k,1} \to A_{i_k}$ in node 7 will be applied and string $\varepsilon'wA_{i_k}D'_{i_1}$ or $\varepsilon'wA_{i_k}D_{i_1}$ appears in node 8.

Notice, that rule `6.4`: $A_n \to \varepsilon'$ can be applied. This case is discussed below. The next evolution step, performed in node 8, can either be $\varepsilon' w A_{i_k} D_{i_1} \xrightarrow{8.1} \varepsilon' w A_{i_k} B_{i_1,0}$ or $\varepsilon' w A_{i_k} D'_{i_1} \xrightarrow{8.2} \varepsilon' w A_{i_k} B_{i_1,0}$. In the following communication step, string $\varepsilon' w A_{i_k} B_{i_1,0}$ can enter node 9 or node 6.

1 Consider the last case (in this case $A_{i_1} \in T$).
At nodes 6, 9 and 10 the following evolution and communication steps are performed:

- Suppose that word $wA_{i_k}B_{i_1,0}$ does not contain nonterminal symbols (except A_n). Let $wA_{i_k}B_{i_1,0} = A_n w' A_{i_k} B_{i_1,0}$, where $w = A_n w'$. So, $w'A_{i_k}A_{i_1}$ is a result and it has appear in node 10. Notice, that if $w = w'A_n w''$ and $w' \neq \varepsilon$, then word $\varepsilon' w' A_n w'' A_{i_k} B_{i_1,0}$ leads to a word which will stay in node 9 forever (if rule `6.4` was applied) or will leave node 9 as word $w'A_n w'' A_{i_k} A_{i_1}$ and enter node 1, and will remain there forever. So, we will consider the following evolution of the word $\varepsilon' w A_{i_k} B_{i_1,0} = \varepsilon' A_n w' A_{i_k} B_{i_1,0}$: $\varepsilon' A_n w' A_{i_k} B_{i_1,0} \xrightarrow{6.5} \varepsilon' A_n w' A_{i_k} A_{i_1} \xrightarrow{6.4} \varepsilon'\varepsilon' w' A_{i_k} A_{i_1}$. Further, string $\varepsilon'\varepsilon' w' A_{i_k} A_{i_1}$ appears in node 9, where symbols ε' will be eliminated by rule `9.1` and, finally, word $w' A_{i_k} A_{i_1}$ enters node 10. This is a result.

  In the case of applying only rule `6.5`, the resulting word $\varepsilon' A_n w' A_{i_k} A_{i_1}$ appears in node 9, where it becomes $A_n w' A_{i_k} A_{i_1}$, leaves node 9, enters node 1 and stays there forever.
- Suppose that word $wA_{i_k}B_{i_1,0}$ contains at least one nonterminal symbol (except A_n). In node 6 symbol $B_{i_1,0}$ is changed to A_{i_1}, after that the resulting word appears in node 1, where it will stay forever, since the output filter requires symbols from B_0.

2 Now consider the evolution of the word $\varepsilon' w A_{i_k} B_{i_1,0}$ in node 9. By applying the corresponding rules, we obtain $\varepsilon' w A_{i_k} B_{i_1,0} \xrightarrow{9.1} w A_{i_k} B_{i_1,0}$. Then, string $wA_{i_k}B_{i_1,0}$ enters node 1 and the rotation of a symbol is over. If $A_{i_1} \in T$, then the string can enter node 6. This case was considered above.

Table 2.

$N, \alpha, \beta, C_0,$	M	PI, FI, PO, FO
$1, *, (2),$ $\{A_1 B_{n,0}\}$	$\{\mathbf{1.1}: A_i \to C_i \mid i \in I,\ rotation\} \cup$ $\{\mathbf{1.2}: A_i \to \varepsilon' \mid i \in I',\ A_i \to \varepsilon\} \cup$ $\{\mathbf{1.3}: B_{j,0} \to B_{s,0} \mid A_j \to A_s,\ j,s \in I'\}$	$PI = \{A_n, B_{n,0}\},$ $FI = C \cup C' \cup \{\varepsilon'\},$ $PO = B_0, FO = \emptyset$
$2, *, (2), \emptyset$	$\{\mathbf{2.1}: C_i \to C'_{i-1},$ $\mathbf{2.2}: B_{j,k} \to B'_{j,k+1} \mid$ $i \in I'',\ j \in I,\ k \in I'_0\} \cup$ $\{\mathbf{2.3}: C_1 \to \varepsilon'\}$	$PI = C,$ $FI = C' \cup B' \cup \{\varepsilon'\},$ $PO = C' \cup \{\varepsilon'\},$ $FO = C \cup B$
$3, *, (2), \emptyset$	$\{\mathbf{3.1}: C'_i \to C_{i-1},$ $\mathbf{3.2}: B'_{j,k} \to B_{j,k+1} \mid$ $i \in I'',\ j \in I,\ k \in I'_0\} \cup$ $\{\mathbf{3.3}: C'_1 \to \varepsilon'\}$	$PI = C',$ $FI = C \cup B \cup \{\varepsilon'\},$ $PO = C \cup \{\varepsilon'\},$ $FO = C' \cup B'$
$4, r, (2), \emptyset$	$\{\mathbf{4.1}: \varepsilon \to D_0\}$	$PI = B \setminus B_0 \cup B',$ $FI = C \cup C' \cup B_0 \cup \{D_0\},$ $PO = \{D_0\}, FO = \emptyset$

Table 2. (*continued*)

$N, \alpha, \beta, C_0,$	M	PI, FI, PO, FO
$5, *, (2), \emptyset$	$\{\mathbf{5.1}: B_{j,k} \to E_{j,k},$ $\mathbf{5.2}: B'_{j,k} \to E_{j,k} \mid j,k \in I, rotation\} \cup$ $\{\mathbf{5.3}: B_{j,k} \to E_{s,t},$ $\mathbf{5.4}: B'_{j,k} \to E_{s,t} \mid$ $j,k,s,t \in I', A_jA_k \to A_sA_t\}$	$PI = \{D_0\},$ $FI = \emptyset,$ $PO = E,$ $FO = B \cup B'$
$6, *, (2), \emptyset$	$\{\mathbf{6.1}: E_{j,k} \to E'_{j,k-1},$ $\mathbf{6.2}: D_i \to D'_{i+1},$ $\mathbf{6.3}: E_{j,1} \to A_j \mid i \in I'_0, j \in I, k \in I''\} \cup$ $\{\mathbf{6.4}: A_n \to \varepsilon'\} \cup$ $\{\mathbf{6.5}: B_{j,0} \to A_j \mid A_j \in T\}$	$PI = E \cup \{B_{j,0} \mid A_j \in T\},$ $FI = E' \cup D' \cup C,$ $PO = D' \cup \{\varepsilon'\},$ $FO = E \cup D \cup$ $\{B_{j,0} \mid A_j \in T\}$
$7, *, (2), \emptyset$	$\{\mathbf{7.1}: E'_{j,k} \to E_{j,k-1},$ $\mathbf{7.2}: D'_i \to D_{i+1},$ $\mathbf{7.3}: E'_{j,1} \to A_j \mid i \in I', j \in I, k \in I''\}$	$PI = E', FI = E \cup D,$ $PO = D, FO = E' \cup D'$
$8, *, (2), \emptyset$	$\{\mathbf{8.1}: D_j \to B_{j,0},$ $\mathbf{8.2}: D'_j \to B_{j,0} \mid j \in I\} \cup$ $\{\mathbf{8.3}: D_j \to B_{s,t},$ $\mathbf{8.4}: D'_j \to B_{s,t} \mid A_j \to A_sA_t, j,s,t \in I'\}$	$PI = D \setminus \{D_0\} \cup D',$ $FI = E \cup E' \cup \{D_0\},$ $PO = \emptyset,$ $FO = D \cup D'$
$9, l, (2), \emptyset$	$\{\mathbf{9.1}: \varepsilon' \to \varepsilon\}$	$PI = \{\varepsilon'\},$ $FI = B \setminus B_0 \cup$ $B' \cup D \cup D',$ $PO = \emptyset, FO = \{\varepsilon'\}$
$10, *, (2), \emptyset$	$\emptyset$	$PI = T, FI = V \setminus T,$ $PO = \emptyset, FO = T$

Rule $A_i \longrightarrow \varepsilon$. Suppose that $A_iwB_{j,0}$ can be found at node 1 and let $w, w', w'' \in A^*$. Then, by evolution, either $A_iwB_{j,0} \xrightarrow{1.2} \varepsilon'wB_{j,0}$ or $A_tw'A_iw''B_{j,0} \xrightarrow{1.2} A_iw'\varepsilon'w''B_{j,0}$. String $\varepsilon'wB_{j,0}$ or $A_iw'\varepsilon'w''B_{j,0}$ can enter node 9 or node 6 (considered above). String $A_iw'\varepsilon'w''B_{j,0}$ will stay in node 9 forever. So, we will consider the transformation of only string $\varepsilon'wB_{j,0}$. At node 9, evolution step $\varepsilon'wB_{j,0} \xrightarrow{9.1} wB_{j,0}$ follows. Now, string $wB_{j,0}$ enters node 1. Thus, we correctly simulated rule $A_i \longrightarrow \varepsilon$ of grammar Γ.

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 $wB_{j,0}$ now is in node 1, we simulated the rule $A_i \longrightarrow A_j$ of grammar Γ 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 8 instead of applying rule $D_j \to B_{j,0}$ ($D'_j \to B_{j,0}$) rule $D_j \to B_{s,t}$ ($D'_j \to B_{s,t}$) will be applied. Then, at node 8 either evolution step $\varepsilon'wD_j \xrightarrow{8.3} \varepsilon'wB_{s,t}$ or evolution step $\varepsilon'wD'_j \xrightarrow{8.4} \varepsilon'wB_{s,t}$ is performed. Then, string $\varepsilon'wB_{s,t}$ can enter only node 4, where, by evolution, $\varepsilon'wB_{s,t} \xrightarrow{4.1} \varepsilon'wB_{s,t}D_0$. The process continues as above, in the case of simulating rotation, so, in several computation steps string $wA_sB_{t,0}$ will be obtained in node 9 which then successfully is communicated to node 1. So, we correctly simulated rule $A_j \longrightarrow A_s A_t$ of grammar Γ.

Rule $A_i A_j \longrightarrow A_s A_t$**.** In node 5 there are rules `5.3` : $B_{i,j} \rightarrow E_{s,t}$ or `5.4` : $B'_{i,j} \rightarrow E_{s,t}$. As in the case of simulating rotation, above, we will obtain string $wA_sB_{t,0}$ in node 9.

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

Corollary 2. *The class of complete HNEPs with 10 nodes is computationally complete.*

4 Conclusions

We have presented a universal complete HNEP with 10 nodes and proved that complete HNEPs with 10 nodes generate all recursively enumerable languages. Thus, we positively answered question 1 from [5] and significantly improved the results of that paper.

References

1. Alhazov, A., Martín-Vide, C., Rogozhin, Y.: On the number of nodes in universal networks of evolutionary processors. Acta Informatica 43(5), 331–339 (2006)
2. Alhazov, A., Martín-Vide, C., Rogozhin, Y.: Networks of Evolutionary Processors with Two Nodes Are Unpredictable. In: Pre-Proceedings of the 1st International Conference on Language and Automata Theory and Applications, LATA 2007, GRLMC report 35/07, Rovira i Virgili University, Tarragona, Spain, pp.521–528 (2007)
3. Alhazov, A., Kudlek, M., Rogozhin, Y.: Nine Universal Circular Post Machines. Computer Science Journal of Moldova 10(3), 247–262 (2002)
4. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.: Solving NP-complete problems with networks of evolutionary processors. In: Mira, J., Prieto, A. (eds.) IWANN 2001. LNCS, vol. 2084. Springer, Heidelberg (2001)
5. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V.: Hybrid networks of evolutionary processors are computationally complete. Acta Informatica 41(4-5), 257–272 (2005)
6. Csuhaj-Varjú, E., Salomaa, A.: Networks of Parallel Language Processors. In: Păun, G., Salomaa, A. (eds.) New Trends in Formal Languages. Control, Cooperation, and Combinatorics. LNCS, vol. 1218, pp. 299–318. Springer, Heidelberg (1997)
7. Kudlek, M., Rogozhin, Y.: Small Universal Circular Post Machines. Computer Science Journal of Moldova 9(1), 34–52 (2001)
8. Kudlek, M., Rogozhin, Y.: New Small Universal Circular Post Machines. In: Freivalds, R. (ed.) FCT 2001. LNCS, vol. 2138, pp. 217–227. Springer, Heidelberg (2001)
9. Martín-Vide, C., Mitrana, V., Perez-Jimenez, M., Sancho-Caparrini, F.: Hybrid networks of evolutionary processors. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 401–412. Springer, Heidelberg (2003)