Computational Completeness of Hybrid Networks of Evolutionary Processors with Seven Nodes

Artiom Alhazov\(^{(A)}\) Erzsébet Csuhaj-Varjú\(^{(B)}\) Carlos Martín-Vide\(^{(C)}\) Yurii Rogozhin\(^{(D)}\)

\(^{(A)}\)Academy of Sciences of Moldova, – Institute of Mathematics and Computer Science, – Academiei 5, MD-2028, Chișinău, Moldova – artiom@math.md (Art)

\(^{(B)}\)Computer and Automation Research Institute, – Hungarian Academy of Sciences – Kende u. 13-17, 1111 Budapest, Hungary – csuhaj@sztaki.hu (Erzs)

\(^{(B)}\)Eötvös Loránd University, – Faculty of Informatics, Department of Algorithms and Their Applications, – Pázmány Péter sétány 1/c, H-1117 Budapest, Hungary –

\(^{(C)}\)Rovira i Virgili University, – Research Group on Mathematical Linguistics, – Pl. Imperial Tarraco 1, 43005 Tarragona, Spain – carlos.martin@urv.cat (Carl)

\(^{(D)}\)Academy of Sciences of Moldova, – Institute of Mathematics and Computer Science, – Academiei 5, MD-2028, Chișinău, Moldova – rogozhin@math.md (Yu)

\(^{(D)}\)Rovira i Virgili University, – Research Group on Mathematical Linguistics, – Pl. Imperial Tarraco 1, 43005 Tarragona, Spain –

Abstract. A hybrid network of evolutionary processors (an HNEP) is a graph where each node is associated with a language processor (an evolutionary processor), a set of words, an input filter and an output filter. The evolutionary 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, by improving our previous result, we prove that HNEPs with 7 nodes are able to generate any recursively enumerable language. We also show that the family of HNEPs with 2 nodes is not computationally complete.

Keywords: Hybrid networks of evolutionary processors, small universal systems, size complexity

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 \([6]\), are proper

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.
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 [3, 4]).

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 [10]. In [7] it was shown that, for an alphabet \( V \), HNEPs with \( 27 + 3 \cdot \text{card}(V) \) nodes are computationally complete. A significant improvement of the result can be found in [1], where it was proved that HNEPs with 10 nodes (irrespective of the size of the alphabet) reach the universal power.

In this paper, we improve our previous result by proving that any recursively enumerable language can be generated by an HNEP having 7 nodes. We also show that the family of HNEPs with 2 nodes is not computationally complete. Although the sharpness of the upper bound in our paper is not proven, we considerably improved the previous result. The gap between universality and non-universality for HNEPs now is very small (it is the same as for famous PCP problem [11]). The construction demonstrates 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 Basic notions

We first recall some basic notions we shall use in the paper. An alphabet is a finite and nonempty set of symbols. The cardinality of a finite set \( A \) is denoted by \( \text{card}(A) \). 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 \( \varepsilon \); 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 nonempty word \( x \), \( \text{alph}(x) \) denotes the smallest alphabet \( W \) such that \( x \in W^* \).

To prove the main result, we simulate type-0 grammars given in Kuroda normal form with hybrid networks of evolutionary processors.

A type-0 grammar in Kuroda normal form is a construct \( G = (N, T, S, P) \), 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 \( A \rightarrow a, A \rightarrow BC, A \rightarrow \varepsilon, AB \rightarrow CD \), where \( A, B, C, D \in N \) and \( a \in T \). It is well-known that the class of languages generated by type-0 grammars in Kuroda normal form is equal to the class of recursively enumerable languages.

In the sequel, following the terminology in [7], we recall the necessary notions concerning evolutionary processors and their hybrid networks. These language processors use so-called evolutionary operations, simple rewriting operations which abstract local gene mutations.

For an alphabet \( V \), we say that a rule \( a \rightarrow b \), with \( a, b \in V \cup \{\varepsilon\} \) is a substitution rule if both \( a \) and \( b \) are different from \( \varepsilon \); 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 \( \text{Sub}_V, \text{Del}_V, \) and \( \text{Ins}_V \), respectively. Given such rules \( \pi, \rho, \sigma, \) and a word \( w \in V^* \), we define the following actions of \( \sigma \) on \( w \): If \( \pi \equiv a \rightarrow b \in \text{Sub}_V, \rho \equiv a \rightarrow \varepsilon \in \text{Del}_V, \) and \( \sigma \equiv \varepsilon \rightarrow a \in \text{Ins}_V, \) then

\[
\begin{align*}
\pi^*(w) &= \begin{cases} \{wbv : \exists u, v \in V^*(w = uav)\}, & \text{otherwise} \\ \{w\}, & \text{otherwise} \end{cases} \\
\rho^*(w) &= \begin{cases} \{uwv : \exists u, v \in V^*(w = uav)\}, & \text{otherwise} \\ \{w\}, & \text{otherwise} \end{cases} \\
\rho^l(w) &= \begin{cases} \{u : w = ua\}, & \text{otherwise} \\ \{w\}, & \text{otherwise} \end{cases} \\
\rho^r(w) &= \begin{cases} \{v : w = av\}, & \text{otherwise} \\ \{w\}, & \text{otherwise} \end{cases} \\
\sigma^*(w) &= \{uav : \exists u, v \in V^*(w = uv)\}, \\
\sigma^l(w) &= \{wa\}, \sigma^r(w) = \{aw\}.
\end{align*}
\]

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 \( \sigma \), action \( \alpha \in \{*, l, r\} \), and \( L \subseteq V^* \), we define the \( \alpha \)–action of \( \sigma \) 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:

\[
\varphi^{(1)}(w; P, F) \equiv P \subseteq \text{alph}(w) \land F \cap \text{alph}(w) = \emptyset
\]

and

\[
\varphi^{(2)}(w; P, F) \equiv \text{alph}(w) \cap P \neq \emptyset \land F \cap \text{alph}(w) = \emptyset.
\]
The construction of these predicates is based on random-context conditions defined by the two sets $P$ (permitting contexts) and $F$ (forbidding contexts).

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

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

- Either $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$. The set $M$ represents the set of evolutionary rules of the processor. Notice that every processor is dedicated to only one type of the evolutionary operations.
- $PI, FI \subseteq V$ are the input permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the output permitting/forbidding contexts of the processor.

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

**Definition 1.** A hybrid network of evolutionary processors (an HNEP, shortly) is a 7-tuple $\Gamma = (V, G, \mathcal{N}, C_0, \alpha, \beta, i_0)$, 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.
- $\mathcal{N} : X_G \rightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $C_0 : X_G \rightarrow 2^{V^*}$ is a mapping which identifies the initial configuration of the network. It associates a finite set of words with each node of the graph $G$.
- $\alpha : X_G \rightarrow \{*, l, r\}$; $\alpha(x)$ defines the action mode of the rules performed in node $x$ on the words occurring in that node.
- $\beta : X_G \rightarrow \{(1), (2)\}$ 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 $\mu_x(\cdot) = \varphi^{\beta(x)}(\cdot, PI_x, FI_x)$, and the output filter is defined as $\tau_x(\cdot) = \varphi^{\beta(x)}(\cdot, PO_x, FO_x)$. That is, $\mu_x(w)$ (resp. $\tau_x$) 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 $x$.
- $i_0 \in X_G$ is the output node of $\Gamma$.

We say that $\text{card}(X_G)$ is the size of $\Gamma$. An HNEP is said to be a complete HNEP, if its underlying graph is a complete graph.

A configuration of an HNEP $\Gamma$, as above, is a mapping $C : X_G \rightarrow 2^{V^*}$ 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 step of a 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 $C \Longrightarrow C'$, if $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_G$.

When the configuration changes by a communication step, then each language processor $\mathcal{N}(x)$, 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 \( x \). 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

\[
C'(x) = (C(x) - \tau_x(C(x))) \cup \{ (x,y) \in E_G \mid \tau_y(C(y)) \cap \mu_x(C(y)) \}
\]

for all \( x \in X_G \).

For an HNEP \( \Gamma \), a computation in \( \Gamma \) is a sequence of configurations \( C_0, C_1, C_2, \ldots \), where \( C_0 \) is the initial configuration of \( \Gamma \), \( C_{2i} \Rightarrow C_{2i+1} \) and \( C_{2i+1} \vdash C_{2i+2} \), for all \( i > 0 \). If we consider HNEPs as language generating devices, then 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 \( \Gamma \) is

\[
L(\Gamma) = \bigcup_{s \geq 0} C_s(i_0).
\]

### 3 Computational Completeness

It is known from [5] that there exists an HNEP with two nodes that generates a non-context-free context-sensitive language. We first show that to obtain computational completeness, HNEPs need at least three nodes.

For an alphabet \( V \), let us consider a morphism \( h_2 \) defined by \( h_2(a) = aa \), \( a \in V \) and let \( L_2 = h_2(V^+) = \{ aa \mid a \in V \}^+ \).

**Lemma 2.** Consider a HNEP \( \Gamma \) with an output node \( N_i \). Let us assume that \( L(\Gamma) = L_i(\Gamma) \subseteq L_2 \). Then, for any step \( 2s \), where \( s \geq 1 \), for any word \( w \in C_{2s}(i) \), and for any operation \( \sigma \in M_i \) it holds that either \( \sigma \) is not applicable to \( w \) or \( \sigma \) is the identical substitution of symbol \( a \).

**Proof:** Suppose that \( \sigma \) is an operation different from the identical substitution of symbol \( a \). Then \( w \in C_{2s}(i) \subseteq L_i(\Gamma) \) and \( \sigma(w) \in C_{2s+1}(i) \subseteq L_i(\Gamma) \), where \( w \neq \sigma(w) \). However, \( w \) and \( \sigma(w) \) differ from each other by only one insertion, deletion or a non-identical substitution of \( a \), which is impossible for words from \( L_2 \).

This statement implies that an HNEP is able to generate any sublanguage of \( L_2 \) only if the output node does not perform any operation different from the identical substitution of \( a \).

**Theorem 3.** The family of HNEPs with two nodes is not computationally complete.

**Proof:** We recall from [7] that HNEPs without insertion nodes generate only finite languages and HNEPs without deletion nodes generate only context-sensitive languages. Then, by Lemma 2, any word arriving in the output node of an HNEP generating a sublanguage of \( L_2 \) should remain unchanged, which means that HNEPs generating any language \( L \subseteq L_2 \) which is not context-sensitive should have at least three nodes.
Theorem 4. Any recursively enumerable language can be generated by a complete HNEP of size 7.

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 $\Gamma = (V, H, N, C_0, \alpha, \beta, 7)$ 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 $N \cup T \cup \{\#\} = A = \{A_1, A_2, \ldots, A_n\}$, $I = \{1, 2, \ldots, n\}$, $I' = \{1, 2, \ldots, n-1\}$, $I'' = \{2, 3, \ldots, n\}$, $I_0 = \{0, 1, 2, \ldots, n\}$, $I_0' = \{0, 1, 2, \ldots, n-1\}$, $B_0 = \{B_{j,0} | j \in I\}$, $B_0' = \{B'_{j,0} | j \in I\}$, $\# = A_n$, $T' = T \cup \#$. Let us define the alphabet $V$ of $\Gamma$ as follows:

$$V = A \cup B \cup B' \cup C \cup C' \cup D \cup D' \cup E \cup E' \cup F \cup G \cup \{\varepsilon\},$$

where

$$B = \{B_{i,j} | i \in I, j \in I_0\}, B' = \{B'_{i,j} | i \in I, j \in I_0\}, C = \{C_i | i \in I\},$$

$$C' = \{C'_i | i \in I\}, D = \{D_i | i \in I_0\}, D' = \{D'_i | i \in I\},$$

$$E = \{E_{i,j} | i, j \in I\}, E' = \{E'_{i,j} | i, j \in I\}, F = \{F_j | j \in I\},$$

$$G = \{G_{i,j} | i, j \in I\}.$$  

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

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 $\Gamma$ in node 1 and having one of the forms $wB_{1,0}$ or $w^\prime w A_n^\prime w B_{1,0}$, where $A_n = \#$, $w, w^\prime, w'' \in (N \cup T)^*$ and $w = w' A_1' w''$. The start symbol $S = A_1$ of $G$ corresponds to an initial word $A_1^\#$, represented as $A_1 B_{1,0}$ in node 1 of HNEP $\Gamma$, 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 $\Gamma$, 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 $\Gamma$.

Rotation.

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

We note that during the simulation symbols $C'_i$ should be transformed to $\varepsilon'$, and this symbol can only be deleted from the left-hand end of the string (node 6). So, the replacement of $C_{i_0}$ by its primed version in a string of the form $A_{i_1} w' C_{i_0} w'' B_{i_k,0}$ 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 \( C_i' w B_{i_k,0} \) cannot leave node 1 and stays there for the next evolution step:

\[
C_i' w B_{i_k,0} \xrightarrow{1.5} C_i' w B'_{i_k,0}.
\]

<table>
<thead>
<tr>
<th>( N, \alpha, \beta, C_i0 )</th>
<th>( M )</th>
<th>( PI, FI, PO, FO )</th>
</tr>
</thead>
</table>
| 1, \( s, (2), \{ A_1B_{n,0} \} \) | \{1.1 : \( A_i \rightarrow C_i' \mid i \in I \} \cup \{1.2 : \( A_i \rightarrow \varepsilon' \mid i \in I', A_i \rightarrow \varepsilon \} \cup \{1.3 : B_{j,0} \rightarrow B_{j,0} \mid A_j \rightarrow A_s, j, s \in I' \} \cup \{1.4 : C_i \rightarrow C_{i-1} \} \cup \{1.5 : B_{j,0} \rightarrow B'_{j,0} \} \cup \{1.6 : B_{j,k} \rightarrow B'_{j,k+1} \mid i \in I', j \in I, k \in I' \} \cup \{1.7 : C_i \rightarrow \varepsilon' \} \cup \{1.8 : E'_{j,k} \rightarrow E_{j,k-1} \} \cup \{1.9 : D_{i} \rightarrow D_{i+1} \} \cup \{1.10 : E'_{i,j} \rightarrow F_{j} \mid i \in I', j \in I, k \in I' \} \} \cup \{2.1 : C_i \rightarrow C_{i-1} \} \cup \{2.2 : B'_{j,k} \rightarrow B_{j,k+1} \mid i \in I', j \in I, k \in I' \} \cup \{2.3 : C_i' \rightarrow C_{i-1}' \} \cup \{2.4 : E'_{j,k} \rightarrow E_{j,k-1}' \} \cup \{2.5 : C_i \rightarrow C_{i+1}' \} \cup \{2.6 : E_{j,0} \rightarrow F_{j} \mid i \in I_0, j \in I, k \in I' \} \cup \{2.7 : A_n \rightarrow \varepsilon' \} \cup \{2.8 : B_{j,0} \rightarrow A_1 \mid A_j \rightarrow T \} \} | \begin{align*}
PI &= \{ A_n, B_{n,0} \} \cup C \cup E' \\
FI &= \{ B' \} \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'
\end{align*} |

| 2, \( s, (2), \emptyset \) | \{3.1 : \( \varepsilon \rightarrow D_0 \} \} | \begin{align*}
PI &= \{ B_{n,0} \mid A_j \rightarrow T \} \cup C' \cup E' \\
FI &= \{ B' \} \cup E \cup D' \cup F \cup G \cup \{ \varepsilon' \} \\
PO &= B' \cup C' \cup \{ \varepsilon' \} \\
FO &= B \cup D \cup E
\end{align*} |

| 3, \( r, (2), \emptyset \) | \{4.1 : B_{j,k} \rightarrow E_{j,k} \} \cup \{4.2 : B_{j,k} \rightarrow E_{j,k} \mid j, k \in I \} \cup \{4.3 : B_{j,k} \rightarrow E_{j,s,t} \} \cup \{4.4 : B_{j,k} \rightarrow E_{s,t} \mid j, k \in I', A_j A_k \rightarrow A_s A_t \} \cup \{4.5 : G_{j,k} \rightarrow E_{j,k} \mid j, k \in I' \} \} | \begin{align*}
PI &= \{ D_0 \} \\
FI &= E \\
PO &= E \\
FO &= B \cup B' \cup G
\end{align*} |

| 4, \( s, (2), \emptyset \) | \{5.1 : D_j \rightarrow B_{j,0} \} \cup \{5.2 : D_j \rightarrow B_{j,0} \mid j \in I \} \cup \{5.3 : E_j \rightarrow A_j \mid j \in I \} \cup \{5.4 : D_j \rightarrow G_{s,t} \} \cup \{5.5 : D_j \rightarrow G_{s,t} \mid A_j \rightarrow A_s A_t, j, s, t \in I' \} \} | \begin{align*}
PI &= D \cup \{ 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
\end{align*} |

| 5, \( s, (2), \emptyset \) | \{6.1 : \( \varepsilon \rightarrow \varepsilon \} \} | \begin{align*}
PI &= \{ \varepsilon' \} \\
FI &= B \setminus \{ D_0 \} \cup D' \\
PO &= \emptyset \\
FO &= \{ \varepsilon' \}
\end{align*} |

| 6, \( s, (2), \emptyset \) | \{7.1 : \( \varepsilon \rightarrow \varepsilon \} \} | \begin{align*}
PI &= \{ \varepsilon \} \\
FI &= B \setminus \{ B_0 \} \cup C \cup C' \cup F \cup (D \setminus \{ D_0 \}) \\
PO &= \emptyset \\
FO &= \{ \varepsilon \}
\end{align*} |

| 7, \( s, (2), \emptyset \) | \emptyset | \begin{align*}
PI &= T \\
FI &= V \setminus T \\
PO &= \emptyset \\
FO &= T
\end{align*} |

Table 1.

Observe, that rules 1.1 and 1.5 may be applied in any order. After then, string \( C_i' w B'_{i_k,0} \) 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:

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

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

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 \(\varepsilon'wB'_{i_k,i_1}\) or \(\varepsilon'wB_{i_k,i_1}\) 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 \(\varepsilon'wB'_{i_k,i_1} \xrightarrow{3.1.1} \varepsilon'wB_{i_k,i_1}D_0\) or evolution step \(\varepsilon'wB_{i_k,i_1} \xrightarrow{3.1.1} \varepsilon'wB_{i_k,i_1}D_0\) is performed. Strings \(\varepsilon'wB'_{i_k,i_1}D_0\) or \(\varepsilon'wB_{i_k,i_1}D_0\) can enter only node 4, where (depending on the form of the string) either evolution step \(\varepsilon'wB_{i_k,i_1}D_0 \xrightarrow{4.1} \varepsilon'wE_{i_k,i_1}D_0\) or evolution step \(\varepsilon'wB_{i_k,i_1}D_0 \xrightarrow{4.2} \varepsilon'wE_{i_k,i_1}D_0\) follows. The obtained word, \(\varepsilon'wE_{i_k,i_1}D_0\), can enter only node 6 where evolution step \(\varepsilon'wE_{i_k,i_1}D_0 \xrightarrow{6.1} wE_{i_k,i_1}D_0\) is performed. Then the strings 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:

\[
wE_{i_k,i_1-t}D_t \xrightarrow{2.4} wE'_{i_k,i_1-(t+1)}D_t \xrightarrow{2.5} wE'_{i_k,i_1-(t+1)}D_{t+1} \text{ (in node 2)},
\]

\[
wE'_{i_k,i_1-t}D'_t \xrightarrow{4.8} wE'_{i_k,i_1-(t+1)}D'_t \xrightarrow{4.9} wE'_{i_k,i_1-(t+1)}D_{t+1} \text{ (in node 1)},
\]

The process continues in nodes 1 and 2 until the second subscript of \(E_{i,j}\) or that of \(E_{i,j}^\prime\) is decreased to 1. In this case, either rule 1.10: \(E_{i_k,1}^\prime \to F_{i_k}\) in node 1 or rule 2.6: \(E_{i_k,1} \to F_{i_k}\) in node 2 is applied and the new string, \(wF_{i_k}D_0\) or \(wF_{i_k}'D'_{i_1}\), 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:

\[
wF_{i_k}D_{i_1}(wF_{i_k}'D'_{i_1}) \xrightarrow{5.1(5.2)} wF_{i_k}B_{i_1,0} \xrightarrow{5.3} wA_{i_k}B_{i_1,0}.\]

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 \(\varepsilon'\) will be deleted.
Corollary 5. The class of complete HNEPs with 7 nodes is computationally complete.
4 Program Check

A proof of Theorem 4 was checked for errors using a program that simulates HNEPs, developed by the first author.

5 Conclusions

We presented a universal complete HNEP with 7 nodes and proved that HNEPs with 2 nodes are not computationally complete. In [1] a complete HNEP with 10 nodes was constructed for simulating the universal Circular Post Machine with 34 states and 2 symbols [2, 8, 9]. Using the technique of the proof of Theorem 4, similar constructions simulating different variants of the universal Turing machine (with limited size parameters) with HNEPs with seven nodes can also be obtained.

References


