Journal of Automata, Languages and Combinatorics **u** (v) w, x–y

SMALL UNIVERSAL NETWORKS OF EVOLUTIONARY PROCESSORS

Sergiu Ivanov
LACL, Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
e-mail: sergiu.ivanov@u-pec.fr

Yurii Rogozhin
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău MD-2028, Moldova
e-mail: rogozhin@math.md

and

Sergey Verlan
LACL, Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
e-mail: verlan@u-pec.fr

ABSTRACT

In this article we consider universal networks of evolutionary processors (NEP) having a small number of rules. We show that it is possible to construct a universal NEP with 5 rules when no input/output encoding is used, and a universal NEP with 4 rules in the case in which such an encoding is allowed. We also give a construction of a universal NEP with 7 rules that efficiently (in polynomial time) simulates any Turing machine.

Keywords: Networks of evolutionary processors, Universality

1. Introduction

The concept of universality was first formulated by A. Turing in [26]. He constructed a universal (Turing) machine capable of simulating the computation of any other (Turing) machine. This universal machine takes as input a description of the machine to simulate, the contents of its input tape, and computes the result of its execution on the given input.

More generally, the universality problem for a class of computing devices (or functions) $\mathfrak{C}$ consists in finding a fixed element $\mathcal{M}$ of $\mathfrak{C}$ able to simulate the computation of any element $\mathcal{M}'$ of $\mathfrak{C}$ using an appropriate fixed encoding. More precisely, if $\mathcal{M}'$ computes y on an input x (we will write this as $\mathcal{M}'(x) = y$), then

$\mathcal{M}'(x) = f(\mathcal{M}(g(\mathcal{M}'), h(x)))$, where h and f are the encoding and decoding functions, respectively, and g is the function retrieving the number of $\mathcal{M}'$ in some fixed enumeration of $\mathfrak{C}$. These functions should not be too complicated, otherwise the universal machine will be trivial, e.g. when f is partially recursive the machine can contain only one instruction – stop. It is commonly admitted that general recursive functions [15] can be used for encoding and decoding. We will use the terminology considered by Korec [16] and call the element $\mathcal{M}$ (weakly) universal for $\mathfrak{C}$. We shall call $\mathcal{M}$ strongly universal (for $\mathfrak{C}$) if the encoding and decoding functions are identities. Some authors [17, 16] implicitly consider only the strong notion of universality as the encoding and decoding functions can perform quite complicated transformations, which are not necessarily doable in the original devices. We refer to [16] for a more detailed discussion of different variants of the universality and to [20] for a survey on this topic. We remark that in the case of devices not working with integers directly, some natural coding of integers should be used in order to consider the above notions.

Networks of language processors are finite collections of rewriting systems (language processors) organized in a communicating system [11]. They are closely related to grammar systems, more specifically to parallel communicating grammar systems [9]. 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 redistribute the resulting strings according to a communication protocol assigned to the system. In a derivation step, any node derives from its language all possible words as its new language. In a communication step, any node sends those words to other nodes that satisfy an output condition given as a regular language, and any node takes those words sent by the other connected nodes that satisfy an input condition also given by a regular language. The language determined by the system is defined as the set of words which appear at some distinguished node during the computation.

Networks of evolutionary processors (NEPs), introduced in [7], and also inspired by cell biology, are special 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. Results on networks of evolutionary processors can be found in [7, 8, 6, 4, 13, 3]. Most of these articles show that networks of evolutionary processors are computationally complete or universal using different number of nodes and eventually additional squeezing mechanisms. Yet, the focus always remains on the number of nodes in the network rather than on the number of rules. It turns out that in most of the cases this second parameter depends on the size of the simulated computing device (e.g., the number of the rules in a type-0 grammar in Kuroda normal form [13]) and is rather large.

In the case of hybrid networks of evolutionary processors (HNEPs), each language processor performs only one of the above operations at some position in the words at that node. The filters are defined by some variants of random-context conditions. The concept was introduced in [22, 21]. A series of papers showed the computational completeness of this variant with different number of nodes [1, 10, 18, 2, 19, 5].

In this article we also consider the problem of the universality for NEPs, but we target another descriptional complexity parameter which was not considered so far in this framework: the number of rules. The motivation for such a research is to find the minimal ingredients needed to achieve a complex (universal) computation. A similar approach is common for the area of small universal Turing or register machines where the number of rules is the main descriptional complexity parameter. We refer to [24] for an overview of recent results on this topic. In this paper we show that there exist strongly universal NEPs with 5 rules and weakly universal NEPs with 4 rules. In these cases we assume a unary encoding of integers in NEPs as described in Section 3. We also show a universal NEP with 7 rules that is able to efficiently simulate any Turing machine in polynomial time (since no integer encoding is used the above discussion about strong and weak universality is not relevant in this case).

2. Preliminaries

2.1. Turing machines

In this paper we consider non-stationary deterministic Turing machines, *i.e.*, those in which at each step of the computation the head moves either to the left or to the right. These machines are given as $M = (Q, \Sigma, a_1, q_1, F, \delta)$, where Q is the set of states, Σ is the tape alphabet, $a_1 \in \Sigma$ is the blank symbol, $q_1 \in Q$ is the initial state, $F \subseteq Q$ is the set of final (halting) states, and δ denotes the set of instructions. Each instruction is of the form (q_i, a_k, q_j, a_l, D) which is interpreted as follows: if the head of M being in state q_i is scanning a cell which contains a_k, then the contents of the scanned cell is replaced by a_l, the head moves to the left ($D = L$) or to the right ($D = R$) and the state of the machine changes to q_j.

By a *configuration* of a Turing machine we mean a string $w_1 q w_2$, where $w_1 \in \Sigma^*$, $w_2 \in \Sigma^+$ and $q \in Q$. A configuration represents the contents of non-empty cells of the working tape of the machine and all the blank symbols in between them, from left to right, (*i.e.* all other cells to the left and to the right are blank), its state, and the position of the head on the tape. The machine head is assumed to read the leftmost letter of w_2. Initially all cells on the tape are blank except finitely many cells. We denote by $C \vdash C'$ the passage from configuration C to C' by applying the corresponding rule of the machine.

Note that the situation in which the tape is empty is represented by the string qa_1, where $a_1 \in \Sigma$ is the empty symbol. The cases in which the Turing machine leaves the region of the tape occupied by the input are represented using the empty symbol a_1 in a similar way.

A halting configuration of a Turing machine is a configuration in which no transition rule is defined for the current state $q \in F$ and the symbol a_k the machine reads on the tape. A computation of a Turing machine on the input word w is a sequence of configurations $C_1, C_2, \ldots, C_t$ which starts in configuration $C_1 = q_1 w$, ends in a halting configuration $C_t = w' q_f w''$ and in which $C_i \vdash C_{i+1}$, $1 \leq i < t$. We will denote this by $M(w) = w'w''$.

It is known that non-stationary deterministic Turing machines are as powerful (in

computational sense) as the generic (non-restricted) ones [25].

A Turing machine U is universal if there exist recursive functions h and f such that $M(c) = h(U(f(g(M,c))))$, where M is any Turing machine, c is any input of M, $g(M,c)$ is the number of the ordered pair (M,c) in some enumeration, e.g. Gödel numbering, f maps $g(M,c)$ to a configuration of U, and h maps a terminal configuration of U to a terminal configuration of M.

2.2. Register machines

A deterministic *register machine* is defined as a 5-tuple $M = (Q, R, q_0, q_f, P)$, where Q is a set of states, $R = \{R_1, \ldots, R_k\}$ is the set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and P is a set of instructions (also called rules) of the following form:

1. *(Increment)* $(q_i, R_t+, q_j) \in P$, $q_i, q_j \in Q, q_i \neq q_j, R_t \in R$ (being in state q_i, increment register R_t and go to state q_j).
2. *(Decrement)* $(q_i, R_i-, q_j) \in P$, $q_i, q_j \in Q, q_i \neq q_j, R_t \in R$ (being in state q_i, decrement register R_t and go to state q_j).
3. *(Zero check)* $(q_i, R_t, q_j, q_s) \in P$, $q_i, q_j, q_s \in Q, R_t \in R$ (being in state q_i, go to q_j if register R_t is not zero or to q_s otherwise).
4. *(Zero test and decrement)* $(q_i, R_t-, q_j, q_s) \in P$, $q_i, q_j, q_s \in Q, R_t \in R$ (being in state q_i, decrement register R_t and go to q_j if successful or to q_s otherwise).
5. *(Stop)* $(q_f, STOP)$ (may only be associated with the final state q_f).

We note that for each state p there is only one instruction of one of the types above.

A configuration of a register machine is given by the $(k+1)$-tuple $(q, n_1, \ldots, n_k)$, where $q \in Q$ and $n_i \in \mathbb{N}, 1 \leq i \leq k$, describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of the types above and in changing the current state to another one. We say that the machine stops if it reaches the state q_f. We say that M computes a value $y \in \mathbb{N}$ on the *input* $x_1, \ldots, x_n$, $x_i \in \mathbb{N}$, $1 \leq i \leq n \leq k$, if, starting from the initial configuration $(q_0, x_1, \ldots, x_n, 0, \ldots, 0)$, it reaches the final configuration $(q_f, y, 0, \ldots, 0)$. We will denote this as $M(x_1, \ldots, x_n) = y$.

It is well-known that register machines compute all partial recursive functions and only them [23]. Therefore, every register machine M with n registers can be associated with the function it computes: an m-ary partial recursive function Φ_M^m, where $m \leq n$. Let $\Phi_0, \Phi_1, \Phi_2, \ldots,$ be a fixed enumeration of the set of unary partial recursive functions. Then, a register machine M is said to be *strongly universal* [16] if there exists a recursive function g such that $\Phi_x(y) = \Phi_M^2(g(x), y)$ holds for all $x, y \in \mathbb{N}$. A register machine M is said to be *(weakly) universal* if there exist recursive functions f, g, h such that $\Phi_x(y) = f(\Phi_M^2(g(x), h(y)))$ holds for all $x, y \in \mathbb{N}$. We remark that here the meaning of the term weakly universal is different from the Turing machines case, where it is commonly used to denote a universal machine working on a tape that has an infinite initial configuration.

We also note that the power and efficiency of a register machine M depend on the set of used instructions. In [16] several sets of instructions are investigated. In particular, it is shown that there are strongly universal register machines with 22 instructions of the forms (q_i, R_t+, q_j) and (q_i, R_t-, q_j, q_s). Moreover, these machines can be effectively constructed.

2.3. Networks of Evolutionary Processors

An *evolution* rule is a rule $a \to b$, with $a, b \in V \cup \{\lambda\}$ and where a and b cannot be both empty. We say that an evolution rule is a substitution rule if both a and b are not λ; it is a deletion rule if $a \neq \lambda$ and $b = \lambda$; it is an insertion rule if $a = \lambda$ and $b \neq \lambda$. The set of all evolution, substitution, deletion, and insertion rules over an alphabet V are denoted by Evo_V, Sub_V, Del_V, and Ins_V, respectively.

Given a rule $\sigma : a \to b \in Evo_V$ we define the result of the application of σ on a word $w \in V^*$:

$$\sigma(w) = \begin{cases} \{ubv \mid w = uav, u, v \in V^*\}, & \text{if } w \text{ contains } a, \\ \{w\}, & \text{otherwise.} \end{cases}$$

We can generalize this notation for a language and write $\sigma(L) = \bigcup_{w \in L} \sigma(w)$.

A *network of evolutionary processors* of size n is the tuple $\Gamma = (V, N_1, \ldots, N_n, k, G)$, where V is an alphabet, $1 \leq k \leq n$ designates the output node and $N_i = (M_i, A_i, IF_i, OF_i)$ is the i-th node (processor), $1 \leq i \leq n$, in which:

- $M_i \subseteq Evo_V$ is a finite set of evolutionary rules,
- A_i is a finite set of strings over V (initial strings),
- IF_i and OF_i are regular languages over V specifying conditions for a string to enter and to exit a node, respectively (input and output filters).

Finally, $G = (\{N_1, \cdots, N_n\}, E)$ is an undirected graph specifying the underlying communication network.

Sometimes the NEPs as we define them here are referred to as *mixed* NEPs, in contrast with the classical definition of this computational model in which the processors are only allowed to carry out one type of operation, i.e., a processor is only allowed to execute either insertions, or deletions, or substitutions [4]. As it will be visible later, allowing mixed processors is a convention that simplifies descriptions of networks without essentially modifying their computational power.

The configuration $C = (C_1, \ldots, C_n)$ of the system consists of the sets of strings appearing in each node (each string appears in an arbitrary large number of copies).

The system evolves from a configuration $C = (C_1, \ldots, C_n)$ to a configuration $C' = (C'_1, \ldots, C'_n)$ in two kinds of alternating steps:

- evolution step $(C \Rightarrow C')$: $C'_i = \bigcup_{\substack{\sigma \in M_i \\ w \in C_i}} \sigma(w)$,
- communication step $(C \vdash C')$: $C'_i = C_i \setminus OF_i \cup \bigcup_{(N_i, N_j) \in E} C_j \cap OF_j \cap IF_i$.

A computation consists of a sequence of configurations, where $C^0 = (A_1, \ldots, A_n)$, $C^{2i} \Rightarrow C^{2i+1}$ and $C^{2i+1} \vdash C^{2i+2}$ for $i \geq 0$. Remember that each string is present in an arbitrarily large number of copies, which means that, after the rule applications of an evolution step, we get arbitrarily many copies of each of the resulting strings in each of the nodes of the network. The result of a (possibly infinite) computation is a language collected in a designated node N_k called the output node. Thus, $L(\Gamma) = \cup_{t \geq 0} C_k^t$. For the purpose of our paper we define the computation of Γ on an input word w, denoted by $\Gamma(w)$. We assume that Γ has the property $A_i = \emptyset$, $i \geq 1$. Then $\Gamma(w)$ can be computed by adding w to A_1 ($A_1 = \{w\}$, $A_i = \emptyset$, $i \geq 2$), obtaining the NEP Γ', and having Γ' evolve as usual; clearly, $\Gamma(w) = L(\Gamma')$.

In this paper we only consider NEPs performing deterministic computations, *i.e.* $\Gamma(w)$ is always a singleton language, and we say that the corresponding word is the result of the computation of Γ on w.

3. Simulation of register machines

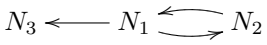
In this section we show that any register machine can be simulated by a NEP. Since such machines deal with integer numbers, we need to modify the definitions of the input and output of a NEP in order to accommodate to this property.

Let $\mathcal{M} = (k, Q, q_1, q_0, P)$ be a register machine with k registers (we remark that the initial state is q_1 and the final state is q_0).

We will use the unary encoding to represent a configuration $(q_i, n_1, \ldots, n_k)$ of $\mathcal{M}$: $1^i 01^{n_1} 0 \ldots 1^{n_k} 0$. In this encoding the number of symbols 1 in the first block corresponds to the state and the size of each next block of 1's corresponds to the value of the respective register. Hence, the initial configuration is represented by the string $101^{n_1} 0 \ldots 1^{n_k} 0$, where $n_1, \ldots, n_k$ is the input of $\mathcal{M}$. We assume that $\mathcal{M}$ produces the result in the first register and empties all other registers before halting.

For a vector $v = (n_1, \ldots, n_k)$, $n_i \in \mathbb{N}$, $k > 0$, we define the result $\Gamma(v)$ of the computation of Γ on v as the length of the string $\Gamma(101^{n_1} 0 \ldots 1^{n_k} 0)$.

We will construct a NEP $\Gamma_{\mathcal{M}} = (\{0, 1, \bar{1}, \bar{0}, \bar{\bar{1}}\}, N_1, N_2, N_3, 3, G)$ with the following communication graph G:



The processors of $\Gamma_{\mathcal{M}}$ are defined as follows:

$$\begin{aligned}
N_1 : IF_1 &= \{0,1\}^*, \\
M_1 &= \left\{\lambda \to \bar{1}, 1 \to \bar{\bar{1}}\right\}, \\
OF_1 &= \left\{\bar{1}^j\bar{\bar{1}}^i 0(1^*0)^{t-1}1^*\bar{1}0(1^*0)^{k-t} \mid (q_i, R_t+, q_j) \in P\right\} \\
&\cup \left\{\bar{1}^j\bar{\bar{1}}^i 0(1^*0)^{t-1}1^*\bar{\bar{1}}0(1^*0)^{k-t} \mid (q_i, R_t-, q_j, q_s) \in P\right\} \\
&\cup \left\{\bar{1}^s\bar{\bar{1}}^i 0(1^*0)^{t-1}0(1^*0)^{k-t} \mid (q_i, R_t-, q_j, q_s) \in P\right\}, \\
A_1 &= \emptyset; \\
\\
N_2 : IF_2 &= \{0,1,\bar{1},\bar{\bar{1}}\}^*, \\
M_2 &= \left\{\bar{1} \to 1, \bar{\bar{1}} \to \lambda\right\}, \\
OF_2 &= \left\{(1^*0)^{k+1}\right\}, \\
A_2 &= \emptyset; \\
\\
N_3 : IF &= \{01^*00^{k-1}\}, \\
M_3 &= \emptyset, \\
OF_3 &= \emptyset, \\
A_3 &= \emptyset.
\end{aligned}$$

The NEP $\Gamma_{\mathcal{M}}$ simulates $\mathcal{M}$ as follows. The main idea is to use processor N_1 to mark the symbols that should be deleted by $\bar{\bar{1}}$ and the symbols that should be added by $\bar{1}$. Since unary encoding is used, adding and deleting symbols permits rewriting both the state and the values of registers using the same set of rules. In more detail, suppose that $\mathcal{M}$ is in configuration $(q_i, n_1, \ldots, n_t, \ldots, n_k)$ and there exists an instruction $(q_i, R_t+, q_j) \in P$. Then $\Gamma_{\mathcal{M}}$ will contain the string $1^i01^{n_1}0\ldots1^{n_t}0\ldots1^{n_k}0$ in processor N_1. The rules from M_1 can now be repeatedly applied, yielding strings over $\{0,1,\bar{1},\bar{\bar{1}}\}^*$. However, only a string of the form $\bar{1}^j\bar{\bar{1}}^i01^{n_1}0\ldots1^{n_t}\bar{1}0\ldots1^{n_k}0$ can pass the filters OF_1 and IF_2 and go to processor N_2. We remind that the marking of this string instructs to add j symbols and delete i symbols in the first block of 1's, thus referring to the transition from q_i to q_j, and to add an additional symbol 1 to the $(t+1)$-th block of 1's, which corresponds to the increment of the register R_t. Now, in order to pass OF_2, all symbols $\bar{\bar{1}}$ should be erased and all instances of $\bar{1}$ rewritten to 1. The only result of this transformation that can pass through OF_2 and IF_1 is the string $1^j01^{n_1}0\ldots1^{n_t+1}0\ldots1^{n_k}0$ which corresponds to the configuration $(q_j, n_1, \ldots, n_t+1, \ldots, n_k)$.

The simulation of a decrement instruction of $\mathcal{M}$ is done in a similar way: the new state is marked and the corresponding register is decremented by marking one of its 1's with a double bar. If there are no instances of 1 in the corresponding block, the state corresponding to the empty register is selected.

Theorem 3.1 *There exist a weakly universal NEP with 4 rules and a strongly universal NEP with 5 rules.*

Proof. The proof follows from the fact that there exist strongly universal register machines, *e.g.* U_{22} from [16]. Hence, $\Gamma_{U_{22}}$ constructed as above will be weakly universal because for any $\mathcal{M}$ it is true that $\Gamma_{\mathcal{M}}(x) = \mathcal{M}(x) + k + 1$. It is possible to achieve a strong universality by adding a new output processor connected to N_3, a rule $0 \to \lambda$ to M_3, and a filter 1^* to OF_3. This rule and the corresponding filter will remove symbols 0 and only the value of the first register will be kept. □

4. Simulation of Turing machines

In this section we will construct a NEP $\Gamma_{\mathcal{T}}$ capable of simulating an arbitrary Turing machine $\mathcal{T} = (Q, \Sigma, a_1, q_1, F, \delta)$, *i.e.* such that for any word $w \in \Sigma^*$ the following holds: $\mathcal{T}(w) = h'(\Gamma_{\mathcal{T}}(f'(w)))$, where f' and h' are the coding and decoding functions correspondingly.

Let $\Sigma = \{a_1, \ldots, a_m\}$ and consider the following (unary) coding: $\phi(a_k) = 1^k 0$, $k \geq 0$, so the empty symbol is coded as 10. We remark that this coding can be extended to words in a standard way. Similarly, consider the coding $\psi(q_i) = 0^i$ for $Q = \{q_1, \ldots, q_n\}$. We represent a configuration $w' q_i a_k w''$ of $\mathcal{T}$ in the following way: $\phi(w')\psi(q_i)\phi(a_k)\phi(w'')$. Accordingly, we represent the initial configuration of $\mathcal{T}$ as follows: $10 \cdot 0 \cdot \phi(w)$, where w is the input of $\mathcal{T}$ and the highlighted instance of 0 is the code of the initial state q_1 of $\mathcal{T}$. If w is empty, it is replaced by an instance of a_1 to assure that there is at least one symbol to the left and to the right of the head.

Hence, the function f' is defined as $f'(x) = \phi(a_1)\psi(q_1)\phi(x)$ and the function h' as $h'(x) = \pi(z^{-1}(x))$, where the functions z and π are defined as follows: $z(x) = \psi(x)$ if $x \in Q$ and $z(x) = \phi(x)$ otherwise; $\pi(x) = \lambda$ if $x \in Q$ and $\pi(x) = x$ if $x \notin Q$.

We construct the NEP $\Gamma_{\mathcal{T}} = (\{0, 1, \bar{1}, \bar{0}, \bar{\bar{1}}\}, N_1, N_2, N_3, N_4, 4, G)$ with the following communication graph G:

The processors of this network are defined as follows:

$$
\begin{aligned}
N_1 : IF_1 &= \left\{(1^+0)^+0^i(1^+0)^+ \mid q_i \notin F\right\},\\
M_1 &= \left\{\lambda \to \bar{1}, 1 \to \bar{\bar{1}}, \lambda \to \bar{0}\right\},\\
OF_1 &= \left\{(1^+0)^+\ 0^i\ 1^k\bar{1}^{l-k}0\ \bar{0}^j\ (1^+0)^+ \mid (q_i, a_k, q_j, a_l, R) \in \delta, l \geq k\right\}\\
&\quad \cup \left\{(1^+0)^+\ 0^i\ 1^k\bar{1}^{l-k}0\ \bar{0}^j\ \bar{1}\bar{0} \mid (q_i, a_k, q_j, a_l, R) \in \delta, l \geq k\right\}\\
&\quad \cup \left\{(1^+0)^+\ 0^i\ 1^l\bar{\bar{1}}^{k-l}0\ \bar{0}^j\ (1^+0)^+ \mid (q_i, a_k, q_j, a_l, R) \in \delta, l < k\right\}\\
&\quad \cup \left\{(1^+0)^+\ 0^i\ 1^l\bar{\bar{1}}^{k-l}0\ \bar{0}^j\ \bar{1}\bar{0} \mid (q_i, a_k, q_j, a_l, R) \in \delta, l < k\right\}\\
&\quad \cup \left\{(1^+0)^+\ \bar{0}^j\ (1^+0)\ 0^i\ 1^k\bar{1}^{l-k}0\ (1^+0)^* \mid (q_i, a_k, q_j, a_l, L) \in \delta, l \geq k\right\}\\
&\quad \cup \left\{\bar{1}\bar{0}\ \bar{0}^j\ (1^+0)\ 0^i\ 1^k\bar{1}^{l-k}0\ (1^+0)^* \mid (q_i, a_k, q_j, a_l, L) \in \delta, l \geq k\right\}\\
&\quad \cup \left\{(1^+0)^+\ \bar{0}^j(1^+0)0^i\ 1^l\bar{\bar{1}}^{k-l}0\ (1^+0)^* \mid (q_i, a_k, q_j, a_l, L) \in \delta, l < k\right\}\\
&\quad \cup \left\{\bar{1}\bar{0}\ \bar{0}^j\ (1^+0)\ 0^i1^l\ \bar{\bar{1}}^{k-l}\ 0(1^+0)^* \mid (q_i, a_k, q_j, a_l, L) \in \delta, l < k\right\},\\
A_1 &= \emptyset;\\
N_2 : IF_2 &= \left\{0, 1, \bar{0}, \bar{1}, \bar{\bar{1}}\right\}^*,\\
M_2 &= \left\{0 \to \bar{\bar{1}}\right\},\\
OF_2 &= \left\{(1^+0)^+\ \bar{\bar{1}}^+\ 1^+(\bar{1}^*|\bar{\bar{1}}^*)0\ \bar{0}^+\ \left((1^+0)^+|\bar{1}\bar{0}\right)\right\}\\
&\quad \cup \left\{\left((1^+0)^+|\bar{1}\bar{0}\right)\ \bar{0}^+\ (1^+0)\ \bar{\bar{1}}^+\ 1^+(\bar{1}^*|\bar{\bar{1}}^*)0\ (1^+0)^+\right\},\\
A_2 &= \emptyset;\\
N_3 : IF_3 &= \left\{0, 1, \bar{0}, \bar{1}, \bar{\bar{1}}\right\}^*,\\
M_3 &= \left\{\bar{1} \to 1, \bar{\bar{1}} \to \lambda, \bar{0} \to 0\right\},\\
OF_3 &= \{0, 1\}^*,\\
A_3 &= \emptyset;\\
N_4 : IF_4 &= \left\{(1^+0)^+0^i(1^+0)^+ \mid q_i \in F\right\},\\
M_4 &= \emptyset,\\
OF_4 &= \emptyset,\\
A_4 &= \emptyset.
\end{aligned}
$$

We recall that the result of the computation of $\Gamma_{\mathcal{T}}$ is the (only) string that reaches processor N_4.

We will now describe how the network $\Gamma_{\mathcal{T}}$ simulates the Turing machine $\mathcal{T}$. The main idea is essentially similar to what we have already shown in Section 3: processor

N_1 inserts $\bar{1}$'s to mark additions of 1's to the string and rewrites certain 1's into $\bar{\bar{1}}$'s to schedule them for subsequent deletion. To rewrite the substring $\phi(a_k)$ into $\phi(a_l)$, we either need to add some 1's to the block of 1's coding a_k if $l \geq k$, or to erase some 1's otherwise. This exactly is the mission of N_1, which also inserts a number of $\bar{0}$'s to select the next state of $\mathcal{T}$.

The output filter OF_1 exercises very precise control over which strings are eventually allowed out of N_1. The first line in the definition of OF_1 allows the string out if it corresponds to a transition of $\mathcal{T}$ during which the head moves to the right, a_k is rewritten into a_l, $l \geq k$, and the sequences of 0's and $\bar{0}$'s code the corresponding old state q_i and the new state q_j. The second line in the definition of OF_1 allows N_1 to deal with the situations in which a_k is rewritten into a_l, $l \geq k$, and the head is at the right end of the string: a new empty cell 10 is scheduled for insertion by placing $\bar{1}\bar{0}$ at the right end. The next two lines in the definition of OF_1 have exactly the same mission as the first two lines, but for the case in which $l < k$. Note how the first two lines expect additions of 1's by requiring the presence of $\bar{1}$'s, while the second two lines expect removals of 1 by requiring the presence of double-barred instances of 1. The next four lines handle the same special cases for leftward head moves.

The mission of N_2 is to mark the previous state for deletion. This is done by replacing the corresponding sequence of 0's by a sequence of $\bar{\bar{1}}$'s. We remark that due to the form of the strings that can reach N_2 (the ones which pass the filter OF_1), only those strings will be allowed out of N_2 in which all the 0's representing the old state have been rewritten, and only those 0's.

The role of N_3 is to apply the operations scheduled by N_1 and N_2 by inserting barred symbols and erasing double-barred 1's. N_3 forwards its output to both N_1 and N_4, but only one of these two processors will accept the string, since N_4 only lets in strings representing a configuration of $\mathcal{T}$ in one of its final states, while N_1 will only work on strings which contain the code of a non-final state.

Theorem 4.1 *There exists a universal NEP with 7 rules that simulates any Turing machine in polynomial time.*

Proof. The proof follows from the existence of universal Turing machines that simulate the target machine in polynomial time, see [24] for more details. □

Note that the simulation of each step of a Turing machine happens in a fixed amount of amount of time depending on the codes of the implied symbols.

Finally, we remark that, while the universal Turing machines we need to simulate do not rely on commands which only rewrite symbols without moving the head, it is rather easy to apply the ideas we have just shown to simulate such instructions.

5. Conclusion

In this paper we gave 3 constructions for universal NEPs having a small number of rules. The result from Theorem 3.1 is remarkable in the sense that it uses only 4 elementary rules in order to achieve universality. The constructions we give clearly show that a large part of the computational power of NEPs is provided by filters.

It could therefore be interesting and practical to investigate a similar universality problem for systems in which the filtering components are restricted, *e.g.* NEPs from [14, 12], which use subclasses of regular languages as filters, or HNEPs that use random context filters.

We also remark that, in NEPs, one traditionally considers a single type of operation (insertion, deletion or substitution) per processor. It is not difficult to see that the constructions we give in this paper can be easily adapted to this condition without modifying the number of rules just by increasing the number of nodes in the network.

References

[1] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, Yu. Rogozhin. About Universal Hybrid Networks of Evolutionary Processors of Small Size. In C. Martn-Vide, F. Otto, H. Fernau (Eds.), Proceedings of the 2nd International Conference on Language and Automata Theory and Applications, LATA 2008, *Lecture Notes in Computer Science*, 5196, Springer, 2008, p.28–39.

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

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

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

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

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

[7] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere. Solving NP-complete Problems with Networks of Evolutionary Processors. In J. Mira, A. Prieto (Eds.), IWANN 2001, *Lecture Notes in Computer Science*, 2084, 2001, p.621–628.

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

[9] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, Grammar Systems. Gordon and Breach, 1993.

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

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

[12] J. Dassow, F. Manea, B. Truthe. Networks of Evolutionary Processors: the Power of Subregular Filters. *Acta Informatica*, **50**(1), 2013, p.41–75.

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

[14] J. Dassow, B. Truthe. On Networks of Evolutionary Processors with State Limited Filters. In Second Workshop on Non-Classical Models for Automata and Applications, Proceedings. Austrian Computer Society 2010, p.57–70.

[15] S. Kleene. On Notations for Ordinal Numbers. *Journal of Symbolic Logic*, **3**, 1938, p.150–155.

[16] I. Korec. Small Universal Register Machines. *Theoretical Computer Science*, **168**(2), 1996, p.267–301.

[17] A. I. Malcev. *Algorithms and Recursive Functions*. Groningen, Wolters-Noordhoff Pub. Co., 1970.

[18] F. Manea, C. Martín-Vide, V. Mitrana. All NP-problems can be Solved in Polynomial Time by Accepting Hybrid Networks of Evolutionary Processors of Constant Size. *Information Processing Letters*, **103**, 2007, p.112–118.

[19] F. Manea, C. Martín-Vide, V. Mitrana. On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors. *Mathematical Structures in Computer Science*, **17**(4), 2007, p.753–771.

[20] M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, **231**(2), p.217-251, 2000.

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

[22] C. Martín-Vide, V. Mitrana, M. Pérez-Jiménez, F. Sancho-Caparrini. Hybrid Networks of Evolutionary Processors. In: E. Cantú-Paz et al. (Eds.), Proc. of GECCO 2003, *Lecture Notes in Computer Science*, 2723, 2003, p.401–412.

[23] Marvin Minsky. *Computations: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffts, NJ, 1967.

[24] T. Neary, D. Woods. The Complexity of Small Universal Turing Machines: A Survey. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: 38th Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 385–405. Springer, 2012.

[25] G. Rozenberg, A. Salomaa. *Handbook of Formal Languages*, Springer, 1997.

[26] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. of the London Mathematical Society*, **42**(2), 1936, 230–265.