Accepting Networks of Evolutionary Processors with Subregular Filters

Florin Manea$^{(A)}$ Bianca Truthe

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
{manea,truthe}@iws.cs.uni-magdeburg.de

Abstract

In this paper, we propose a new variant of Accepting Networks of Evolutionary Processors, in which the operations can be applied only arbitrarily to the words, while the filters are languages from several special classes of regular sets. More precisely, we show that the use of filters from the class of ordered, non-counting, power-separating, suffix-closed regular, union-free, definite and combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can accept all the recursively enumerable languages. On the other hand, by using filters that are only finite languages, monoids, nilpotent languages, commutative regular languages, or circular regular languages, one cannot generate all recursively enumerable languages. These results seem interesting as they provide both upper and lower bounds on the classes of languages that one can use as filters in an accepting network of evolutionary processors in order to obtain a complete computational model.

1. Introduction

The computational model considered in this paper, accepting networks of evolutionary processors (ANEPs, for short), was introduced in [8]. It is a bio-inspired model based on an architecture considered in [4]. An ANEP can be seen as a graph having in each node a so-called evolutionary processor. By this we mean a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell containing genetic information encoded in DNA sequences, which may evolve by local evolutionary events, namely point mutations; moreover, each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node are organized in the form of multisets of words, each word appearing in an arbitrarily large number of copies, and all the copies are processed as follows: if at least one rule can be applied to a word w, we obtain all the words that are derived from the word w by applying exactly one of the possible rules at exactly one feasible

$^{(A)}$Also at: Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, RO-010014 Bucharest, Romania (flmanea@fmi.unibuc.ro). The work of Florin Manea is supported by the *Alexander von Humboldt Foundation*.

position in the word w. The computation of an ANEP is conducted as follows. Initially, only one special node, the input node, contain a certain word, the input word. Further, the words are processed in alternative evolutionary and communication steps. In an evolutionary step, the words found in each node are rewritten according to the rules of that node. In a communication step, the words of a node are communicated to the other nodes, as permitted by some filtering condition associated with both the sending and the receveing node. The language accepted by an ANEP consists of all the input words for which we reach, during the computation, a situation in which another special node, the output node, contains at least one string. Results on ANEPs, seen as formal languages accepting devices, were surveyed recently in [7]. However, it is worth noting that in the initial paper [8], as well as subsequent papers [5, 6], it was shown that such networks are computationally complete, i. e., they are able to accept all recursively enumerable languages, given that the filters that restrict the communication are from a special class of regular languages (inspired by the usage of random-context conditions defined by the a set of permitting contexts and one of forbidding contexts).

In this paper we are interested in analyzing the computational power of ANEPs in which the filters are languages from other subclasses of regular languages (depicted in Figure 1, see also [10]). The results we obtain show that, when the filters are ordered, non-counting, power-separating, suffix-closed regular, union-free, definite and combinational languages, we obtain classes of ANEPs that can accept all the recursively enumerable languages. On the other hand, when we restrict to the usage of filters that are only finite languages, nilpotent languages, monoids, commutative or circular regular languages, we cannot accept all recursively enumerable languages. We also show that some of the classes of languages accepted by ANEPs with different types of filters are incomparable. Finally, a non-trivial hierarchy of classes of languages accepted by ANEPs with such filters is obtained (shown in Figure 2). A similar research was carried out for generating networks of evolutionary processors (see [1, 2]), but the obtained results were quite different (and the techniques that were used to show them could not be directly used in the case of ANEPs).

2. Basic Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [9]). We here only recall some notations used in the paper.

The set of the natural numbers is denoted by $\mathbb{N}$. By V^* we denote the set of all words (strings) over an alphabet V (including the empty word λ). The length of a word w is denoted by $|w|$. By V^+ and V^k for some natural number k we denote the set of all non-empty words and the set of all words with length k, respectively. Let V_k be the set of all words over V with a length of at most k, i. e., $V_k = \bigcup_{i=0}^{k} V^i$. We denote by $⧢$ the shuffle operation on words; formally, the shuffle of words $u,v \in V^*$ is a set of words denoted by $u ⧢ v$ and defined recursively as

$$x ⧢ \lambda = \lambda ⧢ x = \{x\},\ x \in V^*, \text{ and}$$
$$ax ⧢ by = \{a\}(x ⧢ by) \cup \{b\}(ax ⧢ y),\ a,b \in V,\ x,y \in V^*.$$

By *FIN*, *REG*, and *RE* we denote the classes of the finite, regular, and recursively enumerable languages, respectively.

A phrase structure grammar is specified as a quadruple $G = (N,T,P,S)$ where N is a set of non-terminals, T is a set of terminals, P is a finite set of rules which are written as $\alpha \to \beta$ with $\alpha \in (N \cup T)^* \setminus T^*$ and $\beta \in (N \cup T)^*$, and $S \in N$ is the axiom. It is known that any recursively enumerable language can be generated by a phrase structure grammar in *Kuroda normal form*, i. e., by a grammar where all rules have one of the following forms:

$$AB \to CD,\ A \to CD,\ A \to x \text{ where } A,B,C,D \in N,\ x \in N \cup T \cup \{\lambda\}.$$

For a language L over an alphabet V, we set

$$\begin{aligned} \mathit{Comm}(L) &= \{a_{i_1}\dots a_{i_n} \mid a_1\dots a_n \in L, n \geq 1, \{i_1,\dots,i_n\} = \{1,\dots,n\}\},\\ \mathit{Circ}(L) &= \{vu \mid uv \in L,\ u,v \in V^*\},\\ \mathit{Suf}(L) &= \{v \mid uv \in L,\ u,v \in V^*\}. \end{aligned}$$

We consider the following restrictions for regular languages. Let L be a language and $V = \mathit{alph}(L)$ the minimal alphabet of L. We say that L is

- *combinational* iff it can be represented in the form $L = V^*A$ for some subset $A \subseteq V$,
- *definite* iff it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *nilpotent* iff L is finite or $V^* \setminus L$ is finite,
- *commutative* iff $L = \mathit{Comm}(L)$,
- *circular* iff $L = Circ(L)$,
- *suffix-closed* (or *fully initial* or *multiple-entry* language) iff $xy \in L$ for some $x,y \in V^*$ implies $y \in L$ (or equivalently, $\mathit{Suf}(L) = L$),
- *non-counting* (or star-free) iff there is an integer $k \geq 1$ such that, for any $x,y,z \in V^*$, $xy^kz \in L$ if and only if $xy^{k+1}z \in L$,
- *power-separating* iff for any $x \in V^*$ there is a natural number $m \geq 1$ such that either $J_x^m \cap L = \emptyset$ or $J_x^m \subseteq L$ where $J_x^m = \{x^n \mid n \geq m\}$,
- *ordered* iff L is accepted by some finite automaton $\mathcal{A} = (Z,V,\delta,z_0,F)$ where $(Z,\preceq)$ is a totally ordered set and, for any $a \in V$, $z \preceq z'$ implies $\delta(z,a) \preceq \delta(z',a)$,
- *union-free* iff L can be described by a regular expression which is only built by product and star.

It is obvious that combinational, definite, nilpotent, ordered and union-free languages are regular, whereas non-regular languages of the other types mentioned above exist.

By *COMB*, *DEF*, *NIL*, *COMM*, *CIRC*, *SUF*, *NC*, *PS*, *ORD*, and *UF* we denote the classes of all combinational, definite, nilpotent, regular commutative, regular circular, regular suffix-closed, regular non-counting, regular power-separating, ordered, and union-free languages, respectively. Moreover, we add the class *MON* of all languages of the form V^*, where V is an alphabet (languages of *MON* are target sets of monoids; we call them monoidal languages). We set $\mathcal{G} = \{\mathit{FIN}, \mathit{MON}, \mathit{COMB}, \mathit{DEF}, \mathit{NIL}, \mathit{COMM}, \mathit{CIRC}, \mathit{SUF}, \mathit{NC}, \mathit{PS}, \mathit{ORD}, \mathit{UF}\}$. The relations between these classes of languages are investigated, e. g., in [3] and [10], and their set-theoretic relations are given in Figure 1.

$$\begin{tikzcd}
 & & \mathit{REG} & & \\
 & \mathit{PS} \arrow[ur] & & & \\
 & \mathit{NC} \arrow[u] & & \mathit{CIRC} \arrow[uul] & \\
\mathit{ORD} \arrow[ur] & \mathit{DEF} \arrow[u] & \mathit{UF} \arrow[uuu] & \mathit{COMM} \arrow[u] & \mathit{SUF} \arrow[uuull, bend right] \\
\mathit{NIL} \arrow[u] \arrow[ur] & \mathit{COMB} \arrow[u] \arrow[ul] & & & \\
\mathit{FIN} \arrow[u] & \mathit{MON} \arrow[u] \arrow[ul] \arrow[uur, bend right] \arrow[uurr, bend right] \arrow[uurrr, bend right] & & &
\end{tikzcd}$$

Figure 1: Hierarchy of subregular languages (an arrow from X to Y denotes $X \subset Y$, and if two classes are not connected by a directed path then they are incomparable)

We call a rule $\alpha \to \beta$ a
– substitution if $|\alpha| = |\beta| = 1$,
– deletion if $|\alpha| = 1$ and $\beta = \lambda$.
– insertion if $\alpha = \lambda$ and $|\beta| = 1$.

The rules are applied like context-free rewriting rules. We say that a word v derives a word w, written as $v \Longrightarrow w$, if there are words x, y and a rule $\alpha \to \beta$ such that $v = x\alpha y$ and $w = x\beta y$. If the rule p applied is important, we write $v \Longrightarrow_p w$.

Further, we define the accepting networks of evolutionary processors (ANEPs for short).

Definition 2.1 *Let X be a subclass of regular languages.*

(i) *An accepting network of evolutionary processors (of size n) with filters from X is a tuple*

$$\mathcal{N} = (V, U, N_1, N_2, \ldots, N_n, E, N_{n_i}, N_{n_o})$$

where

- *V is a finite alphabet, called the input alphabet of the network,*
- *U is a finite alphabet, called the working alphabet of the network,*
- *for $1 \leq i \leq n$, the evolutionary processor $N_i = (M_i, I_i, O_i)$ is defined by*
  - *M_i is a set of rules of a certain type (substitution, deletion or insertion): $M_i \subseteq \{a \to b \mid a, b \in U\}$ or $M_i \subseteq \{a \to \lambda \mid a \in U\}$ or $M_i \subseteq \{\lambda \to b \mid b \in U\}$,*
  - *I_i and O_i are languages from X, included in U^*; I_i is called the input filter of the processor, and O_i is called the output filter of the processor,*
- *E is a subset of $\{N_1, N_2, \ldots, N_n\} \times \{N_1, N_2, \ldots, N_n\}$, denoting the edges that connect the processors of the network, and*

- *n_i and n_o are two natural numbers such that $1 \leq n_i, n_o \leq n$; N_{n_i} is the input node of the network, and N_{n_o} is the output node of the network.*

(ii) *A configuration C of $\mathcal{N}$ is an n-tuple $C = (C(1), C(2), \ldots, C(n))$ where $C(i)$ is a subset of U^* for $1 \leq i \leq n$.*

(iii) *Let $C = (C(1), C(2), \ldots, C(n))$ and $C' = (C'(1), C'(2), \ldots, C'(n))$ be two configurations of $\mathcal{N}$. We say that C derives C' in one*

- *evolutionary step (written as $C \Longrightarrow C'$) if, for $1 \leq i \leq n$, $C'(i)$ consists of all words $w \in C(i)$ to which no rule of M_i is applicable and of all words w for which there are a word $v \in C(i)$ and a rule $p \in M_i$ such that $v \Longrightarrow_p w$ holds,*
- *communication step (written as $C \vdash C'$) if, for $1 \leq i \leq n$,*

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(N_k, N_i) \in E} (C(k) \cap O_k \cap I_i).$$

The computation of an evolutionary network $\mathcal{N}$ on an input word $w \in V^*$ is a sequence of configurations $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$, such that

- *$C_0^w(n_i) = \{w\}$ and $C_0^w(j) = \emptyset$ for $j \in \{1, \ldots, n\} \setminus \{n_i\}$,*
- *for any $t \geq 0$, C_{2t}^w derives C_{2t+1}^w in one evolutionary step,*
- *for any $t \geq 0$, C_{2t+1}^w derives C_{2t+2}^w in one communication step.*

The computation of an evolutionary network $\mathcal{N}$ on an input word $w \in V^*$ is said to be accepting *if there exists a configuration C_t^w in which $C_t^w(n_o)$ is non-empty.*

(iv) *The language $L(\mathcal{N})$ accepted by $\mathcal{N}$ is defined as*

$$L(\mathcal{N}) = \{w \mid w \in V^*, \text{ the computation of } \mathcal{N} \text{ on } w \text{ is accepting}\}.$$

Intuitively, an ANEP is a graph consisting of n nodes $1, 2, \ldots, n$ associated with the evolutionary processors $N_1, N_2, \ldots, N_n$. From these processors two are distinguished: N_{n_i}, the input node, and N_{n_o}, the output node. The nodes are connected by the set of edges given by E: there is a directed edge from N_k to N_i if and only if $(N_k, N_i) \in E$; such an edge should be seen as a directed communication channel between the processors N_k and N_i. Any processor N_i consists of a set of evolutionary rules M_i, an input filter I_i and an output filter O_i (where I_i and O_i belong to the class X). We say that N_i is a substitution processor or a deletion processor or an insertion processor if the rules in the set M_i are substitutions or insertions or deletions, respectively. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively, via the communication channels. Assume that $w \in V^*$ is the input word of the network. With any node i and any time moment $t \geq 0$ we associate a set $C_t^w(i)$ of words (the words contained in the node at time t, in the computation on the word w). Initially, all the processors N_i do not contain any words, except for N_{n_i} which contains only the word w. In an evolutionary step, we derive from $C_t^w(i)$ all words applying rules from the set M_i. In a communication step, any processor N_i sends out all words $C_t^w(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t^w(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i, i. e., the processor N_i gets in addition all words of $C_t^w(k) \cap O_k \cap I_i$. We start with an evolutionary step and then communication steps and evolutionary steps are alternately performed. A word is accepted by $\mathcal{N}$ is and only if there exists $t > 0$ such that the output node contains at least one string after t steps where performed, i. e., $C_t^w(n_o) \neq \emptyset$. The accepted language consists of all accepted words.

For a class $X \subseteq$ *REG*, we denote the class of all languages generated by networks of evolutionary processors where all filters are of type X by $\mathcal{A}(X)$.

The following facts are obvious.

Lemma 2.2 *Let X and Y be subclasses of REG such that $X \subseteq Y$ holds. Then also the inclusion $\mathcal{A}(X) \subseteq \mathcal{A}(Y)$ holds.* □

Lemma 2.3 *Let X be a subclass of REG. Then the inclusion $X \subseteq \mathcal{A}(X)$ holds.* □

3. Computationally Complete Cases

In this section, we show the computational completeness of some classes $\mathcal{A}(X)$ where X is a subclass of regular languages from $\mathcal{G}$.

Theorem 3.1 $\mathcal{A}(\mathit{COMB}) = \mathit{RE}$.

Proof. The proofs showing that the class of languages accepted by ANEPs with left/right operations and random context filters (see [5, 7]) can be easily adapted to show that $\mathcal{A}(\mathit{COMB}) \subseteq \mathit{RE}$.

In what follows, we will show that $\mathcal{A}(\mathit{COMB}) \supseteq \mathit{RE}$. For this, let L be a recursively enumerable language. Let $G = (N, T, P, S)$ be a grammar in Kuroda normal form that generates L. Further, let $V = N \cup T$ and let $x_1, x_2, \ldots, x_k$ be the elements of V.

We will construct an ANEP that simulates, bottom-up, a derivation in the grammar G. That is, we try to apply the rules of the grammar reversed, until we get a string that contains just one occurrence of the axiom of the grammar. For the rules of the form $A \to x$ with $x \in N \cup T \cup \{\lambda\}$, this can be easily simulated by insertion or substitution rules. For the rules of P that have the form $\alpha \to \beta$ with $|\beta| = 2$, the discussion is more complicated.

More precisely, let $p = \alpha \to \beta$ be a rule of P with $|\beta| = 2$ and $w\beta a_t a_{t-1} \cdots a_1$ be a sentential form of the grammar G with $w \in V^*$ and $a_i \in V$ for all natural numbers i with $1 \leq i \leq t$. We want to store the symbols $a_1, a_2, \ldots, a_t$ together with their positions in the suffix somewhere else in the word such that the subword β appears in the end of the word. There it can be replaced by the left hand side α of the rule (which can be done using nodes with combinational filters). Finally, the symbols $a_1, a_2, \ldots, a_t$ are restored at their correct positions.

Since a word can be arbitrarily large, the position of a letter a_i can be also an arbitrarily large number and hence cannot be represented by a single symbol from the finite working alphabet of the network. In order to overcome this problem, we consider the symbols of V as digits in the base $k+1$: x_i corresponds to the digit i in this base, and there exists one more digit, that does not correspond to any symbol of V, denoted by 0. Now we define a bijective correspondence associating to a word $w = b_m b_{m-1} \cdots b_1$ the number

$$\overline{b_m \ldots b_1 0} = b_m(k+1)^m + b_{m-1}(k+1)^{m-1} + \cdots + b_1(k+1)^1 + 0(k+1)^0$$

in base $k+1$. Now, instead of storing the symbols $a_1, a_2, \ldots, a_t$ together with their positions in the suffix somewhere in the word, we can store in the word d symbols of 1, given that 1^d is the unary representation of the base $k+1$ number $\overline{b_m \ldots b_1 0}$. After we replace β with α, we just have to rewrite at the end of the word the base $k+1$ number that equals to the number of 1 symbols found in the current word (omitting the final 0), and delete these 1 symbols; in other words, we restore $a_t a_{t-1} \cdots a_1$ at the end of the word.

We denote the network that we construct by $\mathcal{N}$. This network has the following structure, for $p \in P$:

$$\begin{tikzcd}
\mathcal{N}_p \arrow[r, leftrightarrow] & \mathcal{N}_{in} \arrow[r] \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] & \mathcal{N}_{out} \\
\mathcal{N}_{st} & \mathcal{N}_{rst} &
\end{tikzcd}$$

Basically, $\mathcal{N}$ is composed of $4+|P|$ subnetworks:

- $\mathcal{N}_{in}$ contains only the input node. This node also controls the computation by choosing which is the next step to be performed: to simulate the reverse application of a rule of the grammar, to locate the righthand side of a rule and to store a suffix of the word as explained above, to restore the suffix, or to verify if the string can be accepted.

- $\mathcal{N}_{out}$ verifies if the input string was reduced to the axiom of the grammar, and if this is the case, it accepts.

- $\mathcal{N}_p$ implements the application of the rule p reversed, for $p \in P$.

- $\mathcal{N}_{st}$ stores a suffix of the word, seen as a base $k+1$ number, as a unary number.

- $\mathcal{N}_{rst}$ restores a suffix of the word from the unary number saved in the word.

Let $V' = \{x' \mid x \in V\}$, $\overline{V} = \{\overline{x} \mid x \in V\}$, and assume that 1, 1$'$, 1$''$, #, #$'$ and $\perp$ are symbols that do not belong to V. Let

$$U = V \cup V' \cup \overline{V} \cup \{1, 1', 1'', \#, \#'\}.$$

The working alphabet of the network is

$$U' = U \cup \{\perp\},$$

while the input alphabet is T.

In what follows, we define in details the nodes and edges of each of the aforementioned subnetworks.

We start with the input subnetwork $\mathcal{N}_{in}$. This contains only the node $N_{in} = (\emptyset, U^*, U^*)$.

The output subnetwork $\mathcal{N}_{out}$ contains two nodes:

$$N_{out}^1 = (\{S \to S'\}, \{S\}^*, U^*),$$
$$N_{out}^2 = (\emptyset, \{S'\}^*, \emptyset).$$

The node N^2_{out} is also the output node of the network $\mathcal{N}$. The edges that are connected to nodes of this subnetwork are: (N_{in}, N^1_{out}) and (N^1_{out}, N^2_{out}). Clearly, a word that is sent to this network reaches the output node if and only if it is equal to S.

Let p be a rule of the form

$$A \to a \text{ with } A \in N \text{ and } a \in N \cup T.$$

Then the network $\mathcal{N}_p$ has a single node

$$N^1_p = (\{\, a \to A \,\}, U^*, U^*).$$

We also have the edges (N_{in}, N^1_p) and (N^1_p, N_{in}).

Let p be a rule of the form

$$A \to \lambda \text{ with } A \in N.$$

Then the network $\mathcal{N}_p$ has a single node

$$N^1_p = (\{\, \lambda \to A \,\}, U^*, U^*).$$

Again, we have the edges (N_{in}, N^1_p) and (N^1_p, N_{in}).

The case when the rule p has the form

$$AB \to CD,$$

where $A, B, C, D \in N$, is a bit more complicated. Keep in mind that when we try to apply this rule reversed we assume that CD are the last symbols of the communicated word. The network $\mathcal{N}_p$ has 7 nodes:

$$\begin{aligned}
N^1_p &= (\{\, D \to D' \,\} \cup \{\, x \to \bot \mid x \in U \,\}, U^*D, U^*),\\
N^2_p &= (\{\, D' \to \lambda \,\}, U^*D', U^*),\\
N^3_p &= (\{\, C \to C' \,\} \cup \{\, x \to \bot \mid x \in U \,\}, U^*C, U^*),\\
N^4_p &= (\{\, C' \to \lambda \,\}, U^*C', U^*),\\
N^5_p &= (\{\, \lambda \to A' \,\}, U^*, U^*),\\
N^6_p &= (\{\, \lambda \to B' \,\}, U^*A', U^*),\\
N^7_p &= (\{\, B' \to B, A' \to A \,\}, U^*B', (U \setminus V')^*).
\end{aligned}$$

We have also the edges (N_{in}, N^1_p), (N^7_p, N_{in}), and (N^i_p, N^{i+1}_p) for $1 \le i \le 6$.

Finally, the case when the rule p has the form

$$A \to CD,$$

where $A, C, D \in N$, is similar to the above. Again, when we try to apply this rule reversed we assume that CD are the last symbols of the communicated word. The network $\mathcal{N}_p$ has 6 nodes:

$$\begin{aligned}
N_p^1 &= (\{\,D \to D'\,\} \cup \{\,x \to \perp \mid x \in U\,\}, U^*D, U^*),\\
N_p^2 &= (\{\,D' \to \lambda\,\}, U^*D', U^*),\\
N_p^3 &= (\{\,C \to C'\,\} \cup \{\,x \to \perp \mid x \in U\,\}, U^*C, U^*),\\
N_p^4 &= (\{\,C' \to \lambda\,\}, U^*C', U^*),\\
N_p^5 &= (\{\,\lambda \to A'\,\}, U^*, U^*),\\
N_p^6 &= (\{\,A' \to A\,\}, U^*A', (U \setminus V')^*).
\end{aligned}$$

The edges are (N_{in}, N_p^1), (N_p^6, N_{in}), and (N_p^i, N_p^{i+1}) for $1 \le i \le 5$.

It is easy to see that, for all the rules p, the processor $\mathcal{N}_p$ rewrites the current word by reversely applying the rule p of the grammar.

We move now to the two networks $\mathcal{N}_{st}$ and $\mathcal{N}_{rst}$. In both cases, we will give an overview on the algorithms implemented by these networks, and then we will give the full constructions.

The network $\mathcal{N}_{st}$ always receives from N_{in} a word of a set $(\#^q \sqcup\!\sqcup 1^t \sqcup\!\sqcup w)\{x\}$ with $w \in V^*$ and $x \in V$. Its computation on such a word follows the deterministic Algorithm *Store*.

Algorithm 1 *Store*: describes the computation of $\mathcal{N}_{st}$ on a word from the set $(\#^q \sqcup\!\sqcup 1^t \sqcup\!\sqcup w)\{x_i\}$ with $w \in V^*$ and $x_i \in V$

```
 1: Replace the last symbol $x_i$ of the input word with $x_i'$;
 2: Insert in the word the symbol $\#'$;
 3: Insert in the word the symbol $1'$;
 4: **repeat**
 5:    Substitute one symbol $1'$ with $1''$;
 6:    Insert $k-1$ symbols $1''$;
 7: **until** the word does not contain $1'$ symbols anymore;
 8: **if** the word contains $\#$ symbols **then**
 9:    Substitute one symbol $\#$ with $\#'$;
10:    Substitute all the symbols $1''$ with $1'$;
11:    Go to step 4;
12: **else**
13:    Substitute all the symbols $\#'$ with $\#$;
14:    **if** the word contains $x_1'$ **then**
15:       Delete $x_1'$;
16:       Substitute all the symbols $1'$ with $1$;
17:    **else**
18:       Substitute one symbol $\#$ with $\#'$;
19:       Substitute $x_i'$ with $x_{i-1}'$;
20:       Go to step 3;
21: The word is now from the set $\#^{q+1} \sqcup\!\sqcup 1^{t+ik^{q+1}} \sqcup\!\sqcup w$.
```

Similar to the case of $\mathcal{N}_{st}$, the network $\mathcal{N}_{rst}$ always receives from N_{in} a word of a set $\#^q ⧢ 1^t ⧢ w$ with $w \in V^*$, $q \in \mathbb{N}$, and $t \in \mathbb{N}$. Its computation on such a word follows the nondeterministic Algorithm *Restore*.

Algorithm 2 *Restore*: describes the computation of $\mathcal{N}_{rst}$ on a word of the set $\#^q ⧢ 1^t ⧢ w$ with $w \in V^*$

```
1: Insert in the word the symbol $x'_1$;
2: Substitute one symbol # with #';
3: if the word does not contain $1'$ symbols anymore then
4:     Substitute $k$ symbols 1 with $1''$ (if this step cannot be completed, the word is blocked);
5: else
6:     Substitute one symbol $1'$ with $1''$;
7:     Substitute $k-1$ symbols 1 with $1''$ (if this step cannot be completed, the word is blocked);
8: if the word does not contain $1'$ symbols anymore then
9:     Substitute all the symbols $1''$ with $1'$;
10: else
11:    Go to step 3;
12: if the word does not contain # symbols anymore then
13:    Delete all the symbols $1'$;
14:    Choose nondeterministically one of the following choices:
15:    case 1: In this case, the symbol restored at the end of the word was $x'_i$, but we assume
       that we should restore $x'_{i+1}$; if this assumption is false, the word is blocked or lost later
       in the computation. Replace $x'_i$ with $x'_{i+1}$; Replace all #' with #; Go to step 2;
16:    case 2: In this case, the symbol restored at the end of the word was $x'_i$, and we assume
       that we should start the restoring a new symbol; if this assumption is false, the word
       is blocked or lost later in the computation. Delete one symbol #'; Replace $x'_i$ with $x_i$;
       Replace all #' with #; Go to step 1;
17:    case 3: In this case, we assume that the suffix was completely restored; if this assumption
       is false, the word is blocked or lost later in the computation. If the word contains 1 it is
       lost; Delete one symbol #'; If the word contains #' it is lost; Replace $x'_i$ with $x_i$;
18: else
19:    Go to step 2;
20: The word is now from the set $\#^{q-1} ⧢ 1^{t-ik^q} ⧢ wx_i$.
```

It is not hard to see that the Algorithms *Store* and *Restore* work correctly (i. e., the statements made in last steps of the algorithms, respectively, hold for any input word that verifies the required form) and that the Algorithms really work in the way we want the two subnetworks $\mathcal{N}_{st}$ and $\mathcal{N}_{rst}$ to work.

In what follows, we show how the Algorithms *Store* and *Restore* can be effectively implemented using networks of evolutionary processors.

The network $\mathcal{N}_{st}$ has $11+(k-1)$ nodes. They are defined by:

$H_1 = (\{\, x_i \to x'_i \,\}, U^* \{\, x_i \mid 1 \le i \le k \,\}, U^*)$;
this implements step 1 of *Store*.

$H_2 = (\{\, \lambda \to \#' \,\}, U^* \{\, x'_i \mid 1 \le i \le k \,\}, U^*)$;
this implements step 2 of *Store*.

$H_3 = (\{\, \lambda \to 1' \,\}, U^* \{\, x'_i \mid 1 \le i \le k \,\}, U^*)$;
this implements step 3 of *Store*.

$H_4 = (\{\, 1' \to 1'' \,\} \cup \{\, a \to \perp \mid a \in U \,\}, U^*, U^*)$;
this implements step 5 of *Store*.

$H'_i = (\{\, \lambda \to 1'' \,\}, U^*, U^*), 1 \le i \le k-1$;
these nodes implement step 6 of *Store*,
H_4 and H'_i for $1 \le i \le k-1$ implement the cycle $4-7$ of *Store*.

$H_5 = (\{\, \# \to \#' \,\} \cup \{\, a \to \perp \mid a \in U \,\}, (U \setminus \{\, 1' \,\})^*, U^*)$;
this implements steps $8, 9$ of *Store*.

$H_6 = (\{\, 1'' \to 1' \,\}, U^*, (U \setminus \{\, 1'' \,\})^*)$;
this implements step 10 of *Store*.

$H_7 = (\{\, \#' \to \# \,\}, (U \setminus \{\, 1', \# \,\})^*, (U \setminus \{\#'\})^*)$;
this implements steps $13, 14$ of *Store*.

$H_8 = (\{\, x'_1 \to \lambda \,\}, U^* \{\, x'_1 \,\}, U^*)$;
this implements step 15 of *Store*.

$H_9 = (\{\, 1' \to 1 \,\}, U^*, (U \setminus \{1'\})^*)$;
this implements step 16 of *Store*.

$H_{10} = (\{\, \# \to \#' \,\}, U^* \{\, x'_i \mid 2 \le i \le k \,\}, U^*)$;
this implements step 18 of *Store*.

$H_{11} = (\{\, x'_i \to x'_{i-1} \mid 2 \le i \le k \,\}, U^*, U^*)$;
this implements step 19 of *Store*.

The edges of this subnetwork are:

(N_{in}, H_1), (H_1, H_2), (H_2, H_3), (H_3, H_4),

(H_4, H'_i) for $1 \le i \le k-1$,

(H'_i, H'_{i+1}) for $1 \le i \le k-2$,

(H'_{k-1}, H_4), (H'_{k-1}, H_5), (H'_{k-1}, H_7), (H_5, H_6), (H_6, H_4),

(H_7, H_8), (H_7, N_{10}), (H_8, H_9), (H_9, N_{in}), (H_{10}, H_{11}), (H_{11}, H_3).

From the explanations provided in the definitions of the nodes it is clear that the network implements the operations of the Algorithm *Store*. Moreover, the edges ensure that the subnetwork executes the steps of the algorithm *Store* in the correct order.

The network $\mathcal{N}_{rst}$ has $13+(k-1)$ nodes. They are defined by:

$N_1 = (\{\,\lambda \to x_1'\,\}, U^*, U^*)$;
this implements step 1 of *Restore*.
$N_2 = (\{\,\# \to \#'\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*\{\,x_i' \mid 1 \le i \le k\,\}, U^*)$;
this implements step 2 of *Restore*.
$N_3 = (\{\,1 \to 1''\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*x_1', U^*)$;
this implements partly steps 3 and 4 of *Restore*.
$N_4 = (\{\,1' \to 1''\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*, U^*)$;
this implements steps 3 and 6 of *Restore*.
$N_i' = (\{\,1 \to 1''\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*, U^*), 1 \le i \le k-1$;
these nodes implement step 7 and complete step 4 of *Restore*.
$N_6 = (\{\,1' \to \lambda\,\}, (U \setminus \{\#\})^*, (U \setminus \{\,1'\,\})^*)$;
this implements step 13 of *Restore*.
$N_7 = (\{\,x_i' \to x_{i+1}' \mid 1 \le i \le k-1\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*, U^*)$;
$N_8 = (\{\,\#' \to \#\,\}, U^*, (U \setminus \{\,\#'\,\})^*)$;
the last 2 nodes implement step 14 of *Restore*.
$N_9 = (\{\,\#' \to \lambda\,\}, U^*, U^*)$;
$N_{10} = (\{\,x_i' \to x_i \mid 1 \le i \le k\,\}, (U \setminus \{\#\})^*, (U \setminus \{\,1'\,\})^*, (U \setminus V')^*)$;
$N_{11} = (\{\,\#' \to \#\,\} \cup \{\,a \to \perp \mid a \in U\,\}, U^*, (U \setminus \{\,\#'\,\})^*)$;
the last 3 nodes implement step 15 of *Restore*.
$N_{12} = (\{\,\#' \to \lambda\,\}, (U \setminus \{\,1\,\})^*, U^*)$;
$N_{13} = (\{\,x_i' \to x_i \mid 1 \le i \le k\,\}, (U \setminus \{\#'\})^*, U^*)$;
the last 2 nodes implement step 16 of *Restore*.

The edges of this subnetwork are:

(N_{in}, N_1), (N_1, N_2), (N_2, N_3), (N_2, N_4),

(N_3, N_i') and (N_4, N_i') for $1 \le i \le k-1$,

(N_i', N_{i+1}') for $1 \le i \le k-2$,

(N_{k-1}', N_3), (N_{k-1}', N_5), (N_5, N_6), (N_6, N_7), (N_6, N_9), (N_6, N_{12}),

(N_7, N_8), (N_8, N_2), (N_9, N_{10}), (N_{10}, N_{11}), (N_{11}, N_1), (N_{12}, N_{13}), (N_{13}, N_{in}).

Once more, it is rather easy to see that the nodes and the edges defined above ensure that the subnetwork $\mathcal{N}_{rst}$ implements exactly the steps of the Algorithm *Restore* and that they are executed in the correct order.

The computation of $\mathcal{N}$ is rather simple to imagine: the node N_{in} chooses (nondeterministically) what should be done next: a reversed rule of the grammar should be applied, the righthand side of a rule should be located in the word and the suffix found after that location should be saved as an unary number, the saved prefix should be restored, or the current word should be sent to the output subnetwork and accepted if it equals the starting symbol S. The language accepted by the network consists of all the words that can be rewritten according to the reversed rules of G (between the application of two such rules it may be possible to store a suffix and then, later, restore it), such that we finally obtain S. This language is exactly L. □

According to Lemma 2.2 and the relations between the classes of $\mathcal{G}$, depicted in Figure 1, we get the following corollary.

Corollary 3.2 $\mathcal{A}(\mathit{REG}) = \mathcal{A}(\mathit{DEF}) = \mathcal{A}(\mathit{ORD}) = \mathcal{A}(\mathit{NC}) = \mathcal{A}(\mathit{PS}) = \mathit{RE}$.

Also networks with suffix-closed regular filters only are computationally complete.

Theorem 3.3 $\mathcal{A}(\mathit{SUF}) = \mathit{RE}$.

Proof. Let $L \subseteq V^*$ be a recursively enumerable language. According to Theorem 3.1 there exists a network $\mathcal{N} = (V, U, N_1, N_2, \ldots, N_n, E, N_1, N_n)$ with evolutionary processors and filters from *COMB* such that $L(\mathcal{N}) = L$. We can assume without losing generality that the output node N_n has no rules. For any node $N_i = (M_i, I_i, O_i)$, we construct the sets

$$\begin{aligned} I_i' &= \{X\}I_i\{Y\} \cup \mathit{Suf}(I_i)\{Y\} \cup \{\lambda\},\\ O_i' &= \{X\}O_i\{Y\} \cup \mathit{Suf}(O_i)\{Y\} \cup \{\lambda\}, \end{aligned}$$

where X and Y are two new symbols. By definition, I_i' and O_i' are suffix-closed.

We consider now the network

$$\mathcal{N}' = (V, U \cup \{X, Y\}, N_0, N_0', N_1', N_2', \ldots, N_n', N_{n+1}', E', N_0, N_{n+1}')$$

with

$$\begin{aligned} N_0 &= (\{\lambda \to X\}, \emptyset, XU^* \cup \{U^*\}),\\ N_0' &= (\{\lambda \to Y\}, XU^*Y \cup \{U^*Y\}, XU^*Y \cup \{U^*Y\}),\\ N_i' &= (M_i, I_i', O_i') \text{ for } 1 \le i \le n-1,\\ N_n' &= (\{X \to \lambda,\ Y \to \lambda\}, I_n', U^*),\\ N_{n+1}' &= (\emptyset, U^*, \emptyset),\\ E' &= E \cup \{\, (N_0, N_0'), (N_0', N_1'), (N_n', N_{n+1}') \,\}. \end{aligned}$$

It is obvious that the filters of all the nodes defined above are suffix-closed, too. Thus, $\mathcal{N}'$ is a network of type *SUF*.

We now prove that $L(\mathcal{N}) = L(\mathcal{N}')$. Let w be an input word for the two networks. The ANEP $\mathcal{N}'$ transforms it into XwY in the nodes N_0 and N_0', and then the word is sent to N_1'; no other processing can be done. Further, the string is processed in $\mathcal{N}'$ according to the rules of M_i, $1 \le i \le n-1$, only; the obtained strings can only be sent to nodes N_s', $1 \le s \le n$. Thus, we simulate a derivation in $\mathcal{N}$ (in $\mathcal{N}'$ we have an X in front of and a Y behind the word w

occurring in $\mathcal{N}$) and a string enters into N'_n if and only if a string obtained from w could have entered N_n. Now, in $\mathcal{N}'$, the X and Y symbols are removed and the resulting word is sent to N'_{n+1}. Hence, $L(\mathcal{N}') = L(\mathcal{N})$. □

Theorem 3.4 *$\mathcal{A}(UF) = RE$.*

Proof. By Corollary 3.2, the relations of Figure 1, and Lemma 2.2, we have $\mathcal{A}(UF) \subseteq RE$.

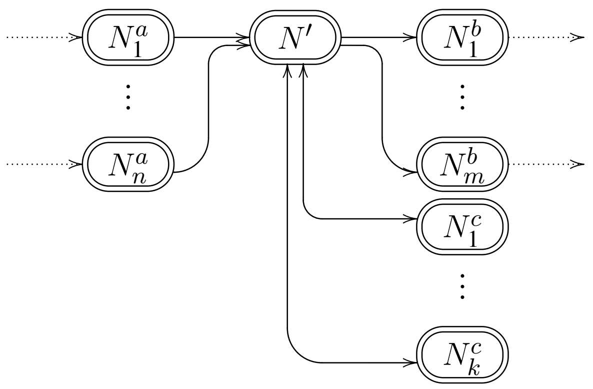
Let $L \subseteq V^*$ be a recursively enumerable language. By Theorem 3.1, we can assume that L is accepted by an ANEP $\mathcal{N}$ with combinational filters, input alphabet V and working alphabet U. We show how this network can be simulated by another network with filters from *UF* and the same input and working alphabets.

Let N be a processor of the network $\mathcal{N}$. Then N has the form

$$N = (M, V_1^*\{a_1, a_2, \dots, a_n\}, V_2^*\{b_1, b_2, \dots, b_m\})$$

with $V_1 \subseteq U$, $a_i \in V_1$ for $1 \le i \le n$, $V_2 \subseteq U$, and $b_j \in V_2$ for $1 \le j \le m$. Let $c_1, c_2, \dots, c_k$ be the other letters of V_2: $\{c_1, c_2, \dots, c_k\} = V_2 \setminus \{b_1, b_2, \dots, b_m\}$. We replace the node N by the subnetwork given in the following figure where the nodes are defined as follows:

$$\begin{aligned} N_i^a &= (\emptyset, V_1^*\{a_i\}, U^*) \text{ for } 1 \le i \le n,\\ N' &= (M, U^*, V_2^*),\\ N_i^b &= (\emptyset, U^*, V_2^*\{b_i\}) \text{ for } 1 \le i \le m,\\ N_i^c &= (\emptyset, U^*, V_2^*\{c_i\}) \text{ for } 1 \le i \le k.\end{aligned}$$



Every edge from a node K to the node N is replaced by edges from K to every node N_i^a for $1 \le i \le n$. Every edge from the node N to a node K is replaced by edges from every node N_i^b for $1 \le i \le m$ to A.

Then a word w passes the node N if and only if it passes the subnetwork defined above. Indeed, w enters the subnetwork if and only if it passes the input filter of one of the nodes N_i^a, which is equivalent to passing the input filter of N. Then a rule is applied to it; this is simulated in the subnetwork in the node N', where every string that entered the subnetwork enters after an evolutionary and a communication step. Further, the string exits the node N if it belongs to the set V_2^* and its last letter is one of the b_i with $1 \le i \le m$; equivalently, in the subnetwork, the word remains in the node N' if it does not belong to V_2^*, otherwise it is communicated to the nodes N_i^b for $1 \le i \le m$ and N_i^c for $1 \le i \le k$ and exits the subnetwork if it passes the output filter of one of the nodes N_i^b. If it does not pass such an output filter, then it passes the output filter of one of the nodes N_i^c and is returned to node N' (which simulates that it remains in the node N as well).

Thus, the construction does not change the language and the obtained network accepts L, too. Moreover, if $V = \{c_1, c_2, \dots, c_r\}$, then

$$V^*\{a\} = (\{c_1\}^*\{c_2\}^* \cdots \{c_r\}^*)^*\{a\}.$$

Therefore all filters of the constructed network above are union-free. Hence, $L \in \mathcal{A}(UF)$. This proves the other inclusion $RE \subseteq \mathcal{A}(UF)$. □

4. Lower Bounds

In this section, we show that the results presented in the previous section are optimal with respect to the hierarchy depicted in Figure 1. We also show a series of proper inclusion results between the classes $\mathcal{A}(X)$ with $X \in \mathcal{G}$.

Theorem 4.1 *$\mathcal{A}(CIRC)$ contains only circular languages.*

Proof. Let $\mathcal{N} = (V, U, N_1, N_2, \ldots, N_n, E, N_{n_i}, N_{n_o})$ be an accepting network of evolutionary processors with filters from the class *CIRC*. Let w be a word over V and w' a circular permutation of w. Assume that there exists $t+1$ words $w'_0 = w', w'_1, w'_2, \ldots, w'_t$ and $t+1$ nodes $N'_0 = N_{n_i}, N'_1, \ldots, N'_t$, such that w'_i was derived from w'_{i-1} in one evolutionary step, in which a rule r_i was applied in the node N'_{i-1} to w'_{i-1}, and then w'_i entered N'_{i+1} for $1 \leq i \leq t-1$. It is rather easy to show, by induction on t, that there exists $t+1$ words $w_0 = w, w_1, \ldots, w_t$ such that w_i is a circular permutation of w'_i for $0 \leq i \leq t$, and w_i was derived from w_{i-1} in one evolutionary step, in which a rule r_i was applied in the node N'_{i-1} to w_{i-1}, and then w_i entered N'_{i+1} for $1 \leq i \leq t$.

It follows that if w is accepted by $\mathcal{N}$ then any circular permutation of w is also accepted by the network. Thus, the language accepted by $\mathcal{N}$ is circular. □

The following corollary is now immediate.

Corollary 4.2 *$\mathcal{A}(CIRC)$ is a proper subset of RE.*

We can show the following result in a manner very similar to the proof of Theorem 4.1, by simply replacing circular permutations with any type of permutations.

Theorem 4.3 *$\mathcal{A}(COMM)$ contains only commutative languages.* □

According to the previous theorem, the Lemmas 2.2 and 2.3, and the fact that not all regular circular languages are commutative, the following corollary is immediate.

Corollary 4.4 *$\mathcal{A}(COMM)$ is a proper subset of $\mathcal{A}(CIRC)$.*

We can also show the following result.

Theorem 4.5 *$\mathcal{A}(NIL)$ is a proper subset of RE.*

Proof. Let $L = \{aw \mid w \in \{a,b\}^*\}$. We show that L cannot be accepted by a network with filters from *NIL*.

For the sake of a contradiction, assume that there exists an ANEP

$$\mathcal{N} = (V, U, N_1, N_2, \ldots, N_n, E, N_{n_i}, N_{n_o})$$

with filters from *NIL* such that $L(\mathcal{N}) = L$.

Let us assume that none of the filters of $\mathcal{N}$ is finite. Let $N'_0, N'_1, \ldots, N'_t$ with $N'_0 = N_{n_i}$ and $N'_t = N_{n_o}$ be the processors on a path from the input node to the output node. Let ℓ be the maximum length of a word w that does not belong to any of the filters of these nodes. Let w be the word $bab^{\ell+t}$. Clearly, w will be processed in the first node N'_0, then it can enter N'_1, where it is further processed and sent to N'_2, and so on. The word is not blocked by any filter since it will be longer than any of the words that are blocked by these filters. So it reaches the output node and it is accepted, a contradiction.

Consequently, at least one of the nodes of $\mathcal{N}$ has a finite filter. Moreover, by arguments similar to the above, there is no path connecting the input node and the output node that contains only nodes with infinite filters. Let ℓ_1 be the maximum length of a word that appears in one of the finite filters of the network's nodes and let ℓ_2 be the maximum length of a word that does not belong to an infinite filter of the network's nodes. Let $\ell = \max(\ell_1, \ell_2)$. Now consider the word $ab^{\ell+1}$. This word belongs to the language L so it also belongs to the language $L(\mathcal{N})$; for each word w', that can be derived from this word by $\mathcal{N}$, denote by $\#(w')$ the number of the initial b symbols that are still present in the word to which we add the number of the symbols that are obtained by (iterated) substitutions from the original b symbols. Consider the $t+1$ words $w_0 = w, w_1, w_2, \ldots, w_t$ and the $t+1$ nodes $N'_0 = N_{n_i}, N'_1, \ldots, N'_t = N_{n_o}$, such that w_i was derived from w_{i-1} in one evolutionary step, in the node N'_{i-1}, and then w_i entered N_{i+1} for $1 \leq i \leq t-1$. Assume that j is minimum such that N'_j has a finite filter and suppose that N'_j has the input filter finite. Clearly, there exists a number $k \leq j$ such that $\#(w_k) < \#(w_{k+1})$; otherwise, the string w_j contains more than ℓ letters and cannot enter the processor N'_j. So N'_k is a deletion node and one of the initial b symbols of the input word or a symbol obtained from these symbols is deleted there. Assume that the pth initial b or the symbol derived from it is deleted in this node. Let w'_i the word obtained from w_i by deleting the symbol derived from the pth initial b symbol so far (denoted in the following x_i) for $i \leq k$. Now it is not hard to see that one can obtain the words $bw'_0, x_1w'_1, x_2w'_2, \ldots, x_kw'_k, w_{k+1}, \ldots, w_t$ on the same path given by the nodes $N'_0 = N_{n_i}, N'_1, \ldots, N'_t = N_{n_o}$ and using the same rules. But this means that the word bw'_0 is accepted by $\mathcal{N}$, a contradiction. A similar argument works for the case when N'_j has the output filter finite.

Thus, we have shown that in all the cases we reach a contradiction and this concludes our proof. $\square$

By Lemma 2.2 we can now also derive the following corollary.

Corollary 4.6 *$\mathcal{A}(MON)$ and $\mathcal{A}(FIN)$ are proper subsets of RE.*

Also, since $\mathcal{A}(MON)$ is included in $\mathcal{A}(COMM)$, by Theorem 4.3, it follows that all the languages in $\mathcal{A}(MON)$ are commutative. On the other hand, it is not hard to see that $\mathcal{A}(NIL)$ contains also finite languages that are not commutative. Thus, we obtain the following corollary.

Corollary 4.7 *$\mathcal{A}(MON)$ is a proper subset of $\mathcal{A}(NIL)$.*

Finally, we show that the computational power of networks with finite filters is weaker than the computational power of networks with nilpotent filters, while networks with monoidal filters are as powerful as networks with commutative filters.

Theorem 4.8 *$\mathcal{A}(FIN)$ is a proper subset of $\mathcal{A}(NIL)$.*

Proof. The first remark is that the non-regular context-free language

$$L = \{ w \mid w \in \{a,b\}^*, |w|_a = |w|_b \}$$

can be accepted by an ANEP with monoidal filters.

Indeed consider the ANEP

$$\mathcal{N} = (\{a,b\}, \{a,b,X,Y,F\}, N_1, N_2, N_3, E, N_1, N_3)$$

with $E = \{ (N_1,N_2), (N_2,N_1), (N_2,N_3) \}$ and the nodes defined as follows:

$$\begin{aligned} N_1 &= (\{ a \to X, Y \to F, b \to F \}, \{X,Y,a,b\}^*, \{X,Y,a,b\}^*), \\ N_2 &= (\{ b \to Y, X \to F, a \to F \}, \{X,Y,a,b\}^*, \{X,Y,a,b\}^*), \\ N_3 &= (\emptyset, \{X,Y\}^*, \emptyset). \end{aligned}$$

The ANEP $\mathcal{N}$ works as follows: In node N_1, an a symbol becomes X and the word goes to node N_2 where a b symbols becomes Y. The process is iterated until no a and b symbols remain. If this case occurs in node 2 then the string enters the output node and it is accepted (an equal number of a and b symbols was found); otherwise, the obtained words are trapped in node N_1 (when the initial word had more b symbols) or in node N_2 (when the initial word had more a symbols). Thus, $L(\mathcal{N}) = L$.

Further we look on the form of the languages accepted by ANEPs with finite filters. Let

$$\mathcal{N} = (V, U, N_1, N_2, \dots, N_n, E, N_{n_i}, N_{n_o})$$

be such a network, and denote by O_{n_i} the output filter of the input node. Denote by O'_{n_i} the subset $O_{n_i} \cap L(\mathcal{N})$. Notice that if N_{n_i} is a substitution or an insertion node then the accepted language is a finite one: It consists only of the words that can be transformed by the rules of the input node into the words of O'_{n_i}. If N_{n_i} is a deletion node then we can easily see that the language accepted by $\mathcal{N}$ is $O'_{n_i} V_1^*$, provided that the rules of N_{n_i} are $M_{n_i} = \{ a \to \lambda \mid a \in V_1 \}$.

According to the above the class of languages accepted by ANEPs with finite filters is a proper subclass of *REG*. Therefore, L cannot be accepted by such a network. □

Remark 4.9 *The previous proof, combined with the fact that $\mathcal{A}(FIN)$ contains non-commutative languages, shows that $\mathcal{A}(MON)$ and $\mathcal{A}(FIN)$ are incomparable. Similar arguments show that $\mathcal{A}(FIN)$ is incomparable with $\mathcal{A}(CIRC)$. Finally, $\mathcal{A}(NIL)$ and $\mathcal{A}(CIRC)$ are also incomparable. Indeed, $\mathcal{A}(NIL)$ contains languages that are not in $\mathcal{A}(CIRC)$, according to the above. On the other hand, $\mathcal{A}(CIRC)$ also contains languages that are not in $\mathcal{A}(NIL)$; for instance, the language $L = \{w \mid w \in \{a,b,c\}^*, w$ is a circular permutation of $(abc)^n$, for some $n \in \mathbb{N}\}$ is clearly in $\mathcal{A}(CIRC)$, but one can show, similarly to the proof of Theorem 4.5, that if a network with nilpotent filters accepts L then this network accepts also words starting with cb, a contradiction.*

Theorem 4.10 $\mathcal{A}(MON) = \mathcal{A}(COMM)$.

Proof. We already know that $\mathcal{A}(MON) \subseteq \mathcal{A}(COMM)$, so we just have to show the inverse inclusion.

For this, let L be a language which is accepted by a network

$$\mathcal{N} = (V, U, N_1, N_2, \ldots, N_n, E, N_{n_i}, N_{n_o})$$

with filters from *COMM*. We will construct an ANEP $\mathcal{N}'$ with monoidal filters that accepts the language L.

First consider a node $N = (M, I, O)$ of the network $\mathcal{N}$, such that I and O are commutative. We construct a new network by replacing this node with four other nodes:

$$\begin{aligned} N_1 &= (\emptyset, U^*, I), \\ N_2 &= (M, U^*, U^*), \\ N_3 &= (\emptyset, U^*, O), \\ N_4 &= (\emptyset, U^*, U^* \setminus O). \end{aligned}$$

These nodes are connected by the edges (N_1, N_2), (N_2, N_3), (N_2, N_4), and (N_4, N_2). Furthermore, whenever we had an edge (K, N) in $\mathcal{N}$ we will have an edge (K, N_1) in the new network, and whenever we had an edge (N, K) in $\mathcal{N}$ then we will have an edge (N_3, K) in the new network.

It is not hard to see that the network obtained by replacing N with the subnetwork we have just described accepts the same language, and still has commutative filters. Moreover, by replacing, one at a time, each node that has commutative filters, according to the procedure above, leads to a new network $\mathcal{N}'$. This network has the property that each of its nodes that has commutative filters has no rules, and has a monoid as input filter.

Next we will construct a new network $\mathcal{N}''$ that accepts the same language and has only monoidal filters. The idea that we implement is the following: we replace each node N of $\mathcal{N}'$ that has an output filter O which is not a monoid by a subnetwork that actually tests whether the communicated word is part of O.

For this, assume that $N = (\emptyset, V_1^*, O)$ is such a node with $O \notin \mathit{MON}$, $U' = \{\, a' \mid a \in U \,\}$, and $\perp$ is a symbol not contained in U. Let $T = (V, Q, q_0, F, \delta)$ be a deterministic finite automaton that accepts O. For each triple $t = (q_1, a, q_2)$ with $a \in V$ and $q_1, q_2 \in Q$ such that $\delta(q_1, a) = q_2$, we construct two nodes:

$$\begin{aligned} N_1^t &= (\{\, a \to a', q_1 \to \perp \,\}, (U \cup U' \cup \{q_1\})^*, (U \cup U' \cup \{q_1\})^*), \\ N_2^t &= (\{\, q_1 \to q_2 \,\}, (U \cup U' \cup \{q_1\})^*, (U \cup U' \cup \{q_2\})^*). \end{aligned}$$

Then we also construct the nodes

$$\begin{aligned} N_{in} &= (\{\, \lambda \to q_0 \,\}, V_1^*, (U \cup \{q_0\})^*), \\ N_{post} &= (\{\, a' \to a \mid a \in U \,\}, (U' \cup F)^*, (U \cup F)^*), \\ N_{out} &= (\{\, q_f \to \lambda \mid q_f \in F \,\}, (U \cup F)^*, U^*). \end{aligned}$$

If we have an edge (K, N) in the network $\mathcal{N}'$, we will now have the edge (K, N_{in}), and if we have an edge (N, K) in $\mathcal{N}'$, then we will now have the edge (N_{out}, K). Also we have the following edges:

- (N_{in}, N_1^t) where t is a triple (q_0, a, q) for some $a \in V$ and $q \in Q$ such that $\delta(q_0, a) = q$;
- (N_1^t, N_2^t) for each triple $t = (q_1, a, q_2)$ with $a \in V$ and $q_1, q_2 \in Q$ such that $\delta(q_1, a) = q_2$;

- (N_2^t, N_1^r) for all the triples $t = (q_1, a, q_2)$ and $r = (q_2, b, q_3)$ with $a, b \in V$ and $q_1, q_2, q_3 \in Q$ such that $\delta(q_1, a) = q_2$ and $\delta(q_2, b) = q_3$;
- (N_3^t, N_{post}) for all the triples $t = (q_1, a, q_f)$ with $a \in V$, $q_1 \in Q$ and $q_f \in F$ such that $\delta(q_1, a) = q_f$;
- (N_{post}, N_{out}).

We can now show that each time a word w passes the node N then it also passes the subnetwork defined above. The key idea is that we will not check if $w \in O$, but if a non-deterministically chosen permutation of w is in O. First, assume that w enters N and, consequently, it also enters N_{in}. Now, in the subnetwork, the symbol q_0 is appended to the word, and it is sent to a subnetwork associated with a triple (q_0, a, q_1) defined as above. In N_1^t, an a symbol becomes a', meaning that it was read by the automaton accepting O; if no a exists in the word, q_0 becomes $\perp$ and the word is trapped in the node. Next the word goes to N_2^t where q_0 becomes q_1. Then the word is sent to N_1^r where $r = (q_1, b, q_2)$ and the process described above is iterated. This continues until the symbol representing the state of the automaton represents a final state and the rest of the symbols are from U'; that is, the whole word was read and the automaton reached a final state or a permutation of w is in O. In this case, the word goes to N_{post} where the markings of the symbols are removed and then it is sent to N_{out} where the state-symbol is removed. Finally, the string exists the subnetwork exactly as it exits the node N.

Note that the filters of the above subnetwork ensure that at every moment we have at most one state-symbol, and no other derivations, than the one we discussed, can occur.

Clearly, the network obtained by replacing N with the associated subnetwork accepts the same language and has only monoidal filters. Moreover, if we replace each node that has commutative filters according to the procedure above, we obtain a new network $\mathcal{N}''$ that accepts L and has only monoidal filters. This concludes our proof. □

5. Conclusions

The results we have obtained can be seen in Figure 2. A solid arrow from X to Y denotes $X \subset Y$ (X is a proper subset of Y); if two classes are not connected by a directed path then they are incomparable.

$$\begin{tikzcd}
 & \begin{array}{l} RE = \mathcal{A}(REG) = \mathcal{A}(PS) = \mathcal{A}(NC) = \mathcal{A}(ORD) \\ \quad = \mathcal{A}(DEF) = \mathcal{A}(COMB) = \mathcal{A}(UF) = \mathcal{A}(SUF) \end{array} & \\
\mathcal{A}(NIL) \arrow[ur] & & \mathcal{A}(CIRC) \arrow[ul] \\
\mathcal{A}(FIN) \arrow[u] & \mathcal{A}(MON) = \mathcal{A}(COMM) \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 2: Hierarchy of the language classes accepted by ANEPs with subregular filters

It remains to be studied how the different types of ANEPs can be used to accept languages that are important in practice (for instance, different subclasses of the Chomsky hierarchy). Also, it seems interesting to us to define computational complexity measures for these networks as it was defined in the case of ANEPs with left/right operations and random-context filters and to investigate whether such networks can be used to accept efficiently different classes of languages. Finally, we will investigate the existence of trade-offs between the efficiency of accepting a given language and the structural complexity of the filters used.

References

[1] J. DASSOW, F. MANEA, B. TRUTHE, Networks of Evolutionary Processors with Subregular Filters. In: *5th International Conference on Language and Automata Theory and Applications (LATA), Tarragona, Spain, May 26–31, 2011, Proceedings*. LNCS, Springer-Verlag, 2011. Accepted.

[2] J. DASSOW, B. TRUTHE, On Networks of Evolutionary Processors with State Limited Filters. In: H. BORDIHN, R. FREUND, T. HINZE, M. HOLZER, M. KUTRIB, F. OTTO (eds.), *Second Workshop on Non-Classical Models of Automata and Applications (NCMA), Jena, Germany, August 23–24, 2010, Proceedings*. books@ocg.at 263, Österreichische Computer Gesellschaft, Austria, 2010, 57–70.

[3] I. M. HAVEL, The theory of regular events II. *Kybernetika* **5** (1969) 6, 520–544.

[4] W. D. HILLIS, *The Connection Machine*. MIT Press, Cambridge, MA, USA, 1986.

[5] F. MANEA, M. MARGENSTERN, V. MITRANA, M. J. PEREZ-JIMENEZ, A New Characterization of NP, P and PSPACE with Accepting Hybrid Networks of Evolutionary Processors. *Theor. Comp. Sys.* **46** (2010) 2, 174–192.

[6] 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** (2007) 4, 753–771.

[7] F. MANEA, C. MARTÍN-VIDE, V. MITRANA, Accepting Networks of Evolutionary Word and Picture Processors: A Survey. In: C. MARTÍN-VIDE (ed.), *Scientific Applications of Language Methods*. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory 2, World Scientific, 2010, 525–560.

[8] M. MARGENSTERN, V. MITRANA, M. J. PÉREZ-JIMÉNEZ, Accepting Hybrid Networks of Evolutionary Processors. In: *DNA Computing, 10th International Workshop on DNA Computing*. LNCS, Springer-Verlag Berlin, 2004, 235–246.

[9] G. ROZENBERG, A. SALOMAA, *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[10] B. WIEDEMANN, *Vergleich der Leistungsfähigkeit endlicher determinierter Automaten*. Diplomarbeit, Universität Rostock, 1978.