☺ ⟶ ☺☺
AG Automaten und Formale Sprachen

Preprint AFL-2011-01
Otto-von-Guericke-Universität Magdeburg, Germany

Networks of Evolutionary Processors with Subregular Filters

Jürgen Dassow Florin Manea[(A)] Bianca Truthe

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

Abstract

In this paper we propose a hierarchy of classes of languages, generated by networks of evolutionary processors with the filters in several special classes of regular sets. More precisely, we show that the use of filters from the class of ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite and combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can generate all the recursively enumerable languages. On the other hand, the use of filters that are only finite languages allows only the generation of regular languages, but not all regular languages can be generated. If we use filters that are monoids, nilpotent languages or commutative regular languages, we obtain the same family of languages which contains non-context-free languages but not all regular languages. These results seem to be of interest because they provide both upper and lower bounds on the classes of languages that one can use as filters in a network of evolutionary processor in order to obtain a complete computational model.

1. Introduction

An important part of theoretical computer science is the study of problems and processes connected with regular sets. In the last years a lot of papers appeared in which, for such problems and processes, the effect of going from arbitrary regular sets to special regular sets was studied. We here mention four such topics.

– It is a classical result that any nondeterministic finite automaton with n states can be transformed into a deterministic one with 2^n states, which accepts the same language, and that this exponential blow-up with respect to the number of states is necessary in the worst cases. In [2], this problem is studied if one restricts to the case that the automata accept special regular languages only. It is shown, that the situation does not change for suffix-closed and star-free regular languages; however, for some classes of definite languages, the size of the deterministic automaton is bounded by $2^{n-1}+1$.

[(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*.

– A number α, $n \leq \alpha \leq 2^n$, is called magic (w. r. t. n), if there is no nondeterministic finite automaton with n states such that the minimal deterministic finite automaton has α states. It is known that no magic numbers exist if $n \geq 3$. This situation changes if one considers subregular families of languages. For instance, only the values α which satisfy the condition $n+1 \leq \alpha \leq 2^{n-1}+1$ are possible for prefix-free regular languages (see [15]).
– In the last 20 years the behaviour of the (nondeterministic) state complexity under operations is intensively studied, i. e., it is asked for the size of the minimal (non)deterministic finite automaton for the language obtained from languages with given sizes. For many operations, the worst case is exactly determined. It has been shown that one gets smaller sizes if one restricts to special regular languages (see [12], [13], [3], and [16]).
– In order to enlarge the generative power, some mechanisms connected with regular languages were introduced, which control the derivations in context-free grammars. For instance, the sequence of applied rules in a regularly controlled grammar, the current sentential form in a conditional grammar and the levels of the derivation tree in a tree controlled grammar have to belong to given regular languages. In the papers [7], [9], [8], and [10], the change in the generative power, if one restricts to special regular sets, is investigated.

In this paper we continue the research along this direction. We consider the effect of special regular filters for generating evolutionary networks.

Networks of language processors have been introduced in [6] by E. CSUHAJ-VARJÚ and A. SALOMAA. Such a network can be considered as a graph where the nodes are sets of productions and at any moment of time a language is associated with a node. 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 where the outgoing words have to satisfy an output condition given as a regular language (called output filter), and any node takes words sent by the other nodes if the words satisfy an input condition also given by a regular language (called input filter). The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, in [4] a special type of networks of language processors was introduced which are called networks with evolutionary processors because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found, e. g., in [4], [5], [17]. For instance. in [5], it was shown that networks of evolutionary processors are complete in that sense that they can generate any recursively enumerable language.

Modifications of evolutionary networks with evolutionary processors concern restrictions in the type of the nodes and the mode of applying a rule. In [1], it is investigated how the generative power behaves if one restricts to networks with at most two types of nodes only. Moreover, in the case that one allows that some insertions and deletions can only be performed at the begin or end of the word one has also restricted to special regular filters given by random context conditions.

In this paper, we modify the filters. We require that the filters have to belong to a special subset of the set of all regular languages. We show that the use of filters from the class of ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite and

combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can generate all the recursively enumerable languages. On the other hand, the use of filters that are only finite languages allows only the generation of regular languages, but not all regular languages can be generated. If we use filters that are monoids, nilpotent languages or commutative regular languages, we obtain the same family of languages which contains non-context-free languages but not all regular languages. These results seem to be of interest because they provide both upper and lower bounds on the classes of languages that one can use as filters in a network of evolutionary processor in order to obtain a complete computational model.

2. Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [18]). We here only recall some notations used in the paper.

By V^* we denote the set of all words (strings) over 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., $V_k = \bigcup_{i=0}^{k} V^i$.

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 productions 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.

By *REG*, *CF*, and *RE* we denote the families of regular, context-free, and recursively enumerable languages, respectively.

For a language L over V, we set

$$\begin{aligned} \mathit{Comm}(L) &= \{a_{i_1} \dots a_{i_n} \mid a_1 \dots a_n \in L,\ n \geq 1,\ \{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}\}, \\ \mathit{Circ}(L) &= \{vu \mid uv \in L,\ u, v \in V^*\}, \\ \mathit{Suf}(L) &= \{v \mid uv \in L,\ u, v \in V^*\} \end{aligned}$$

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 $L = \mathit{Circ}(L)$,
- *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 $x, y, z \in V^*$, $xy^k z \in L$ 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 families 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 family *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

$$\mathcal{G} = \{FIN, MON, COMB, DEF, NIL, COMM, CIRC, SUF, NC, PS, ORD, UF\}.$$

The relations between families of $\mathcal{G}$ are investigated e. g. in [14] and [20]. and their set-theoretic relations are given in Figure 1.

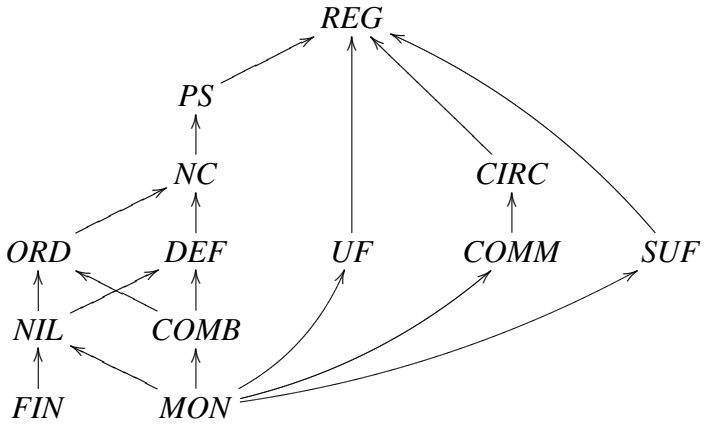


Figure 1: Hierarchy of subregular languages (an arrow from X to Y denotes $X \subset Y$, and if two families are not connected by a directed path then they are incomparable)

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

The productions 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 production $\alpha \to \beta$ such that $v = x\alpha y$ and $w = x\beta y$. If the rule p applied is important, we write $v \Longrightarrow_p w$.

We introduce insertion as a counterpart of deletion. We write $\lambda \to a$, where a is a letter. The application of an insertion $\lambda \to a$ derives from a word w any word $w_1 a w_2$ with $w = w_1 w_2$ for some (possibly empty) words w_1 and w_2.

We now introduce the basic concept of this paper, the networks of evolutionary processors (NEPs for short).

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

(i) *A network of evolutionary processors (of size n) with filters of the set X is a tuple*

$$\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$$

where
- *V is a finite alphabet,*
- *for $1 \leq i \leq n$, $N_i = (M_i, A_i, I_i, O_i)$ where*
  - *M_i is a set of rules of a certain type: $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$,*
  - *A_i is a finite subset of V^*,*
  - *I_i and O_i are languages from X over V,*
- *E is a subset of $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, and*
- *j is a natural number such that $1 \leq j \leq n$.*

(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 V^* 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 $1 \leq i \leq n$,*

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

The computation of an evolutionary network $\mathcal{N}$ is a sequence of configurations $C_t = (C_t(1), C_t(2), \ldots, C_t(n))$, $t \geq 0$, such that
- *$C_0 = (A_1, A_2, \ldots, A_n)$,*
- *for any $t \geq 0$, C_{2t} derives C_{2t+1} in one evolutionary step,*
- *for any $t \geq 0$, C_{2t+1} derives C_{2t+2} in one communication step.*

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

$$L(\mathcal{N}) = \bigcup_{t \geq 0} C_t(j)$$

where $C_t = (C_t(1), C_t(2), \ldots, C_t(n))$, $t \geq 0$ is the computation of $\mathcal{N}$.

Intuitively, a network with evolutionary processors is a graph consisting of some, say n, nodes $N_1, N_2, \ldots, N_n$ (called processors) and the set of edges given by E such that there is a directed edge from N_k to N_i if and only if $(k, i) \in E$. Any processor N_i consists of a set of evolutionary rules M_i, a set of words A_i, an input filter I_i and an output filter O_i. We say that N_i is a substitution node or a deletion node or an insertion node if $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$, 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. With any node N_i and any time moment $t \geq 0$ we associate a set $C_t(i)$ of words (the words contained in the node at time t). Initially, N_i contains the words of A_i. In an evolutionary step, we derive

from $C_t(i)$ all words applying rules from the set M_i. In a communication step, any processor N_i sends out all words $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ 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 $C_t(k) \cap O_k \cap I_i$. We start with an evolutionary step and then communication steps and evolutionary steps are alternately performed. The language consists of all words which are in the node N_j (also called the output node, j is chosen in advance) at some moment t, $t \geq 0$.

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

The following fact is obvious.

Lemma 2.2 *Let X and Y be subfamilies of REG such that $X \subseteq Y$. Then the inclusion*

$$\mathcal{E}(X) \subseteq \mathcal{E}(Y)$$

holds. □

The following theorem is known (see, e. g., [5]).

Theorem 2.3 $\mathcal{E}(\mathit{REG}) = \mathit{RE}$.

3. Some General Results

We start with some results which hold for every type of filters.

Lemma 3.1 *For every network $\mathcal{N}$ of evolutionary processors, there is a network $\mathcal{N}'$ of evolutionary processors that generates the same language as $\mathcal{N}$ and has the property that its output node N' has the form $N' = (\emptyset, \emptyset, I', O')$ for some regular languages I', O' over the network's working alphabet and no edge is leaving N'.*

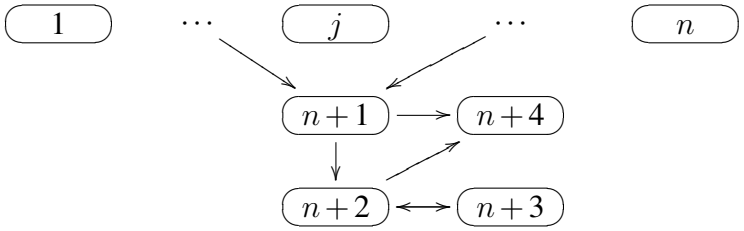
Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a network of evolutionary processors where the output node N_j has not the required property: $N_j \neq (\emptyset, \emptyset, I_j, O_j)$ for any sets I_j, O_j or there is an edge leaving node N_j. We define a new network $\mathcal{N}' = (V, N'_1, N'_2, \ldots, N'_{n+4}, E', n+4)$ by

$$\begin{aligned}
&N'_i = N_i \text{ for } 1 \leq i \leq n,\\
&N'_i = (M_i, \emptyset, I_i, O_i) \text{ for } n+1 \leq i \leq n+4,\\
&E' = E \cup \{\, (i, n+1) \mid (i,j) \in E \,\}\\
&\qquad \cup \{\, (n+1, n+2), (n+1, n+4), (n+2, n+3), (n+2, n+4), (n+3, n+2) \,\}
\end{aligned}$$

where

$$\begin{array}{llll}
M_{n+1} = \emptyset, & M_{n+2} = M_j, & M_{n+3} = \emptyset, & M_{n+4} = \emptyset,\\
A_{n+1} = A_j, & A_{n+2} = \emptyset, & A_{n+3} = \emptyset, & A_{n+4} = \emptyset,\\
I_{n+1} = I_j, & I_{n+2} = V^*, & I_{n+3} = V^* \setminus O_j, & I_{n+4} = V^*,\\
O_{n+1} = V^*, & O_{n+2} = V^*, & O_{n+3} = V^*, & O_{n+4} = V^*.
\end{array}$$

The network is illustrated below:



The new output node N'_{n+4} satisfies the condition because $N'_{n+4} = (\emptyset, \emptyset, V^*, V^*)$ and no edge leaves the node N'_{n+4}. We now show that $L(\mathcal{N}') = L(\mathcal{N})$.

The subnetwork consisting of $N'_1, N'_2, \ldots, N'_n$ is the same as $\mathcal{N}$. The initial sets of N'_j and N'_{n+1} as well as the input filters and incoming edges coincide. Hence, if a word w is in N_j at an even moment t, then w is also in this moment in node N'_j and N'_{n+1}. The word is then sent unchanged to the output node N'_{n+4}. Thus, $w \in L(\mathcal{N})$ and $w \in L(\mathcal{N}')$. Additionally, w is also sent to N'_{n+2} where the same rules as in N_j can be applied. Hence, if a word v is derived in N_j (and, hence, $v \in L(\mathcal{N})$) then v is derived in N'_{n+2} and will be sent to the output node in the next communication step, hence, $v \in L(\mathcal{N}')$. If the word v remains in N_j then a word $u \in L(\mathcal{N})$ will be derived from v in N_j. In $\mathcal{N}'$, the word v will also be sent to N'_{n+3} which takes the word and sends it back to N'_{n+2} where it will be derived to u which will be sent to the output node afterwards. Hence, as long as a word is modified in N_j, the same word is modified in N'_{n+2} with intermediate communication to N'_{n+3} and all these words also arrive in the output node. Thus, $L(\mathcal{N}) \subseteq L(\mathcal{N}')$.

Every word $w \in L(\mathcal{N}')$ came to node N'_{n+4} from node N'_{n+1} or N'_{n+2}. If it came from N'_{n+1} then the word was also in node N_j, hence, $w \in L(\mathcal{N})$. If it came from N'_{n+2} then it has been derived from a word v which came from N'_{n+1} or N'_{n+3}. If v came from N'_{n+1} then v was also in N_j and has derived w, hence, $w \in L(\mathcal{N})$. If v came from N'_{n+3} then v was previously in node N'_{n+2} and was derived from a word u. Furthermore, $v \notin O_j$. If u came from N'_{n+1} then u was also in N_j and has derived v which remained there and derived w, hence, $w \in L(\mathcal{N})$. If u came from N'_{n+3} then the argumentation can be repeated because for every word in u in N'_{n+2} there was a word $\tilde{u}$ in N'_{n+1} with $\tilde{u} \Longrightarrow^*_{M_j} u$ and all words during this derivation did not belong to O_j. Hence, $\tilde{u}$ was also in N_j where the same derivation of u took place. Thus, we obtain $L(\mathcal{N}') \subseteq L(\mathcal{N})$.

Since $L(\mathcal{N}') = L(\mathcal{N})$ the network $\mathcal{N}'$ has the required properties. □

Theorem 3.2 *Let $X \in \mathcal{G}$. Then each language $L \in X$ can be generated by a NEP $\mathcal{N}$ with at most two nodes and with filters from X.*

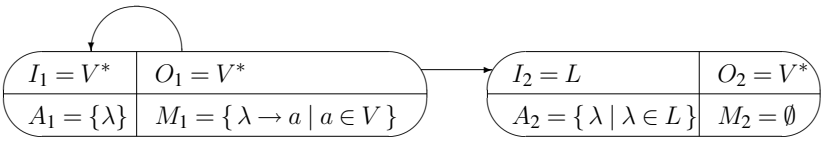
Proof. Let $X = \mathit{FIN}$. Let L be a finite set over V. Then the evolutionary network

$$(V, (\emptyset, L, \emptyset, \emptyset), \emptyset, 1)$$

with all filters from *FIN* generates L.

If $X \neq \mathit{FIN}$, then $\mathit{MON} \subseteq X$ holds by Figure 1. Moreover, let $L \in X$ be a language over an alphabet V.

We construct the NEP $\mathcal{N} = (V, N_1, N_2, E, 2)$ given as



Every word $w \in V^+$ will be derived in node N_1 and be communicated to node N_2 which accepts all words that also belong to L. The language generated by $\mathcal{N}$ is

$$L(\mathcal{N}) = A_2 \cup (V^+ \cap L) = L.$$

All filters are of type X. □

Corollary 3.3 *For each class $X \in \mathcal{G}$, we have $X \subseteq \mathcal{E}(X)$.* □

Corollary 3.4 *For each class $X \in \mathcal{G}$, we have MON $\subseteq \mathcal{E}(X)$.*

Proof. By the relations given in Figure 1 and Corollary 3.3, it is sufficient to show the inclusion $MON \subseteq \mathcal{E}(FIN)$. Let V be an alphabet and $L = V^*$. Then the evolutionary network $(V, (\{\, \lambda \to a \mid a \in V \,\}, \{\lambda\}, \emptyset, \emptyset), \emptyset, 1)$ with all filters from *FIN* generates L. Thus, any monoidal language $L = V^*$ belongs to $\mathcal{E}(FIN)$. □

4. Computationally Complete Cases

In this section we present the computational completeness of some families $\mathcal{E}(X)$.

Theorem 4.1 *$\mathcal{E}(SUF) = RE$ and $\mathcal{E}(CIRC) = RE$.*

Proof. First we show that $\mathcal{E}(SUF) = RE$.

Let L be a recursively enumerable language. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a network with evolutionary processors and filters from the class *REG* such that $L(\mathcal{N}) = L$. For any node $N_i = (M_i, A_i, I_i, O_i)$, we construct the sets

$$I_i' = \{X\} I_i \{Y\} \cup Suf(I_i)\{Y\} \cup \{\lambda\},$$
$$O_i' = \{X\} O_i \{Y\} \cup Suf(O_i)\{Y\} \cup \{\lambda\},$$

where X and Y are two new symbols. By definition, I_i' and O_i' are suffix-closed. We assume that the network $\mathcal{N}$ has the property $N_j = (\emptyset, \emptyset, I_j, O_j)$ and no edge leaves the output node (according to the previous Lemma).

We consider the network

$$\mathcal{N}' = (V \cup \{X, Y\}, N_1', N_2', \ldots, N_n', N_{n+1}', N_{n+2}', E', n+2)$$

with

$$\begin{aligned} N_i' &= (M_i, \{X\} A_i \{Y\}, I_i', O_i') \text{ for } 1 \le i \le n, \\ N_{n+1}' &= (\{X \to \lambda,\ Y \to \lambda\}, \emptyset, I_j', V^*), \\ N_{n+2}' &= (\emptyset, \emptyset, V^*, \emptyset), \\ E' &= E \cup \{\, (i, n+1) \mid (i,j) \in E \,\} \cup \{\, (n+1, n+2) \,\}. \end{aligned}$$

It is obvious that the filters of N'_{n+1} and N'_{n+2} are suffix-closed, too. Thus $\mathcal{N}'$ is a network of type *SUF*.

We now prove that $L(\mathcal{N}) = L(\mathcal{N}')$. We start with words of the form XwY and as long as these words are changed according to rules of M_i, $1 \leq i \leq n$, they can only be sent to nodes N'_s, $1 \leq s \leq n$, and N'_{n+1}. 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 get into N'_{n+1} exactly those words XwY whose subword w comes into N_j. Now X and Y are removed and the resulting word w is sent to N'_{n+2}. Other words cannot arrive in N'_{n+2} and other words do not appear in N_j. Hence, we have $L(\mathcal{N}') = L(\mathcal{N})$.

To show that $\mathcal{E}(\mathit{CIRC}) = \mathit{RE}$, we repeat the previous proof with the following modifications. We set

$$I'_i = \mathit{Circ}(\{X\}I_i\{Y\}) \text{ and } O'_i = \mathit{Circ}(\{X\}O_i\{Y\}) \text{ for } 1 \leq i \leq n.$$

This ensures that $\mathit{Circ}(F) = F$ for all filters F of the new network $\mathcal{N}'$. Then the proof proceeds as in the case of suffix-closed filters. □

Theorem 4.2 $\mathcal{E}(\mathit{DEF}) = \mathit{RE}$.

Proof. 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 productions have one of the following forms:

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

We construct a network of evolutionary processors with definite filters only that simulates a phrase structure grammar in Kuroda normal form.

Let $G = (N,T,P,S)$ be a grammar in Kuroda normal form. Further, let $V = N \cup T$ and let $x_1, x_2, \ldots, x_s$ be the elements of V.

Let $p = \alpha \to \beta$ be a rule of P and $w\alpha a_t a_{t-1} \cdots a_1$ be a sentential form of the grammar G with $w \in V^*$ and $a_i \in V$ for all natural numbers i with $1 \leq i \leq t$.

The idea of the simulation is to store the letters $a_1, a_2, \ldots, a_t$ together with their positions in the suffix somewhere else in the word to obtain the subword α in the end of the word. There it can be replaced by the right hand side β of the rule (by definite filters it can be checked at the end of a word that the rule is applied correctly). After that, the letters $a_1, a_2, \ldots, a_t$ are restored at their correct positions.

Since a word can be arbitrarily large, the position of a letter a_i can be an arbitrarily large number and hence cannot be represented by a single symbol from a finite repository. The trick here is to encode the position by the number of occurrences of a special symbol respresenting a_i. To be more precise, we encode the position i of a letter a by 2^i occurrences of the symbol $[a]$. If a symbol a occurs at positions $i_1, i_2, \ldots, i_p$, then the number of occurrences of the symbol $[a]$ in the word will be $2^{i_1} + 2^{i_2} + \cdots + 2^{i_p}$. Hence, the number of occurrences of a symbol $[a]$ in a word – read as a binary number – indicates by '1' at which positions in the suffix $a_t a_{t-1} \cdots a_0$ the letter a occurs.

We now construct a network $\mathcal{N}$ for simulating a grammar in Kuroda normal form. Let $p_1, \ldots, p_k$ be the rules of the form $A \to BC$ with $A,B,C \in N$ ($k \geq 0$). Let $p_{k+1}, \ldots, p_m$ be the rules of the form $AB \to CD$ with $A,B,C,D \in N$ ($m \geq k$). Let $p_{m+1}, \ldots, p_q$ be the rules of the

form $A \to x$ with $A \in N$ and $x \in V$ ($q \geq m$). Let $p_{q+1}, \ldots, p_n$ be the rules of the form $A \to \lambda$ with $A \in N$ ($n \geq q$).

For each rule p_i with $1 \leq i \leq m$, we define two mappings l_i and r_i as follows:

$$\begin{aligned} l_i &: \{2\} \longrightarrow N, \text{ if } 1 \leq i \leq k, \\ l_i &: \{1,2\} \longrightarrow N, \text{ if } k+1 \leq i \leq m, \\ r_i &: \{1,2\} \longrightarrow N, \text{ if } 1 \leq i \leq m. \end{aligned}$$

If p_i is a rule $A \to BC$ then we set $l_i(2) = A$, $r_i(1) = B$, and $r_i(2) = C$. If p_i is a rule $AB \to CD$ then we set $l_i(1) = A$, $l_i(2) = B$, $r_i(1) = C$, and $r_i(2) = D$ (the values are the nonterminals of the left hand side and the right hand side at the respective position). As intermediate symbols, we introduce symbols $\langle i, j \rangle$ where i is the number of a rule ($1 \leq i \leq m$) and j is a position ($1 \leq j \leq 2$). We collect these symbols into two sets $\langle 1 \rangle$ and $\langle 2 \rangle$:

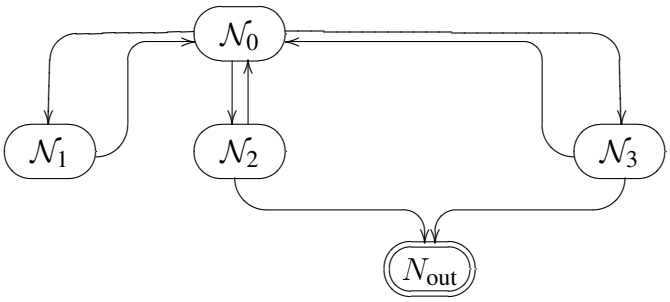
$$\langle 1 \rangle = \{ \langle i, 1 \rangle \mid 1 \leq i \leq m \}, \quad \langle 2 \rangle = \{ \langle i, 2 \rangle \mid 1 \leq i \leq m \}.$$

Further let $V' = \{ x' \mid x \in V \}$, $[V] = \{ [x] \mid x \in V \}$, and $\langle V \rangle = \{ \langle x \rangle \mid x \in V \}$ be mutually disjoint sets. We set

$$\hat{V} = V \cup V' \cup [V] \cup \langle V \rangle \cup \{ \mathsf{I}, \mathsf{I}' \} \cup \langle 1 \rangle \cup \langle 2 \rangle.$$

Let F be a symbol that does not occur in the set $\hat{V}$.

We define the network $\mathcal{N}$ over the alphabet $U = \hat{V} \cup \{ F \}$. The network has the following structure, where N_{out} denotes the output node:



The subnetwork $\mathcal{N}_0$ consists of the only node (the initial node) N_0 defined by

$$M_0 = \emptyset, \; A_0 = \{ S \}, \; I_0 = V^*, \; O_0 = V^*.$$

In the cycle consisting of $\mathcal{N}_0$ and $\mathcal{N}_1$, the simulation of the rules $p_1, p_2, \ldots, p_m$ (where the length of the right hand side is greater than one) is performed. In the cycle of $\mathcal{N}_0$ and $\mathcal{N}_2$, the application of the rules $p_{m+1}, p_{m+2}, \ldots, p_q$ is simulated. In the cycle of $\mathcal{N}_0$ and $\mathcal{N}_3$, the erasing rules of P are simulated (if P does not contain such rules, the subnetwork $\mathcal{N}_3$ is not needed).

The subnetwork $\mathcal{N}_2$ consists of the node N_2 defined by

$$\begin{aligned} M_2 &= \{ p_{m+1}, p_{m+2}, \ldots, p_q \}, \\ A_2 &= \emptyset, \; I_2 = V^*, \; O_2 = V^*. \end{aligned}$$

The subnetwork $\mathcal{N}_3$ consists of the node N_3 defined by

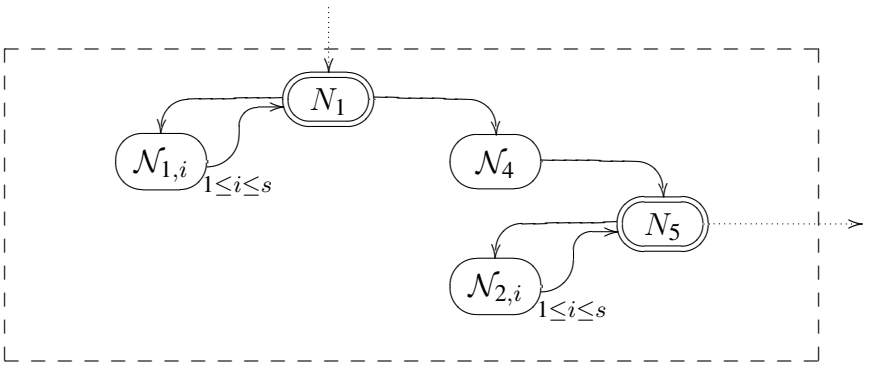
$$M_3 = \{ p_{q+1}, p_{q+2}, \ldots, p_n \},$$
$$A_3 = \emptyset,\ I_3 = V^*,\ O_3 = V^*.$$

The rules $p_{m+1}, \ldots, p_n$ may lead to a terminal word (in contrast to the rules $p_1, \ldots, p_m$). Therefore, terminal words can only be produced in the nodes N_2 and N_3. The words from these nodes are also sent to the output node N_{out}, which takes all incoming terminal words:

$$M_{\text{out}} = \emptyset,\ A_{\text{out}} = \emptyset,\ I_{\text{out}} = T^*,\ O_{\text{out}} = U^*.$$

All words that arrive in this node form the language that is generated by the network.

The subnetwork $\mathcal{N}_1$ has the form



where the node N_1 is defined by

$$M_1 = \{ x \to x' \mid x \in V \} \cup \{ l_i(2) \to \langle i, 2 \rangle \mid 1 \le i \le n \},$$
$$A_1 = \emptyset,\ I_1 = \hat{V}^*,\ O_1 = \hat{V}^*.$$

This node marks a symbol for removing from the end or for replacing according to a rule.

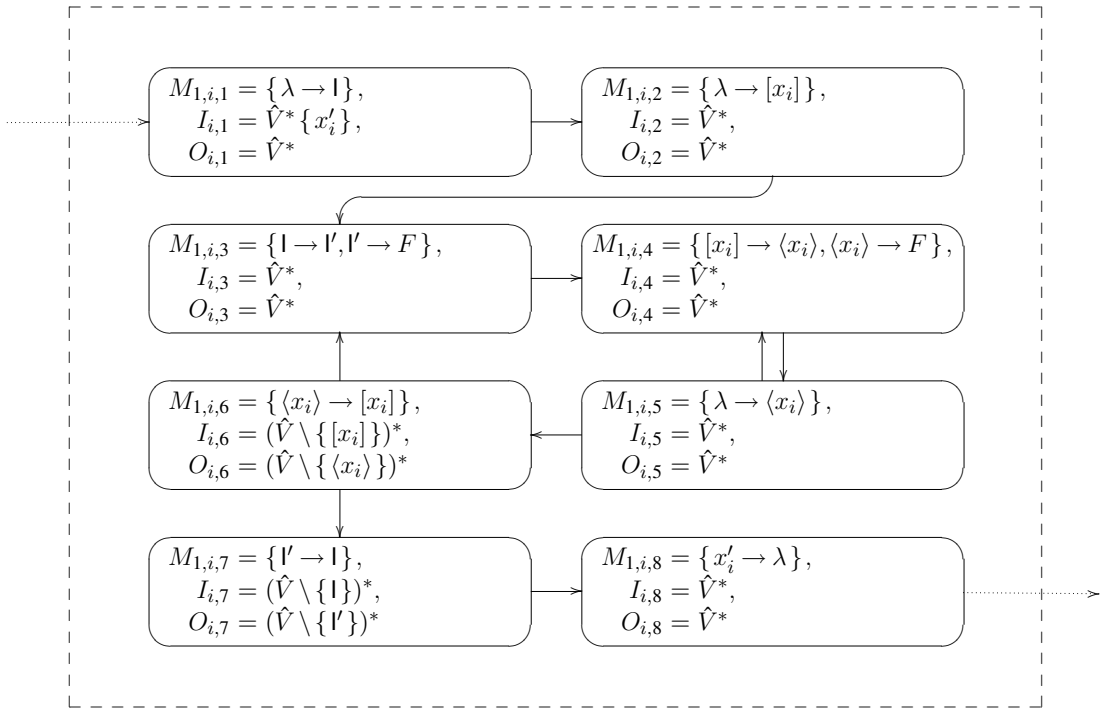
In each subnetwork $\mathcal{N}_{1,i}$ for $1 \le i \le s$, the elimination of the last symbol is performed if it is equal to x'_i. In the subnetwork $\mathcal{N}_4$, the application of a rule is simulated. The node N_5 is defined by

$$M_5 = \emptyset,\ A_5 = \emptyset,\ I_5 = \hat{V}^*,\ O_5 = \hat{V}^*.$$

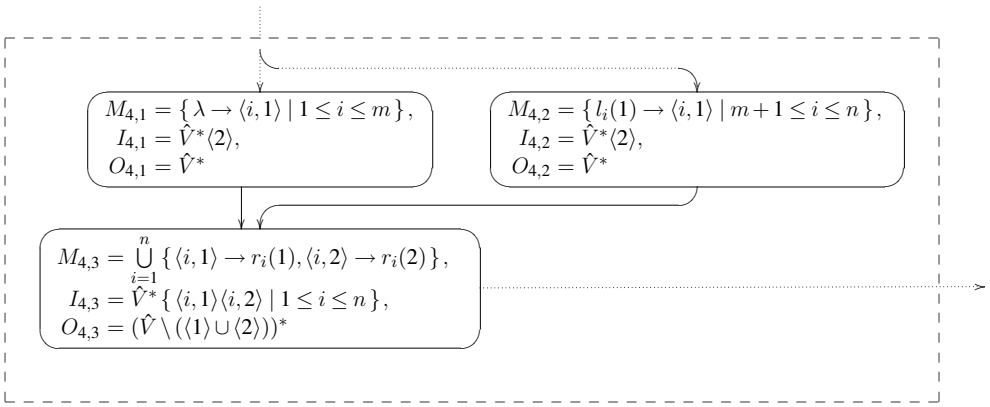
In each subnetwork $\mathcal{N}_{2,i}$ for $1 \le i \le s$, the letter x_i is restored at the end of the current word if it has been there originally.

For $1 \le i \le s$, the subnetwork $\mathcal{N}_{1,i}$ checks whether the last symbol of the word is the letter x'_i. If this is not the case, then the word is lost. Otherwise, the symbol 'I' is inserted which indicates the position of the last letter in the original suffix $a_t a_{t-1} \cdots a_1$ (the number of occurrences of the symbol 'I' is equal to the index of the last letter in the suffix). For example, let the current suffix be $a_t a_{t-1} \cdots a_{j+1} a'_j$. Let x_i be the letter a_j. Then the subnetwork $\mathcal{N}_{1,i}$ (and no other subnetwork $\mathcal{N}_{1,l}$) processes the word. After inserting the symbol 'I', the word contains exactly j occurrences of this symbol. Then the symbol $[x_i]$ (which is equal to $[a_j]$) is inserted 2^j times (one symbol is inserted and then the number of occurrences is doubled as many times as the symbol 'I' appears). Finally, the marked letter a'_j is deleted.

The network $\mathcal{N}_{1,i}$ has the following form ($1 \leq i \leq s$) – the initial set is empty in each node:



In the subnetwork $\mathcal{N}_4$, the application of a rule p_i with $1 \leq i \leq m$ is simulated. This subnetwork has the following form (the initial set is empty in each node):

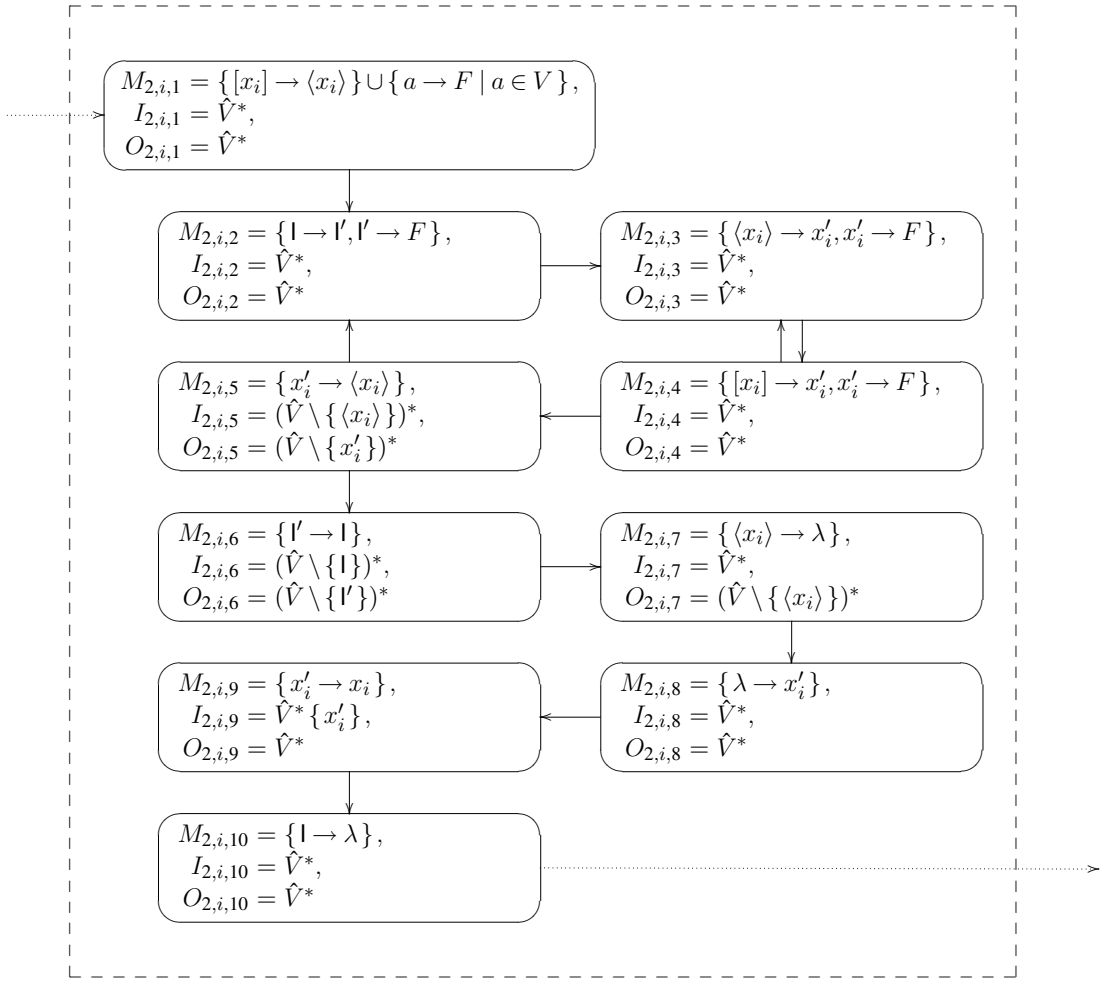


The nodes $N_{4,1}$ and $N_{4,2}$ take the word if its last symbol has been marked for the simulation of a rule by a symbol of the set $\langle 2\rangle$. Then a symbol of the set $\langle 1\rangle$ is produced (inserted, if the marking stands for a rule of the form $A \to BC$, or obtained by substitution, if the marking stands for a rule of the form $AB \to CD$). The third node $N_{4,3}$ checks whether at both places the same rule was chosen and whether the markings are in the correct order (whether the word ends with a subword $\langle i,1\rangle\langle i,2\rangle$ for some rule p_i with $1 \leq i \leq n$. If it is correct, the intermediate symbols are replaced by the respective symbols of the right hand side of the rule, otherwise the word is lost.

If the rule was simulated, the word is sent to node N_5 from where it is distributed to every subnetwork $\mathcal{N}_{2,i}$ for $1 \leq i \leq s$. Each subnetwork $\mathcal{N}_{2,i}$ checks whether the letter x_i has to be restored at the end of the word. Let j be the number of occurrences of the symbol 'I'. If the

symbol $[x_i]$ occurs 2^j times in the word, then $a_j = x_i$ and x_i is restored, otherwise, the word is lost in this network because, at some moment, the only applicable rule is a rule which introduces the 'fail' symbol F (but for every number j between t and 1, the condition $a_j = x_i$ is satisfied for some letter x_i and, hence, that subnetwork succeeds). When all letters of the suffix have been restored, the word sent from node N_5 to the node N_0 does not contain auxiliary symbols. Hence, it is taken by this node and the simulation of a rule p_i with $1 \leq i \leq n$ is finished.

The subnetwork $\mathcal{N}_{2,i}$ has the following form for $1 \leq i \leq s$ (the initial set is empty in each node):



The network $\mathcal{N}$ constructed above for an arbitrary phrase structure grammar has only filters that are definite languages. This proves the claim. □

Theorem 4.3 *$\mathcal{E}(COMB) = RE$.*

Proof. The network constructed in the proof of Theorem 4.2 with definite filters has – up to one exception – only combinational filters. The exception is the node $N_{4,3}$ where the input filter $\hat{V}^*\{\langle i,1\rangle\langle i,2\rangle \mid 1 \leq i \leq n\}$ is not combinational. We now replace the node $N_{4,3}$ as follows. For

each i, $1 \leq i \leq n$, we define the nodes

$$N_{4,3,i,1} = (\{\langle i,2\rangle \to \lambda\}, \emptyset, \hat{V}^*\{\langle i,2\rangle\}, \hat{V}^*\{\langle i,1\rangle\}),$$
$$N_{4,3,i,2} = (\{\lambda \to \langle i,2\rangle\}, \emptyset, \hat{V}^*\{\langle i,1\rangle\}, \hat{V}^*),$$
$$N_{4,3,i,3} = (\{\langle i,1\rangle \to r_i(1), \langle i,2\rangle \to r_i(2)\}, \emptyset, \hat{V}^*\{\langle i,2\rangle\}), (\hat{V} \setminus (\langle 1\rangle \cup \langle 2\rangle))^*).$$

Then we connect these nodes to the nodes as follows. The ingoing edges of $N_{4,3,i,1}$ are those from $N_{4,3}$; there are edges from $N_{4,3,i,1}$ to $N_{4,3,i,2}$ and from $N_{4,3,i,2}$ to $N_{4,3,i,3}$ and the outgoing edges of $N_{4,3,i,3}$ are the outgoing edges of $N_{4,3}$.

We note that all filters of the constructed network are combinational languages.

It is easy to see that a word passes through $N_{4,3}$ if and only if it passes in succession the nodes $N_{4,3,i,1}$, $N_{4,3,i,2}$, and $N_{4,3,i,3}$ for some i, $1 \leq i \leq n$ and conversely. Moreover, in both cases the word is changed in the same manner in both cases. Hence the constructed network generates the recursively enumerable languages from which we started in part i). Thus $RE \subseteq \mathcal{E}(COMB)$.

Because $\mathcal{E}(COMB) \subseteq \mathcal{E}(DEF)$ by the relations of Figure 1 and Lemma 2.2, the equality $RE = \mathcal{E}(COMB)$ follows. □

Theorem 4.4 $\mathcal{E}(UF) = RE$.

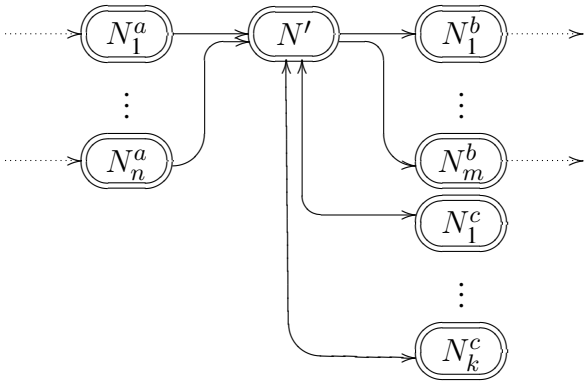
Proof. By Theorem 2.3, the relations of Figure 1, and Lemma 2.2, we have $\mathcal{E}(UF) \subseteq RE$.

Let $L \in RE$. By Theorem 4.3 we can assume that L is generated by an evolutionary network $\mathcal{N}$ with combinational filters and $L = L(\mathcal{N})$. Let U be the alphabet of the network. Furthermore, let N be a node of the network. Then N has the form

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

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

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



Every edge from a node K to the node N is replaced by edges from K to every node N_i^a for $1 \leq i \leq n$. Every edge from the node N to a node K is replaced by edges from every node N_i^b for $1 \leq i \leq m$ 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 $1 \leq i \leq m$; 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 $1 \leq i \leq m$ and N_i^c for $1 \leq i \leq k$ 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).

If we replace all nodes of the network $\mathcal{N}$ as described above, we obtain a network $\mathcal{N}'$ which also generates the language L. Moreover, if $V = \{ x_1, x_2, \ldots, x_p \}$, then

$$V^*\{a\} = (\{x_1\}^*\{x_2\}^* \cdots \{x_p\}^*)^*\{a\}.$$

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

By the relations shown Figure 1, Lemma 2.2, and Theorem 2.3, we obtain the following theorem.

Theorem 4.5 *$\mathcal{E}(ORD) = \mathcal{E}(NC) = \mathcal{E}(PS) = RE$.*

5. Computationally Non-Complete Cases

We first discuss the case of finite filters. We start with a certain normal form for networks with finite filters.

Lemma 5.1 *For each NEP $\mathcal{N}$ with only finite filters, we can construct a NEP $\mathcal{N}'$ with only one processor and finite filters that generates the same language as $\mathcal{N}$.*

Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a NEP with finite filters. Let B_j be the set of all words that enter some time the node N_j:

$$\begin{aligned} B_j &= \{ w \in V^* \mid \exists t \, \exists i : (i,j) \in E \text{ and } w \in C_{2t+1}(i) \cap O_i \cap I_j \} \\ &= \left\{ w \in V^* \mid \exists i : (i,j) \in E \text{ and } w \in O_i \cap I_j \cap \bigcup_{t \geq 0} C_{2t+1}(i) \right\}. \end{aligned}$$

The set B_j is finite ($B_j \subseteq I_j$) and can be computed since the set $L'(i) = \bigcup\limits_{t \geq 0} C_{2t+1}(i)$ is regular (see proof of Theorem 5.2, the equation also holds for $L'(i)$ instead of $L(\mathcal{N})$).

Let $\mathcal{N}' = (V, N_1', \emptyset, 1)$ be the NEP with the processor $N_1' = (M_j, A_j \cup B_j, \emptyset, O_j)$. Let C and C' be the configurations of N and N', respectively. We show inductively that every word w which is in node N_j at a time $2t'$ or $2t'+1$ (for a $t' \geq 0$) is also in node N_1' at a time $2t''$ or $2t''+1$ (for a $t'' \geq 0$), respectively, and vice versa.

- $w \in C_0(j)$. Then $w \in A_j$ and therefore $w \in C_0'(1)$.
- $w \in C_{2t'+1}(j)$ with $t' \geq 0$. Then

(a) $w \in C_{2t'}(j)$ and M_j is not applicable or
(b) there is a word $v \in C_{2t'}(j)$ which yields w ($v \Longrightarrow_{M_j} w$).

In Case (a), we have $w \in C'_{2t''}(1)$ for a $t'' \geq 0$ by induction hypothesis. Since no rule is applicable, we also have $w \in C'_{2t''+1}(1)$. In Case (b), we have $v \in C'_{2t''}(1)$ for a $t'' \geq 0$ by induction hypothesis. Since $v \Longrightarrow_{M_j} w$ it is $w \in C'_{2t''+1}(1)$.

- $w \in C_{2t'}(j)$ with $t' \geq 1$. Then

  (a) there is k with $1 \leq k \leq n$, $(k,j) \in E$, and $w \in C_{2t'-1}(k) \cap O_k \cap I_j$ or
  (b) $w \in C_{2t'-1}(j) \setminus O_j$.

  In Case (a), we have $w \in B_j$ and therefore $w \in C'_0(1)$. In Case (b), we have $w \in C'_{2t''+1}(1)$ for a $t'' \geq 0$ by induction hypothesis. Since $w \notin O_j$ we also have $w \in C'_{2t''+2}(1)$.

- $w \in C'_0(1)$. Then

  (a) $w \in A_j$ or
  (b) $w \in B_j$.

  In Case (a), we also have $w \in C_0(j)$. In Case (b), we have $w \in C_{2t'}(j)$ for a $t' \geq 1$.

- $w \in C'_{2t''+1}(1)$ with $t'' \geq 0$. Then

  (a) $w \in C'_{2t''}(1)$ and M_j is not applicable or
  (b) there is a word $v \in C'_{2t''}(1)$ which yields w ($v \Longrightarrow_{M_j} w$).

  In Case (a), we have $w \in C_{2t'}(j)$ for a $t' \geq 0$ by induction hypothesis. Since no rule is applicable, we also have $w \in C_{2t'+1}(j)$. In Case (b), we have $v \in C_{2t'}(j)$ for a $t' \geq 0$ by induction hypothesis. Since $v \Longrightarrow_{M_j} w$ it is $w \in C_{2t'+1}(1)$.

- $w \in C'_{2t''}(1)$ with $t'' \geq 1$. Then $w \in C'_{2t''-1}(1) \setminus O_j$ (because there is no 'real' communication). We have $w \in C_{2t'+1}(j)$ for a $t' \geq 0$ by induction hypothesis. Since $w \notin O_j$ we also have $w \in C_{2t'+2}(j)$.

By this induction, it is shown that $L(\mathcal{N}') = L(\mathcal{N})$. □

Theorem 5.2 *$\mathcal{E}(FIN) \subset REG$.*

Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a network with finite filters. Obviously, a word w is in N_j if and only if it is in A_j or satisfies I_j or is obtained from a word in N_j by application of a rule in M_j. We set

$$U = \{\, a \mid \lambda \to a \in M_j \,\},\ V' = \{\, a' \mid a \in V \,\},\ \text{and}\ U' = \{\, a' \mid a \in U \,\}.$$

Let $h : (V \cup V')^* \to V^*$ be the homomorphism defined by

$$h(a) = a \text{ for } a \in V \quad \text{and} \quad h(a') = \begin{cases} \lambda, & \text{for } a' \in U', \\ a, & \text{for } a' \in V' \setminus U', \end{cases}$$

and $\tau : (V \cup V')^* \to V^*$ be the finite substitution where $\tau(a) = \tau(a')$ for $a \in V$ and $\tau(a)$ consists of all $b \in V \cup \{\lambda\}$ such that there are an integers $s \geq 0$ and $b_0, b_1, \ldots, b_{s-1} \in V$ and $b_s \in V \cup \{\lambda\}$

such that $a = b_0$, $b = b_s$, and $b_i \to b_{i+1} \in M_j$ for $0 \leq i \leq s-1$ (note that $s = 0$ implies $a = b$). Furthermore, let

$$k = \max\{\, |w| \mid w \in O_j \cup I_j \cup A_j \,\} + 1.$$

We note the following facts:

- Assume that there is a word w of length at least k in $L(\mathcal{N})$. Then w is in $C_t(j)$ for some t. By its length, it cannot leave the node, and thus all words which have a length at least k and can be obtained by application of rules of M_j to w belong to $L(\mathcal{N})$, too.
- If w with $|w| \geq k+1$ is in $L(\mathcal{N})$, then w is obtained from a word $v \in L(\mathcal{N})$ of length k by application of rules in M_j (since substitutions and deletions do not increase the length, the shortest words in $L(\mathcal{N})$ with length at least k are obtained by an insertion from a word of length less than k and thus they have length k).

Now it is easy to see that

$$L(\mathcal{N}) = (L(\mathcal{N}) \cap \bigcup_{i=0}^{k-1} V^i) \cup (\tau(h^{-1}(L(\mathcal{N}) \cap V^k)) \cap \bigcup_{i \geq k} V^i)$$

holds. Since finite languages are regular and regular languages are closed under inverse homomorphisms, finite substitutions, intersection, and union, $L(\mathcal{N})$ is regular. Hence $\mathcal{E}(\mathit{FIN}) \subseteq \mathit{REG}$ holds.

Let $V = \{a\}$ and $L = \{a\} \cup \{\, a^n \mid n \geq 3 \,\}$. Obviously, L is regular.

Suppose the language L is generated by a network with only finite filters. By Lemma 5.1, there is then a network $\mathcal{N}$ with only one node $N = (M, A, \emptyset, O)$ that generates L. Since L is infinite, this node must be inserting. Hence, the rule set is $M = \{\, \lambda \to a \,\}$. If the initial set A contains λ then $\lambda \in L(\mathcal{N})$ which is in contrast to $\lambda \notin L$. If the initial set A contains a or aa then the word aa belongs to the generated language $L(\mathcal{N})$ which is in contrast to $aa \notin L$. If the initial set only contains words a^n with $n \geq 3$ then the word a cannot be generated but $a \in L$ which is a contradiction, too. Hence, there is no network with only finite filters that generates the language L. Thus, $L \in \mathit{REG} \setminus \mathcal{E}(\mathit{FIN})$. □

The following results show that the use of filters from the remaining language families, i. e., from *MON* or *NIL* or *COMM* leads to the same class of languages.

Theorem 5.3 $\mathcal{E}(\mathit{MON}) = \mathcal{E}(\mathit{COMM})$.

Proof. According to Lemma 2.2, we have the inclusion $\mathcal{E}(\mathit{MON}) \subseteq \mathcal{E}(\mathit{COMM})$. We now show the converse inclusion $\mathcal{E}(\mathit{COMM}) \subseteq \mathcal{E}(\mathit{MON})$.

Let $V = \{\, x_1, x_2, \ldots, x_r \,\}$ be an alphabet. Let $\mathcal{N}$ be a network of evolutionary processors with the working alphabet V and where all filters are commutative regular languages. We assume that $\mathcal{N}$ has the property that all output filters are monoidal languages and that the nodes with non-monoidal input filters have no evolution rules and no initial words (according to the considerations in the beginning of this section).

We now show how a node $N = (\emptyset, \emptyset, I, O)$ of the network $\mathcal{N}$ with an arbitrary commutative regular input filter (and a monoidal output filter) can be simulated by a network where all filters are monoidal.

For the alphabet V, we define four sets $\tilde{V}$, V', $\tilde{V}'$, and $\hat{V}$:

$$\begin{aligned}\tilde{V} &= \{\, \tilde{x} \mid x \in V \,\},\\ V' &= \{\, x' \mid x \in V \,\},\\ \tilde{V}' &= \{\, \tilde{x}' \mid x \in V \,\}, \text{ and}\\ \hat{V} &= V \cup \tilde{V} \cup V' \cup \tilde{V}'.\end{aligned}$$

Furthermore, let $h : V \longrightarrow \tilde{V}$ be the isomorphism with $h(x) = \tilde{x}$ for $x \in V$. By $\tilde{L}$, we denote the language $h(L)$.

Let $G = (N, \tilde{V}, P, S)$ be a regular grammar that generates the language $\tilde{I}$. We now create a network that simulates G_i. We assume that all rules in P have the form $A \to aB$ or $A \to a$ for nonterminals $A, B \in N$ and a terminal symbol $a \in \tilde{V}$. Additionally, the rule $S \to \lambda$ is permitted if the axiom S does not occur on the right hand side of a rule.

For each nonterminal $X \in N$, we set

$$R(X) = \{\, \langle aY \rangle \mid X \to aY \in P \,\}$$

as the set of symbols representing the right hand sides of the nonterminating rules,

$$T_t(X) = \{\, a \mid a \in \tilde{V} \text{ and } X \to a \in P \,\}$$

as the set of all terminal symbols that are generated by X with terminating,

$$N_{post}(X) = \{\, Y \mid Y \in N \text{ and } \exists a \in \tilde{V} : X \to aY \in P \,\}$$

as the set of all nonterminals that are generated by X, and

$$T_{pre}(X) = \{\, a \mid a \in \tilde{V} \text{ and } \exists Y \in N : Y \to aX \in P \,\}$$

as the set of all terminal symbols that are generated at the same time as X. The terminating nonterminals (those nonterminals $X \in N$ for which a rule $X \to a \in P$ exists for a terminal symbol $a \in \tilde{V}$) are gathered in the set N_t. Furthermore, we set $R = \{\, \langle aY \rangle \mid \exists X \in N : X \to aY \in P \,\}$.

For every nonterminal $X \in N$, we define two nodes

$$N_X = (M_X, A_X, I_X, O_X) \quad \text{and} \quad N_{X'} = (M_{X'}, A_{X'}, I_{X'}, O_{X'})$$

as follows:

$$\begin{aligned} M_X &= \{\, \lambda \to \langle aY \rangle \mid X \to aY \in P \,\}, & M_{X'} &= \{\, \langle aX \rangle \to a \mid a \in T_{pre}(X) \,\},\\ A_X &= \emptyset, & A_{X'} &= \emptyset,\\ I_X &= (V \cup \tilde{V})^*, & I_{X'} &= (\{\, \langle aX \rangle \mid a \in T_{pre}(X) \,\} \cup V \cup \tilde{V})^*,\\ O_X &= (R \cup V \cup \tilde{V})^*, & O_{X'} &= (V \cup \tilde{V})^*. \end{aligned}$$

We put an edge from N_X to $N_{Y'}$ if $Y \in N_{post}(X)$ and an edge from $N_{X'}$ to N_X for each nonterminal $X \in N$. In these nodes, the application of rules of the form $X \to aY$ is simulated. First, the word is in node N_X, then a is inserted somewhere in the word (by $N_{Y'}$) and then the word is in node N_Y.

For each terminating nonterminal $X \in N_t$, we additionally define a node

$$N_{X_t} = (M_{X_t}, A_{X_t}, I_{X_t}, O_{X_t})$$

by the sets $M_{X_t} = \{ \lambda \to a \mid a \in T_t(X) \}$, $A_{X_t} = \emptyset$, and $I_{X_t} = O_{X_t} = (V \cup \tilde{V})^*$ and connect the node $N_{X'}$ to this node.

In these nodes, the simulation of the derivation ends. The resulting words move on to a chain of nodes where one copy of each letter of V and $\tilde{V}$ is inserted. Let these nodes be

$$N_{x_i} = (\{ \lambda \to x_i \}, \emptyset, (V \cup \tilde{V})^*, (V \cup \tilde{V})^*) \text{ and } N_{\tilde{x}_i} = (\{ \lambda \to \tilde{x}_i \}, \emptyset, (V \cup \tilde{V})^*, (V \cup \tilde{V})^*)$$

for $1 \leq i \leq r$. We connect all nodes N_{X_t} for $X \in N_t$ to N_{x_1}, every node N_{x_i} to $N_{\tilde{x}_i}$ for $1 \leq i \leq r$, and every node $N_{\tilde{x}_i}$ to $N_{x_{i+1}}$ for $1 \leq i \leq r-1$. This ensures that every letter of $V \cup \tilde{V}$ occurs at least once in a word (we need this for technical reasons in the sequel). The words obtained move then to another chain of nodes where it is checked whether the original word (over V – scattered over the whole word) is letter equivalent upto $\tilde{}$ (upto the isomorphism h) to the word generated by the grammar G_i (over $\tilde{V}$ – also scattered over the whole word). Let these nodes be

$$N_{x'_i} = (\{ x_i \to x'_i,\ x'_i \to F \}, \emptyset, I_{x'_i}, \hat{V}^*)$$

with

$$I_{x'_i} = (\bigcup_{k=1}^{j-1} \{ x'_k, \tilde{x}'_k \} \cup \bigcup_{k=j}^{r} \{ x_k, \tilde{x}_k, x'_k, \tilde{x}'_k \})^*$$

and

$$N_{\tilde{x}'_i} = (\{ \tilde{x}_i \to \tilde{x}'_i,\ \tilde{x}'_i \to F \}, \emptyset, \hat{V}^*, \hat{V}^*)$$

for $1 \leq j \leq r$. The symbol F is a new symbol that cannot be derived. If this symbol is introduced then the word is kept inside the node forever.

In the end, we delete the symbols of $\tilde{V}'$ from the word in node

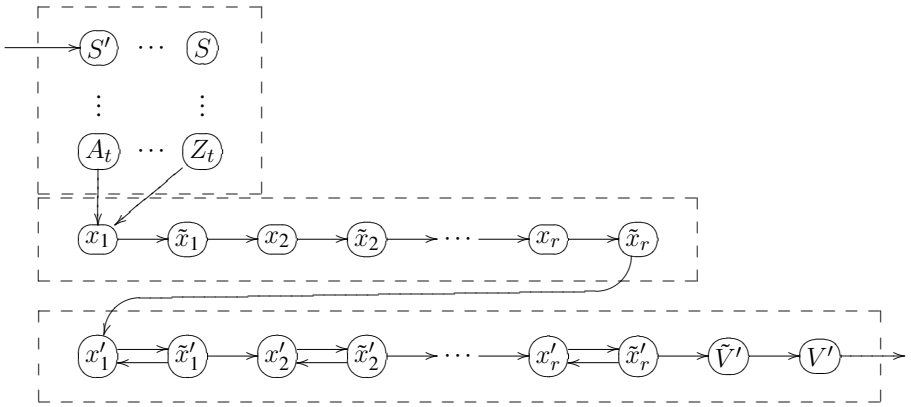
$$N_{\tilde{V}'} = (\{ \tilde{x}'_i \to \lambda \mid 1 \leq j \leq r \}, \emptyset, (V' \cup \tilde{V}')^*, (V')^*)$$

and replace the primed symbols by their unprimed counterparts in node

$$N_{V'} = (\{ x'_i \to x_i \mid 1 \leq j \leq r \}, \emptyset, (V')^*, V^*).$$

We connect $N_{\tilde{x}_r}$ to $N_{x'_1}$, every node $N_{x'_i}$ to $N_{\tilde{x}'_i}$ and $N_{\tilde{x}'_i}$ to $N_{x'_i}$ for $1 \leq j \leq r$, and every node $N_{\tilde{x}'_i}$ to $N_{x'_{i+1}}$ for $1 \leq j \leq r-1$ as well as $N_{\tilde{x}'_r}$ to $N_{\tilde{V}'}$ and $N_{\tilde{V}'}$ to $N_{V'}$. In this chain, first the letters x_1 and $\tilde{x}_1$ are marked one by one. If the numbers are equal then the word can move to node $N_{x'_2}$ were the marking of the letters x_2 and $\tilde{x}_2$ begins. If the numbers of x_i and $\tilde{x}_i$ are different for some j then the word cannot move on because an F is introduced. If for $1 \leq j \leq r$ the numbers of x_i and $\tilde{x}_i$ coincide then the word finally arrives in node $N_{\tilde{V}'}$ where the symbols $\tilde{x}'_i$ are removed and it continues to $N_{V'}$ where the original word is restored. Hence, a word w passes the node N_i (the input filter I_i and output filter O_i anyway) if and only if there is a computation in the network between the nodes $N_{S'}$ and $N_{V'}$ described above such that the word w finally leaves the node $N_{V'}$.

The network described above is illustrated in the following picture.



The filters are all monoidal. The entrance node of the network is $N_{S'}$. Hence, the edges that lead to N_i now lead to $N_{S'}$. The exit node of the network is $N_{V'}$. Hence, the edges that leave N_i now leave the node $N_{V'}$.

If this construction is repeated for every node with a non-monoidal input filter, one obtains a network that generates the same language as $\mathcal{N}$ and which has only monoidal filters. Hence, the inclusion $\mathcal{E}(\mathit{COMM}) \subseteq \mathcal{E}(\mathit{MON})$ follows which yields with $\mathcal{E}(\mathit{MON}) \subseteq \mathcal{E}(\mathit{COMM})$ the equality. □

Theorem 5.4 $\mathcal{E}(\mathit{MON}) = \mathcal{E}(\mathit{NIL})$.

Proof. By Lemma 2.2, $\mathcal{E}(\mathit{MON}) \subseteq \mathcal{E}(\mathit{NIL})$.

In order to prove the inverse inclusion, we first show that any language of $\mathcal{E}(\mathit{NIL})$ can be generated by an evolutionary network $\mathcal{N}$ where all filters are finite languages or monoidal languages.

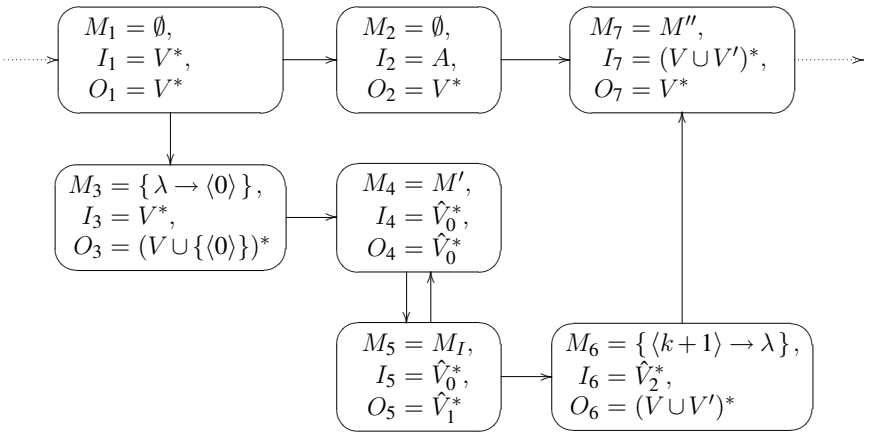
Let $\mathcal{N}$ be a NEP with a working alphabet V where all filters are nilpotent. The complement of a nilpotent language is also a nilpotent language. According to the considerations in the beginning of this section, we can assume that $\mathcal{N}$ has the property that all output filters are V^* and that the nodes with non-monoidal input filters have no evolution rules and no initial words.

We show how to simulate a node with an arbitrary nilpotent input filter by a network with finite or monoidal filters only. Let $N = (\emptyset, \emptyset, I, O)$ be such a node. If I is finite or monoidal then the filter has already a desired form. Let I be an infinite, non-monoidal, nilpotent language. Then I can be expressed as $I = V^* V^{k+1} \cup A$ with $A \subset V_k$ for some natural number $k \geq 0$.

Let F be a symbol not in V and let

$$
\begin{aligned}
V' &= \{\, a' \mid a \in V \,\}, \\
\hat{V} &= V \cup V', \\
\hat{V}_0 &= \hat{V} \cup \{\, \langle i \rangle \mid 0 \leq i \leq k \,\}, \\
\hat{V}_1 &= \hat{V} \cup \{\, \langle i \rangle \mid 1 \leq i \leq k+1 \,\}, \\
\hat{V}_2 &= \hat{V} \cup \{\, \langle k+1 \rangle \,\}, \\
M' &= \{\, a \to a' \,\}\, a \in V \cup \{\, \langle i \rangle \to F \mid 0 \leq i \leq k \,\}, \\
M'' &= \{\, a' \to a \,\}\, a \in V, \\
M_s &= \{\, \langle i \rangle \to \langle i+1 \rangle \mid 0 \leq i \leq k \,\}.
\end{aligned}
$$

We construct the following network simulating the node N (all sets A_i are empty):



Let w be a word of I. Then w belongs to A or it contains at least $k+1$ letters. If it belongs to A, the word can pass the network via node N_2. If it contains at least $k+1$ letters, the word can take the other path through the network. In node N_3, the symbol $\langle 0 \rangle$ is inserted. In the cycle of the nodes N_4 and N_5, a letter a is marked as a' and the symbol $\langle i \rangle$ is increased alternatingly until $k+1$ letters are primed. Then the word moves to node N_6 where the symbol $\langle k+1 \rangle$ is removed. It moves on to node N_7 where the primed symbols are unmarked again to obtain the word w.

If in node N_4 a rule $\langle i \rangle \to F$ is applied then the word cannot leave the node anymore. If a word w does not belong to I, then it does not contain $k+1$ letters. The word enters the node N_4 before $\langle k+1 \rangle$ has reached and all letters are primed. Then the trap symbol F is introduced and no word derived from w can leave the network (note, it does not pass node N_2 either).

Hence, the network simulates the node N.

We now replace the nodes with finite input/output filter by nodes with monoidal filters and change the graph in an appropriate way. (We note that this construction is not algorithmic since we do not determine the filters; we only known that they can be chosen in that form.)

First let $N = (M, A, I, O)$ be a node with a finite input filter $I \subset V^*$. Since I is finite, only a finite set $I' \subset I$ can enter the node N during the computations. Therefore only the words of the set $A \cup I' \cup M(A) \cup M(I')$ occur in the node N. Thus we replace N by the node $N' = (\emptyset, A \cup I' \cup M(A) \cup M(I'), V^*, O)$ and cancel all ingoing edges. Obviously, this construction does not change the generated language. Moreover, all input filters of the obtained network are monoidal.

Now let $\overline{N} = (\overline{M}, \overline{A}, \overline{I}, \overline{O})$ be a node with a finite output filter $\overline{O} \subset \overline{V}^*$. Then the set of words which leave $\overline{N}$ during the computations is a finite subset $\overline{O}'$ of $\overline{O}$. If $\overline{N}$ is not the output node, we replace $\overline{N}' = (\emptyset, \overline{O}', I, \overline{V}^*)$ and cancel all ingoing edges. Again, we obtain an evolutionary network generating the same language as the given network. If $\overline{N}$ is the output node, we replace the node $\overline{N}$ by $\overline{N}'$ as above and add a node $\overline{N}'' = (M, A, I, \overline{V}^*)$ which has no outgoing edges and there is an edge from a node Z to $\overline{N}''$ if and only there is an edge from Z to $\overline{N}$. Again it is easy to see that this construction does not change the generated language. Now, also all output filters are monoidal languages.

Thus the language $L(\mathcal{N})$ belongs to $\mathcal{E}(MON)$. □

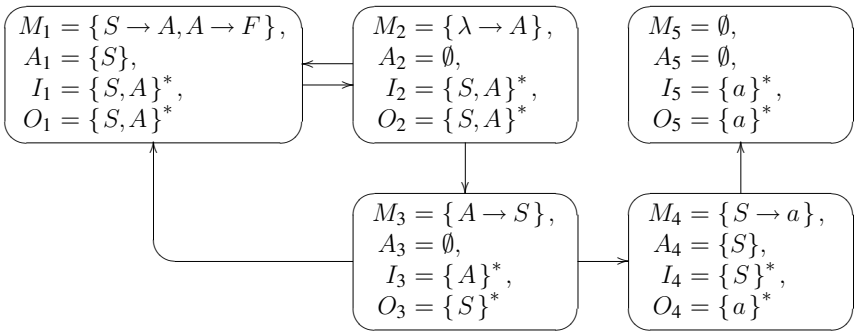
We now present some relations of $\mathcal{E}(MON)$ to other language families.

Theorem 5.5 $\mathcal{E}(FIN) \subset \mathcal{E}(MON)$.

Proof. Since $FIN \subset NIL$, we obtain $\mathcal{E}(FIN) \subseteq \mathcal{E}(NIL)$ by Lemma 2.2. By Theorem 3.2, the nilpotent language $L = \{a\} \cup \{ a^n \mid n \geq 3 \}$ is contained in $\mathcal{E}(NIL)$. However, by the second part of the proof of Theorem 5.2, L is not contained in $\mathcal{E}(FIN)$. Thus $\mathcal{E}(FIN) \subset \mathcal{E}(NIL)$. The statement now follows from Theorem 5.4. □

Lemma 5.6 *The family $\mathcal{E}(MON)$ contains a non-semi-linear (hence non-regular and non-context-free) language.*

Proof. Let $V = \{ S, A, F, a \}$ and $\mathcal{N} = (V, N_1, N_2, N_3, N_4, N_5, E, 5)$ be the following network:



In the beginning, we have the word S in node N_1. We consider a word S^n for $n \geq 1$ in node N_1 in an even moment (in the beginning or after a communication step). One occurrence of S is replaced by A, then the word is sent to node N_2 where another copy of A is inserted. This word w goes back to node N_1 and it goes on to node N_3 which takes it if no S appears in the word. If in N_1 the rule $A \to F$ is applied then the symbol F is introduced which cannot be replaced. Due to the output filter O_1, the word will be trapped in N_1 for ever. If, in the word w, no S is present, then the only rule which can be applied is $A \to F$ and the cycle is stopped. If w still contains an S, then it is replaced by A and N_2 inserts another A. So, the words move between N_1 and N_2 where alternatingly an S is replaced by A and an A is inserted until the word only contains As. The word is then A^{n+1}. Hence, the number of letters has been doubled.

In N_3, each A is replaced by S. The word is S^{n+1} when it leaves N_3. It moves to N_1 and to N_4. In N_1, the cycle starts again with a word S^m for $m \geq 1$. All arriving words in N_4 have the form S^n with $n \geq 2$. In order to cover also the case $n = 1$, the initial language of this node consists of S. In N_4, every letter S is replaced by the symbol a before the word leaves to node and moves to the output node N_5.

Hence, $L(\mathcal{N}) = \left\{ a^{2^n} \mid n \geq 0 \right\}$. □

Corollary 5.7 *$NIL \subset \mathcal{E}(MON)$ and $COMM \subset \mathcal{E}(MON)$.*

Proof. The inclusions follow from Corollary 3.3 and Theorems 5.3 and 5.4. The strictness follows from Lemma 5.6. □

Finally, we give a result which can be understood as a lower bound for the generative power of monoidal filters.

Theorem 5.8 *Let L be a semi-linear language. Then $Comm(L) \in \mathcal{E}(MON)$.* □

Proof. For each semi-linear language L, a regular grammar G can be constructed which generates a language that is letter-equivalent to L, i. e., $\psi(L(G)) = \psi(L)$ ([19]). Then we have

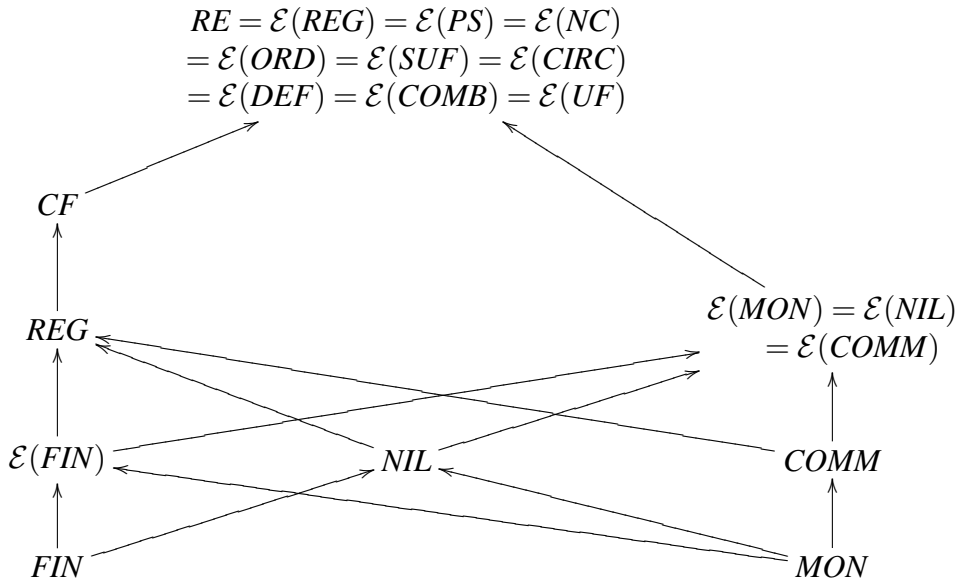
$$\psi^{-1}(\psi(L(G))) = \psi^{-1}(\psi(L))$$

and therefore $Comm(L(G)) = Comm(L)$. Thus, for any semi-linear language L, a regular grammar G can be constructed with $Comm(L(G)) = Comm(L)$. For a regular grammar G, a network with only monoidal filters that generates the language $Comm(L(G))$ can be constructed analogously to the construction in the proof of Theorem 5.3. □

6. Conclusion

If we combine all the results of the preceding sections, we get the following diagram which we state as a theorem.

Theorem 6.1 *The following diagram holds.*



The subregular classes considered in this paper are defined by combinatorial or algebraic properties of the languages. In [11], subclasses of *REG* defined by descriptional complexity have been considered. Let REG_n be the set of regular languages which can be accepted by deterministic finite automata. Then we have

$$REG_1 \subset REG_2 \subset REG_3 \subset \cdots \subset REG_n \subset \cdots \subset REG.$$

By Lemma 5.6 and [11], Lemma 4.1 and Theorems 4.3, 4.4., and 4.5, we get

$$\mathcal{E}(REG_1) \subset \mathcal{E}(MON) \subset \mathcal{E}(REG_2) = \mathcal{E}(REG_3) = \cdots = RE$$

and the incomparability of $\mathcal{E}(REG_1)$ with *REG* and *CF*.

References

[1] A. ALHAZOV, J. DASSOW, C. MARTÍN-VIDE, Y. ROGOZHIN, B. TRUTHE, On Networks of Evolutionary Processors with Nodes of Two Types. *Fundamenta Informaticae* **91** (2009), 1–15.

[2] H. BORDIHN, M. HOLZER, M. KUTRIB, Determination of Finite automata accepting subregular languages. *Theoretical Computer Science* **410** (2009), 3209–3222.

[3] J. BRZOZOWSKI, G. JIRÁSKOVÁ, B. LI, Quotient complexity of ideal languages. In: *LATIN 2010*. LNCS 6034, Springer-Verlag Berlin, 2010, 208–221.

[4] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, J. M. SEMPERE, Solving NP-Complete Problems With Networks of Evolutionary Processors. In: *IWANN '01: Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks, LNCS 2084*. Springer-Verlag Berlin, 2001, 621–628.

[5] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, J. M. SEMPERE, Networks of Evolutionary Processors. *Acta Informatica* **39** (2003) 6–7, 517–529.

[6] E. CSUHAJ-VARJÚ, A. SALOMAA, Networks of Parallel Language Processors. In: *New Trends in Formal Languages – Control, Cooperation, and Combinatorics*. LNCS 1218, Springer-Verlag Berlin, 1997, 299–318.

[7] J. DASSOW, Subregularly controlled derivations: the context-free case. *Rostocker Mathematisches Kolloquium* **34** (1988), 61–70.

[8] J. DASSOW, Grammars with commutative, circular, and locally testable conditions. In: *Automata, Formal Languages, and Related Topics – Dedicated to Ferenc Gécseg on the occasion of his 70th birthday*. University of Szeged, 2009, 27–37.

[9] J. DASSOW, H. HORNIG, Conditional grammars with subregular conditions. In: *Proc. Internat. Conf. Words, Languages and Combinatorics II*. World Scientific Singapore, 1994, 71–86.

[10] J. DASSOW, R. STIEBE, B. TRUTHE, Generative capacity of subregularly tree controlled grammars. *International Journal of Foundations of Computer Science* **21** (2010), 723–740.

[11] J. DASSOW, B. TRUTHE, On networks of evolutionary processors with filters accepted by two-state-automata. *Fundamenta Informaticae* (2011). To appear.

[12] Y.-S. HAN, K. SALOMAA, State complexity of basic operations on suffix-free regular languages. *Theoretical Computer Science* **410** (2009) 27–29, 2537–2548.

[13] Y.-S. HAN, K. SALOMAA, D. WOOD, Nondeterministic state complexity of basic operations for prefix-suffix-free regular languages. *Fundamenta Informaticae* **90** (2009) 1–2, 93–106.

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

[15] M. HOLZER, S. JAKOBI, M. KUTRIB, The magic number problem for subregular language families. In: *Proceedings of 12th Internat. Workshop Descriptional Complexity of Formal Systems*. University of Saskatchewan, Saskatoon, 2010, 135–146.

[16] G. JIRÁSKOVÁ, T. MASOPUST, Complexity in union-free languages. In: *Developments in Language Theory*. LNCS 6224, Springer-Verlag Berlin, 2010, 255–266.

[17] C. MARTÍN-VIDE, V. MITRANA, Networks of Evolutionary Processors: Results and Perspectives. In: *Molecular Computational Models: Unconventional Approaches*. 2005, 78–114.

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

[19] A. SALOMAA, *Formal Languages*. Springer-Verlag, Berlin, 1978.

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