Acta Informatica (2013) 50:41–75
DOI 10.1007/s00236-012-0172-0

ORIGINAL ARTICLE

Networks of evolutionary processors: the power of subregular filters

Jürgen Dassow · Florin Manea · Bianca Truthe

Received: 1 February 2012 / Accepted: 18 October 2012 / Published online: 21 November 2012

Abstract In this paper, we propose a hierarchy of families of languages generated by networks of evolutionary processors where the filters belong to 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 every regular language can be generated. If we use filters that are monoids, nilpotent languages, or commutative regular languages, we obtain one and 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 families of languages that one can use as filters in a network of evolutionary processors in order to obtain a complete computational model.

Keywords Networks of evolutionary processors · Subregular languages · Generative power

Mathematics Subject Classification (2000) 68Q05 · 68Q42 · 68Q45 · 68Q85

This paper is an extended version of the contribution presented at the 5th International Conference on Language and Automata Theory and Applications (LATA), Tarragona (Spain), May 2011, LNCS 6838, 262–273.

J. Dassow (✉) · B. Truthe
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg,
PSF 4120, 39016 Magdeburg, Germany
e-mail: dassow@iws.cs.uni-magdeburg.de

B. Truthe
e-mail: truthe@iws.cs.uni-magdeburg.de

F. Manea
Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
e-mail: flm@informatik.uni-kiel.de

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 non-deterministic 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 number α, $n \leq \alpha \leq 2^n$, is called magic (w.r.t. n), if there is no non-deterministic 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 (non-deterministic) 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 [3,12,13], 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], 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 networks of evolutionary processors as language generators.

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 family 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 families 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.

Let V be an alphabet. By V^* we denote the set of all words (strings) over the alphabet V (including the empty word λ). The length of a word w is denoted by $|w|$. By V^+ we denote the set of all non-empty words. For a natural number k, we denote by V^k the set of all words over the alphabet V with length k. We write V_k for the set of all words over V with a length of at most k:

$$V_k = \bigcup_{i=0}^{k} V^i.$$

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

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

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

– *combinational* iff it has 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

$$L = Comm(L),$$

– *circular* iff

$$L = Circ(L),$$

– *suffix-closed* (or *fully initial* or *multiple-entry* language) if the relation $xy \in L$ for some $x, y \in V^*$ implies $y \in L$ or equivalently,

$$L = Suf(L),$$

– *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. In this paper, we consider among the commutative, circular, suffix-closed, non-counting, and power-separating languages only those that are also regular. So we do not necessarily mention the regularity then.

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 Fig. 1.

In [11], networks were studied where all the filters belong to a set of languages that are accepted by deterministic finite automata with a fixed number of states. By REG_i for $i \geq 1$, we denote the family of languages that are accepted by deterministic finite automata with exactly i states.

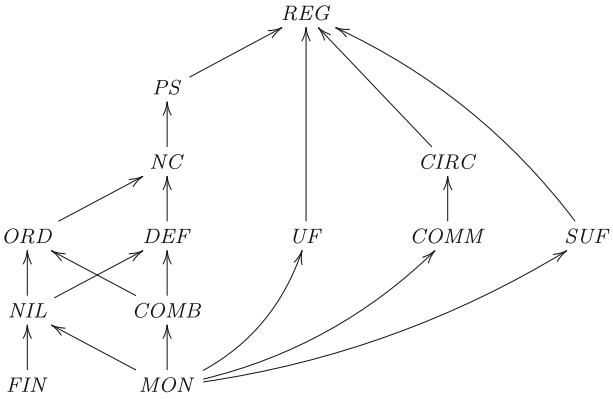


Fig. 1 Hierarchy of subregular languages (an *arrow* from X to Y denotes $X \subset Y$ and if two families are not connected by a directed path then they are incomparable)

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 $w = x\beta y$. In order to indicate also the applied rule p, 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 1 Let X be a family of regular languages.

i) A network of evolutionary processors (of size n) with filters from the family 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 if $M_i \subseteq \{a \to b \mid a, b \in V\}$ (by any rule, a letter is substituted by another one),
- a deletion node if $M_i \subseteq \{a \to \lambda \mid a \in V\}$ (by any rule, a letter is deleted), or
- an insertion node if $M_i \subseteq \{\lambda \to b \mid b \in V\}$ (by any rule, a letter is inserted).

Every node has rules from one type only. 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 by applying rules from the set M_i. In a communication step, any processor N_i sends out all words from the set $C_t(i) \cap O_i$ (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, $t \geq 0$.

If two networks of evolutionary processors generate the same language, we say that the networks are equivalent to each other.

For a family $X \in \mathcal{G}$, we denote the family of languages generated by networks of evolutionary processors where all filters are of type X by $\mathcal{E}(X)$. By $\mathcal{E}(REG_i)$ for $i \geq 1$, we denote the family of languages that are generated by networks where all filters are accepted by deterministic finite automata over the alphabet of the network and with at most i states. It is essential that the input alphabets of all the automata in a network are defined over the

whole alphabet of the network because otherwise it could happen that the acceptance or rejection of a word is not defined. For example, let a network be defined over the alphabet $\{a, b\}$ and consider a node in this network that allows all words that do not contain the letter b to enter this node and only those words. If the automaton representing this input filter would be defined only over the alphabet a, then the words containing b could not be handled. This gives a difference between networks with filters from a family $X \in \mathcal{G}$ and those where the filters are represented by automata. In the first situation, the filters of a network can be arbitrarily chosen from the family X, whereas in the second case, they depend on each other in that sense that they must have the same input alphabet.

If we compare two networks with respect to the generative capacity and both have filters that are independent from each other or both have filters represented by automata, then the following fact is helpful.

Lemma 1 *Let X and Y be two families of the set $\mathcal{G}$ such that $X \subseteq Y$ or let $X = REG_i$ and $Y = REG_j$ for natural numbers i and j with $1 \leq i \leq j$. Then the inclusion*

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

holds.

Proof Let X and Y be two language families with the properties given in the statement. Further, let L be a language generated by a network $\mathcal{N}$ with all filters from the family X. Then the network $\mathcal{N}$ has also all filters from the family Y. Hence, $L \in \mathcal{E}(Y)$, which yields the inclusion. □

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

Theorem 1 $\mathcal{E}(REG) = RE$.

3 Some general results

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

Lemma 2 *For every network $\mathcal{N}$ of evolutionary processors over an alphabet V, there is a network $\mathcal{N}'$ of evolutionary processors over the same alphabet V that generates the same language as $\mathcal{N}$ and has the property that its output node N' has the form $N' = (\emptyset, \emptyset, V^*, V^*)$ and no edge leaves the output node 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 $N_j = (M_j, A_j, I_j, O_j)$ has not the required property:

$$N_j \neq (\emptyset, \emptyset, V^*, V^*)$$

or there is an edge leaving node N_j. We define a new network

$$\mathcal{N}' = (V, N_1', N_2', \ldots, N_{n+4}', E', n+4)$$

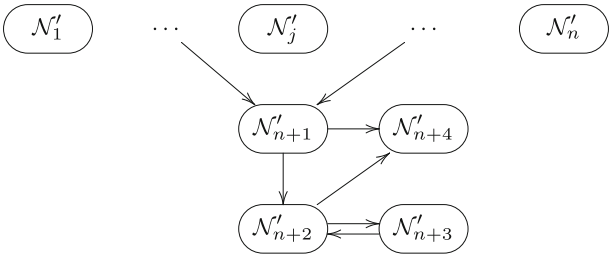
by

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

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 the word 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 $L(\mathcal{N}') \subseteq L(\mathcal{N})$.

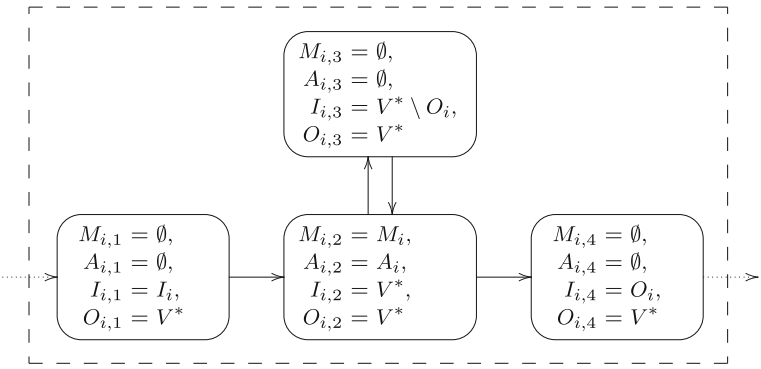
Since $L(\mathcal{N}') = L(\mathcal{N})$ the network $\mathcal{N}'$ has the required properties. □

In the previous lemma, we stated that a kind of normal form exists where the output node does not 'actively' contribute to the language generated (it only receives words and selects words according to the input filter but does not modify them nor does it send words to other nodes).

We now give another kind of normal form.

Lemma 3 *For each network of evolutionary processors over an alphabet V, there is an equivalent network with the property that all output filters are equal to V^* and that the nodes with input filters different from V^* have no evolution rules and no initial words.*

Proof The property stated in the lemma can be achieved in the following way. Let $\mathcal{N}$ be a network of evolutionary processors with a working alphabet V and let $N_i = (M_i, A_i, I_i, O_i)$ be a node of the network. This node can be simulated by the following network:



The incoming edges now arrive at node $N_{i,1}$, the outgoing edges leave from node $N_{i,4}$. The evolution in this network is the same as in the original node N_i. A word that arrives in node N_i also arrives in node $N_{i,2}$ via $N_{i,1}$. A word that leaves the node N_i also leaves the network via $N_{i,4}$, and a word that remains in N_i (because it does not pass the output filter) is sent from $N_{i,2}$ to $N_{i,3}$ and is then returned unchanged to node $N_{i,2}$. All output filters are equal to V^* and nodes with input filters (possibly) different from V^* have no evolution rules and no initial words.

From the construction in the proof of the previous lemma can be seen that if the input filters of the original network belong to a family X which is closed under complement then the input filters of the constructed network also belong to the family X.

The Lemmas 2 and 3 are useful when we want to construct an equivalent network with certain properties from a given network in such a way that the new network simulates the original one. Since the set V^* for an alphabet V belongs to every considered language family except *FIN*, we can often leave nodes with such filters as they are. Those nodes that have to be changed do not contain rules or initial words. Hence, these lemmas allow us to concentrate on simulating the input filters only (and furthermore, the output node – which has a special role – does not need to be considered).

We now continue with some lower bounds for language families.

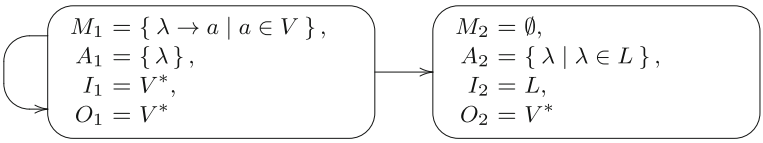
Theorem 2 *Let $X \in \mathcal{G} \cup \{REG_i | i \geq 1\}$. Then each language $L \in X$ can be generated by a NEP $\mathcal{N}$ with at most two nodes and with filters from X.*

Proof Let $X = FIN$. 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 the language L.

Let $X \neq FIN$ and 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 will be communicated to node N_2 which accepts all words that also belong to the language L. The language generated by the network $\mathcal{N}$ is

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

All filters are of type X. □

The previous theorem yields the following more general result.

Corollary 1 *For each family $X \in \mathcal{G} \cup \{REG_i | i \geq 1\}$, we have $X \subseteq \mathcal{E}(X)$.*

Any monoidal language can be generated by a network of evolutionary filters where all filters belong to a subregular family of languages considered in this paper.

Corollary 2 *For each family $X \in \mathcal{G} \cup \{REG_i | i \geq 1\}$, we have $MON \subseteq \mathcal{E}(X)$.*

Proof By the relations given in Figure 1 and the statement of Corollary 1, it is sufficient to show the inclusions

$$MON \subseteq \mathcal{E}(FIN) \quad \text{and} \quad MON \subseteq \mathcal{E}(REG_1).$$

Let V be an alphabet and $L = V^*$. Then the evolutionary network

$$(V, (\{\lambda \to a | a \in V\}, \{\lambda\}, \emptyset, \emptyset), \emptyset, 1)$$

generates the language L. Moreover, the filters are finite and acceptable by a deterministic finite automaton over the alphabet V with one state. Thus, any monoidal language $L = V^*$ belongs to the families $\mathcal{E}(FIN)$ and $\mathcal{E}(REG_1)$. □

4 Computationally complete cases

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

A network of evolutionary processors with arbitrary regular filters can be changed to a network that generates the same language but has filters only that are all either suffix-closed or circular. This yields the following statement.

Theorem 3 *$\mathcal{E}(SUF) = RE$ and $\mathcal{E}(CIRC) = RE$.*

Proof It is sufficient to prove that any recursively enumerable language can be generated by a network of evolutionary processors where all filters are suffix-closed or all filters are circular regular languages.

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 of evolutionary processors with arbitrary regular filters 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

$$\begin{aligned} N'_i &= (M_i, \{X\}A_i\{Y\}, I'_i, O'_i) \text{ for } 1 \le i \le n, \\ N'_{n+1} &= (\{X \to \lambda,\ Y \to \lambda\}, \emptyset, I'_j, V^*), \\ N'_{n+2} &= (\emptyset, \emptyset, V^*, \emptyset), \\ E' &= E \cup \{(i, n+1) | (i, j) \in E\} \cup \{(n+1, n+2)\}. \end{aligned}$$

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, $1 \le i \le n$, they can only be sent to nodes N'_s, $1 \le s \le n$, 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 $\mathcal{E}(CIRC) = RE$, we repeat the previous proof with the following modifications. We set

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

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

Any recursively enumerable language can also be generated if we allow only definite languages as filters.

Theorem 4 $\mathcal{E}(DEF) = RE$.

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, 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 $1 \le i \le t$.

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 representing 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:

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

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 non-terminals 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 $\langle 2 \rangle$:

$$\begin{aligned} \langle 1 \rangle &= \{\langle i, 1 \rangle | 1 \leq i \leq m\}, \\ \langle 2 \rangle &= \{\langle i, 2 \rangle | 1 \leq i \leq m\}. \end{aligned}$$

Further let

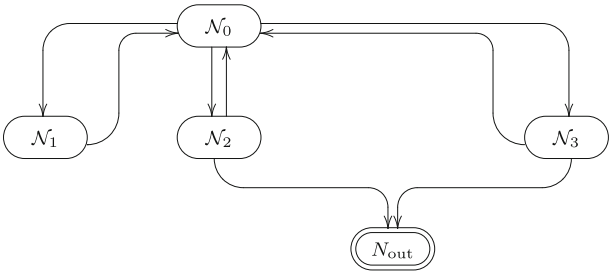
$$\begin{aligned} V' &= \{x' | x \in V\}, \\ [V] &= \{[x] | x \in V\}, \text{ and} \\ \langle V \rangle &= \{\langle x \rangle | x \in V\} \end{aligned}$$

be mutually disjoint sets. We set

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

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

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

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

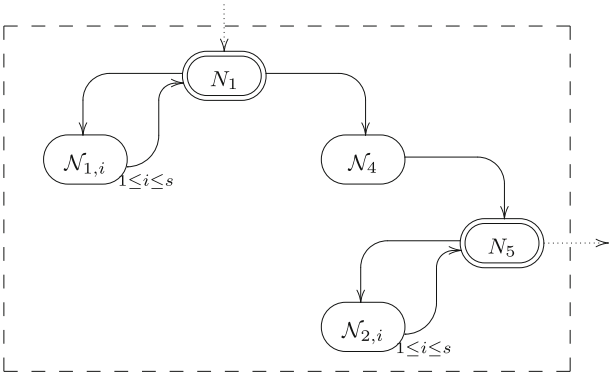
$$M_3 = \{p_{q+1}, p_{q+2}, \ldots, p_n\},$$
$$A_3 = \emptyset,\ I_3 = V^*,\ O_3 = V^*.$$

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

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

This node marks a symbol for removing from the end or for replacing according to a rule. If the marked symbol is not the right-most symbol, the word will be lost in the next communication step because no adjacent node allows the word to enter. If the right-most

symbol is marked for deleting, exactly one of the subnetworks $\mathcal{N}_{1,i}$ for $1 \leq i \leq s$ will take the word, delete the last symbol, and store its information somewhere in the word.

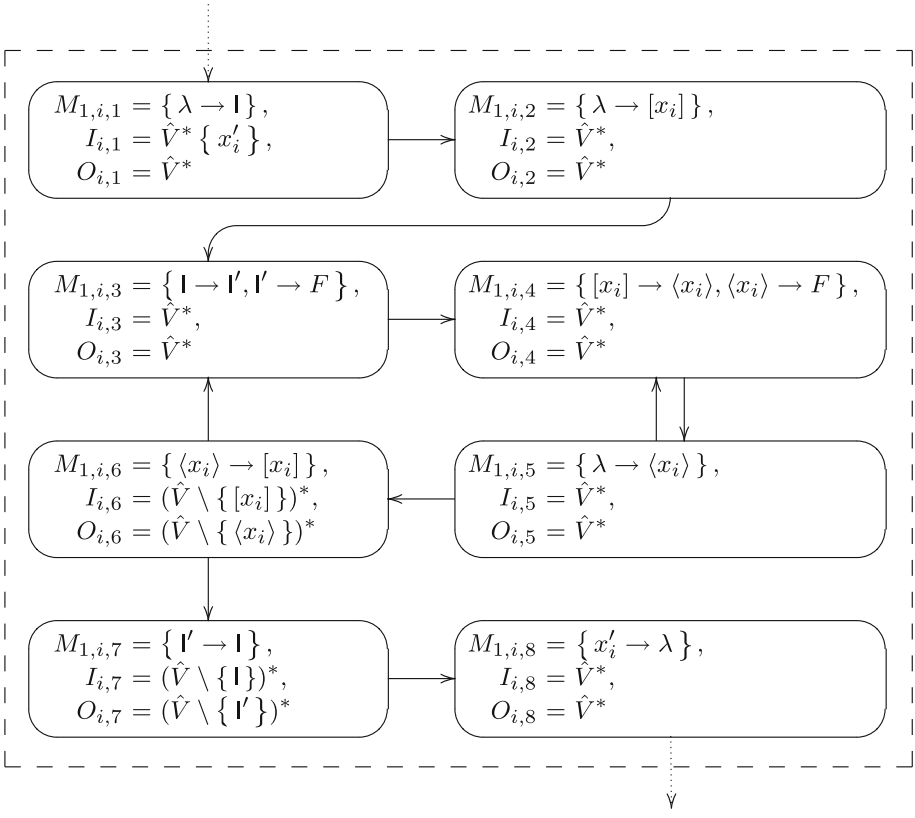
In each subnetwork $\mathcal{N}_{1,i}$ for $1 \leq i \leq s$, the relocation (elimination and storage of its information somewhere else) of the last symbol is performed if it is equal to x_i'. In the subnetwork $\mathcal{N}_4$, the application of a rule is simulated. The node N_5 is defined by

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

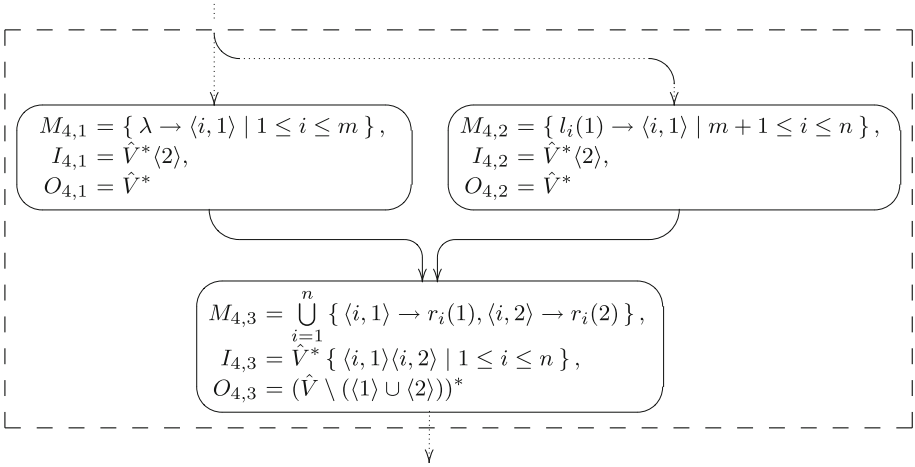
In each subnetwork $\mathcal{N}_{2,i}$ for $1 \leq i \leq s$, the letter x_i is restored at the end of the current word if it has been there originally.

For $1 \leq i \leq s$, 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:



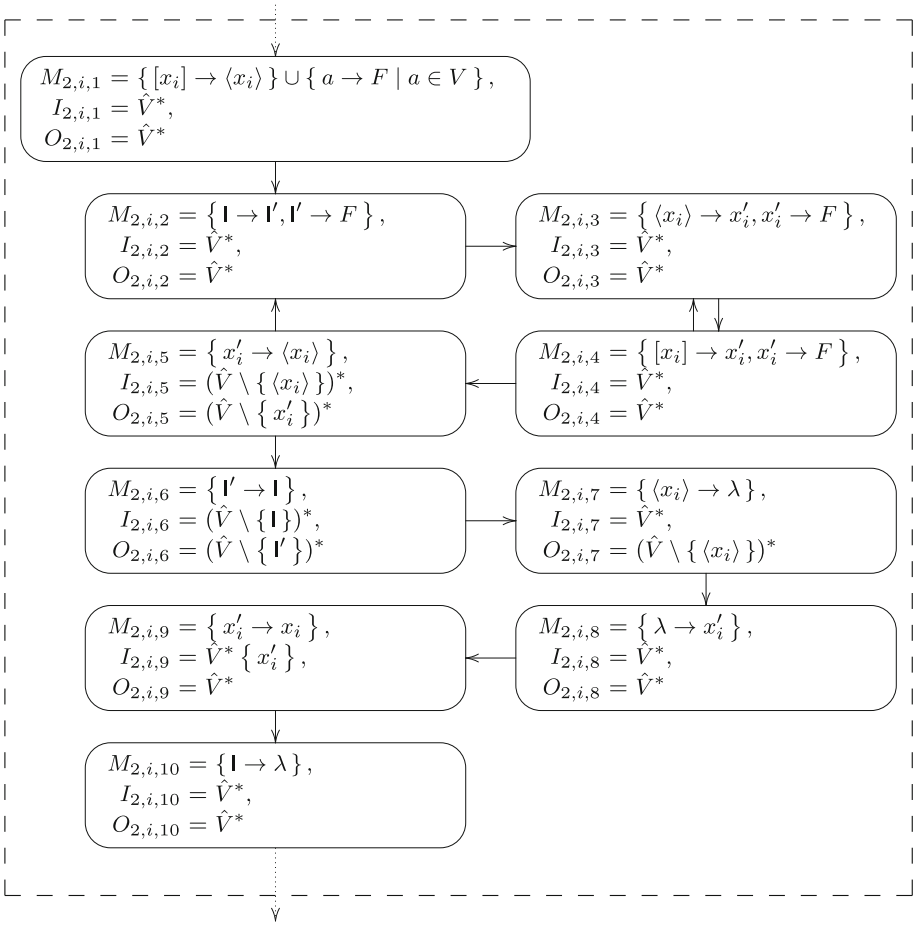
In the subnetwork $\mathcal{N}_4$, the application of a rule p_i with $1 \leq i \leq m$ is simulated. This subnetwork has the following form (the initial set is empty in each node):



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 'l'. 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):



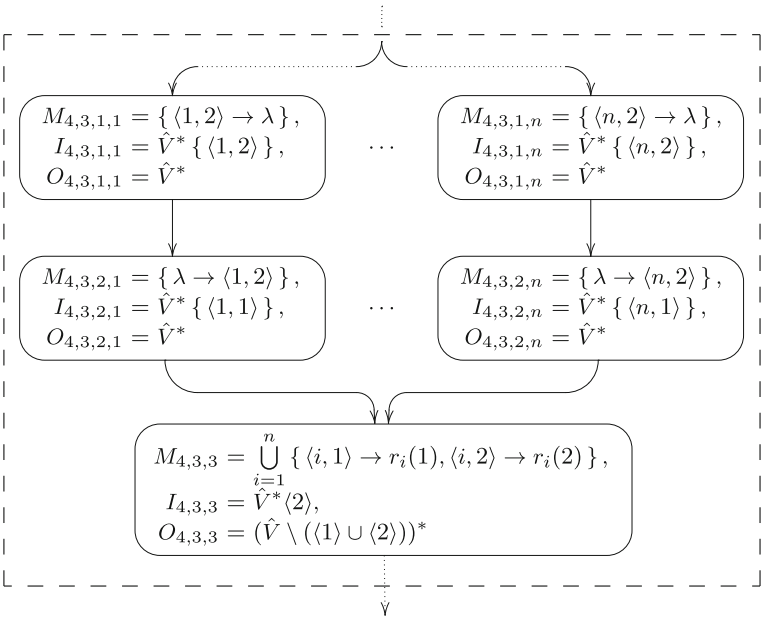
The network $\mathcal{N}$ constructed above for an arbitrary phrase structure grammar has only filters that are definite languages. This proves the claim. □

Even if we restrict the filters to combinational languages, any recursively enumerable language can be generated.

Theorem 5 $\mathcal{E}(COMB) = RE$.

Proof The network $\mathcal{N}$ constructed in the proof of Theorem 4 with definite filters has—up to one exception – only combinational filters. The exception is node $N_{4,3}$ where the input filter $\hat{V}^*\{\langle i, 1\rangle\langle i, 2\rangle | 1 \leq i \leq n\}$ is not combinational.

We now replace the node $N_{4,3}$ by the following network (the initial set is empty in each node):



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

On the path of the nodes $N_{4,3,1,i}$ and $N_{4,3,2,i}$ for $1 \leq i \leq n$, the incoming word is checked for the suffix $\langle i,1\rangle\langle i,2\rangle$. A word enters node $N_{4,3,3}$ if and only if it enters node $N_{4,3}$. In both nodes, it is changed in the same manner. Hence, a word leaves the subnetwork if and only if it leaves node $N_{4,3}$. Thus, the subnetwork simulates the behaviour of node $N_{4,3}$ and we obtain a network with combinational filters only that generates the same language as the network $\mathcal{N}$.

This yields the relation $\mathcal{E}(COMB) = \mathcal{E}(DEF)$ and, together with the equality $\mathcal{E}(DEF) = RE$ (Theorem 4), we obtain the equality $\mathcal{E}(COMB) = RE$. □

With the help of the previous theorem, we show the next one.

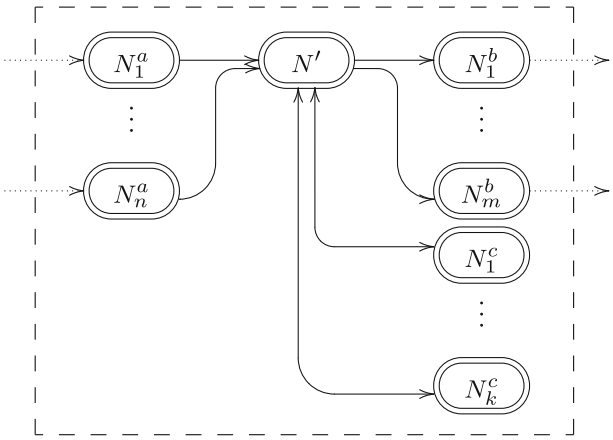
Theorem 6 $\mathcal{E}(UF) = RE$.

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

Let $L \in RE$. By Theorem 5 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:

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

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).

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 we get $L \in \mathcal{E}(UF)$. This proves the other inclusion $RE \subseteq \mathcal{E}(UF)$. □

By the relations shown Fig. 1, Lemma 1, and Theorem 1, we obtain the following theorem.

Theorem 7 $\mathcal{E}(ORD) = \mathcal{E}(NC) = \mathcal{E}(PS) = RE$.

5 Finite filters

In this section, we discuss the case of finite filters.

We first show that we can assume without loss of generality that the output node has no outgoing edges. Let $\mathcal{N}' = (V, N_1, N_2, \ldots, N_n, E', j)$ be a network with finite filters only. We first add a node N_{n+1} that is almost the same node as the output node N_j but has no outgoing edges and declare this node to be the new output node. This yields a new network $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, N_{n+1}, E, n+1)$ with

$$N_{n+1} = (M_j, A_j, I_j, O_j) \quad \text{and} \quad E = E' \cup \{(i, n+1) | (i, j) \in E'\}.$$

Now the output node only collects words and modifies them according to its rules (like the original output node) but does not bring them back to circulation in the network. However, at each moment, the nodes N_j and N_{n+1} contain the same words. Thus, the networks $\mathcal{N}'$ and $\mathcal{N}$ generate the same language.

Corollary 3 *For each network with finite filters only, there is an equivalent network with only finite filters and the property that the output node has no outgoing edges.*

As we will see in the sequel, the language generated by a network with finite filters only and with a deleting or substituting output node is finite. We also show how the language can be computed from the filters. From now on, we assume that the network considered has $n+1$ nodes $N_1, N_2, \ldots N_{n+1}$ where the node N_{n+1} is the output node and has no outgoing edges.

If a node is a substitution or insertion node, then words cannot become shorter in this node. If a word in such a node among the nodes $N_1, N_2, \ldots, N_n$ is longer than any word of the respective output filter, then it is useless – it cannot move to another node and it cannot be changed to a word that could leave the node, so neither itself nor a possible modification can ever enter the output node. We set

$$m_i = \max\{|w| | w \in O_i\}$$

as the maximal length of words in the output filter O_i for $1 \leq i \leq n+1$. In the computation of a network, there are two alternating phases: From an even moment to an odd moment, words are modified in a node according to the node's rules (evolutionary step). From an odd moment to an even moment, words are communicated in the network (communication step). For the generated language, the exact moments are not of importance (only the parity of the moment).

We consider the cooperation of the nodes $N_1, N_2, \ldots, N_n$. We take, for each node, all words that are contained in this node in an even moment and all those present in some odd moment unless they are useless:

– For $i = 1, 2, \ldots, n$, if N_i is deleting, we set

$$L_t^{\mathrm{e}}(i) = \bigcup_{m=0}^{t} C_{2m}(i) \quad \text{for } t \geq 0 \text{ and}$$

$$L_t^{\mathrm{o}}(i) = \bigcup_{m=0}^{t} C_{2m+1}(i) \quad \text{for } t \geq 0.$$

– For $i = 1, 2, \ldots, n$, if N_i is substituting or inserting, we set

$$L_t^{\mathrm{e}}(i) = V_{m_i} \cap \bigcup_{m=0}^{t} C_{2m}(i) \quad \text{for } t \geq 0 \text{ and}$$
$$L_t^{\mathrm{o}}(i) = V_{m_i} \cap \bigcup_{m=0}^{t} C_{2m+1}(i) \quad \text{for } t \geq 0.$$

All these sets are monotonically increasing with the time t. In a deleting node N_i, a word cannot be longer than

$$\max\{|w||w \in A_i \cup I_i\}.$$

For a substituting or inserting node N_i, a word in any of the sets $L_t^{\mathrm{o}}(i)$ and $L_t^{\mathrm{e}}(i)$ for $t \geq 0$ cannot contain more than m_i letters. Thus, all these sets are finite. Hence, there is a moment t^* such that none of these sets changes anymore (for $1 \leq i \leq n$, we have $L_{t^*}^{\mathrm{e}}(i) = L_{t^*+1}^{\mathrm{e}}(i)$ and $L_{t^*}^{\mathrm{o}}(i) = L_{t^*+1}^{\mathrm{o}}(i)$). This moment can be computed as well as the sets $L_{t^*}^{\mathrm{e}}(i)$ and $L_{t^*}^{\mathrm{o}}(i)$ for $1 \leq i \leq n$.

If the output node is deleting, then the language generated is

$$L(\mathcal{N}) = L_{t^*}^{\mathrm{e}}(n+1) \cup L_{t^*}^{\mathrm{o}}(n+1).$$

The set is finite and can be computed.

If the output node is substituting or inserting, then the language generated consists of all words that are present in the output node at the beginning of the computation, all words that are communicated to this node at some moment, all words that are derived from those words, and all words that are derived from words contained in the node that do no leave the node in a communication step. The language generated can be obtained successively as follows:

$$\begin{aligned}
L_1 &= A_{n+1} \cup \bigcup_{m \geq 1} \bigcup_{(k,n+1) \in E} (C_{2m-1}(k) \cap O_k \cap I_{n+1}),\\
L_2 &= L_1 \cup \{v | \exists w \in L_1 \, \exists r \in M_{n+1} : w \Longrightarrow_r v\},\\
L_l &= L_{l-1} \cup \{v | \exists w \in L_{l-1} \setminus O_{n+1} \, \exists r \in M_{n+1} : w \Longrightarrow_r v\} \text{ for } l \geq 3,\\
L(\mathcal{N}) &= \bigcup_{i \geq 1} L_i.
\end{aligned}$$

The set L_1 can also be written as

$$L_1 = A_{n+1} \cup \bigcup_{(k,n+1) \in E} \left(\Big(\bigcup_{m \geq 1} C_{2m-1}(k) \Big) \cap O_k \cap I_{n+1} \right)$$

due to commutativity and distributivity. If a node N_k is deleting, then

$$\bigcup_{m \geq 1} C_{2m-1}(k) = L_{t^*}^{\mathrm{o}}(k).$$

If a node N_k is substituting or inserting, then

$$L_{t^*}^{\mathrm{o}}(k) = \left(\bigcup_{m \geq 1} C_{2m-1}(k) \right) \cap V_{m_k}$$

and, since $O_k \subseteq V_{m_k}$, we obtain

$$\left(\bigcup_{m \geq 1} C_{2m-1}(k) \right) \cap O_k = L^{\mathrm{o}}_{t^*}(k) \cap O_k.$$

This yields for the language L_1

$$L_1 = A_{n+1} \cup \bigcup_{(k,n+1) \in E} \left(L^{\mathrm{o}}_{t^*}(k) \cap O_k \cap I_{n+1} \right). \tag{1}$$

The language is finite and computable.

If the output node is substituting, then, from set L_2 on, no word can occur that is longer than any word of L_1 (when deriving, the length does not change). Hence, there is a number $l \geq 1$ such that $L_{l+1} = L_l$ and then

$$L(\mathcal{N}) = \bigcup_{i \geq 1}^{l} L_i.$$

Thus, the language generated is finite and computable.

Corollary 4 *The language generated by a network with finite filters only and with a deleting or substituting output node is finite and can be computed from the filters.*

If the output node is inserting, we continue as follows. We compute all words of the generated language up to the length of $m_{n+1} + 1$. Words with that length cannot leave the output node anymore. From those words on, any word belongs to the generated language that can be obtained by successively applying rules of the node.

By $L_i^<$ we denote the set of all words of L_i that have a length less than $m_{n+1} + 1$; by $L_i^=$ we denote the set of all words of L_i that have a length of $m_{n+1} + 1$; by $L_i^>$ we denote the set of all words of L_i that have a length greater than $m_{n+1} + 1$ for $i \geq 1$:

$$\begin{aligned} L_i^< &= L_i \cap V_{m_{n+1}}, \\ L_i^= &= L_i \cap V^{m_{n+1}+1}, \\ L_i^> &= L_i \setminus V_{m_{n+1}+1}. \end{aligned}$$

The sets $L_i^<$ for $i \geq 1$ are all subsets of the finite set $V_{m_{n+1}}$. By construction of these sets, we know that the equality $L_i^< = L_{i+1}^<$ implies the equality of all further languages: $L_{i+1}^< = L_{i+2}^<$. Hence, all the languages $L_i^<$ for $i \geq 1$ are finite and computable.

Let

$$K(\mathcal{N}) = \bigcup_{i \geq 1} L_i^< \tag{2}$$

be the set of all words generated by the network $\mathcal{N}$ that have a length of at most m_{n+1} letters.

The sets $L_i^=$ for $i \geq 1$ are all subsets of the finite set $V^{m_{n+1}+1}$. By construction of these sets, we know that the equality $L_i^= = L_{i+1}^=$ implies the equality of all further languages: $L_{i+1}^= = L_{i+2}^=$. Hence, all the languages $L_i^=$ for $i \geq 1$ are finite and computable. For the sets $L_i^>$ with $i > 1$, we obtain

$$L_i^> = L_{i-1}^> \cup \{v | \exists w \in L_{i-1}^= \cup L_{i-1}^> \, \exists r \in M_{n+1} : w \Longrightarrow_r v\}.$$

The union of all those sets consists of all words that are in L_1 and have a length greater than $m_{n+1} + 1$, all words that are derived (in at least one step) from these words by rules of the

set M_{n+1}, as well as all words that are derived (in at least one step) from the words of length $m_{n+1}+1$ contained at some moment in the output node. Let $S(\mathcal{N})$ denote the set

$$S(\mathcal{N}) = L_1^{>} \cup \bigcup_{i \geq 1} L_i^{=} \tag{3}$$

and let $D(\mathcal{N})$ be the set of all words that are derived in one or more steps from the words of the set $S(\mathcal{N})$. Since the output node is inserting, no word from the set $S(\mathcal{N})$ or $D(\mathcal{N})$ can leave the output node (because the words are longer than the longest word in the output filter O_{n+1}). Let U be the set of all letters that can be inserted by this node:

$$U = \{a | \lambda \to a \in M_{n+1}\}.$$

Then we obtain

$$D(\mathcal{N}) = \{u_1 w_1 u_2 w_2 \dots u_l w_l u_{l+1} | w_1 w_2 \dots w_l \in S(\mathcal{N}),\ u_1 u_2 \dots u_{l+1} \in U^+\} \tag{4}$$

is the set of all words that are obtained by inserting at least one symbol from the set U into a word from the set $S(\mathcal{N})$. Since the set $S(\mathcal{N})$ is finite and computable, the language $D(\mathcal{N})$ is regular and computable. This yields for the language generated by the network

$$L(\mathcal{N}) = \bigcup_{i \geq 1} L_i^{<} \cup \bigcup_{i \geq 1} L_i^{=} \cup L_1^{>} \cup \bigcup_{i \geq 2} L_i^{>}.$$

Corollary 5 *The language generated by a network with finite filters only and with an inserting output node can be computed from the filters and is the union*

$$L(\mathcal{N}) = K(\mathcal{N}) \cup S(\mathcal{N}) \cup D(\mathcal{N})$$

of the sets $K(\mathcal{N})$, $S(\mathcal{N})$, and $D(\mathcal{N})$ defined above (Eqs. 2, 3, and 4).

The language generated by the network is regular and computable from the filters of the network, because the united sets are regular ($K(\mathcal{N})$ and $S(\mathcal{N})$ even finite) and computable.

Corollary 6 $\mathcal{E}(FIN) \subseteq REG$.

There is a regular language that cannot be generated by a network with finite filters only.

Lemma 4 *The language $L = \{a\} \cup \{a^n | n \geq 3\}$ can not be generated by a network with finite filters only.*

Proof Suppose the language L is generated by a network with only finite filters. According to the considerations above, the output node is an inserting node (otherwise, only finite languages are generated). Hence, the rule set is $M = \{\lambda \to a\}$. Since the letter a belongs to the language generated, it belongs to the initial language of the output node or it arrives there by communication. In both cases, the word aa is obtained by evolution in the output node and belongs to the language generated which is a contradiction.

Hence, there is no network with only finite filters that generates the language L. Thus, $L \notin \mathcal{E}(FIN)$. □

The previous statement yields the proper inclusion.

Theorem 8 $\mathcal{E}(FIN) \subset REG$.

Proof From Corollary 6, we have the inclusion $\mathcal{E}(FIN) \subseteq REG$. By Lemma 4, we know that the regular language $L = \{a\} \cup \{a^n | n \geq 3\}$ does not belong to the family $\mathcal{E}(FIN)$. Hence, the inclusion is proper. □

We now continue with certain normal forms for networks with finite filters.

Lemma 5 *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. We assume that the output node has no outgoing edges what can always be achieved due to Corollary 3.

The set of all words that arrive in the output node is

$$B = \bigcup_{(k,j)\in E} \left(L^{\mathrm{o}}_{t^*}(k) \cap O_k \cap I_j\right)$$

according to Eq. 1.

If we add the words of the set B to the initial language of the output node, we can remove all incoming edges from the output node, hence, we can delete all other nodes from the network without changing the language generated. Thus, the network $\mathcal{N}' = (V, N_1', \emptyset, 1)$ with the node $N_1' = (M_j, A_j \cup B, \emptyset, O_j)$ has only one node with finite filters and generates the same language as $\mathcal{N}$. □

Lemma 6 *For each NEP $\mathcal{N}$ over an alphabet V with only finite filters, we can construct an equivalent NEP $\mathcal{N}'$ with two processors where every filter is equal to V^*.*

Proof Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a NEP with finite filters. We assume that the output node has no outgoing edges what can always be achieved due to Corollary 3.

If the output node is substituting or deleting, the network generates a finite language L according to Corollary 4. Hence, the network

$$\mathcal{N}' = (V, N_1', \emptyset, 1)$$

with

$$N_1' = (\emptyset, L, V^*, V^*)$$

also generates the language L and has all filters from the family REG_1.

If the output node is inserting, then there are finite sets $K(\mathcal{N})$ and $S(\mathcal{N})$, such that all words of the language $L(\mathcal{N})$ are in one of these sets or belong to the set $D(\mathcal{N})$ that are derived from the words of $S(\mathcal{N})$ (see Corollary 5).

We now construct a network

$$\mathcal{N}' = (V, N_1', N_2', \{(1,2),(1,1)\}, 2)$$

with the two nodes

$$N_1' = (M_j, S(\mathcal{N}), V^*, V^*) \quad \text{and} \quad N_2' = (\emptyset, K(\mathcal{N}) \cup S(\mathcal{N}), V^*, V^*).$$

The language generated by this network consists of all words of the set $K(\mathcal{N})$, all words of the set $S(\mathcal{N})$, and all words that are derived from words of $S(\mathcal{N})$ by rules of the original output node N_j.

Hence,

$$L(\mathcal{N}') = K(\mathcal{N}) \cup S(\mathcal{N}) \cup D(\mathcal{N}) = L(\mathcal{N})$$

and every filter is equal to V^*. □

The previous corollary immediately yields the following inclusion.

Corollary 7 $\mathcal{E}(FIN) \subseteq \mathcal{E}(REG_1)$.

With the next lemma, we can even prove the strictness of the inclusion.

Lemma 7 *The language* $L = \{w | w \in \{a, b, c\}^* \text{ and } |w|_a = |w|_b = |w|_c\}$ *cannot be generated by a network with finite filters only.*

Proof Let us assume that the language L can be generated by a network with finite filters only.

According to Lemma 5, there is a network $\mathcal{N}$ with one node N and finite filters only that generates the language L. Since L is infinite and the initial language of the node N contains only finitely many words, infinitely many words have to be produced by the rules. Since the numbers of occurrences of the letters a, b, and c are unbounded, the node N has rules for inserting all three letters. But then, from a word $w \in L$ that is longer than any word in the output filter, also the word aw is obtained which does not belong to the language L which is a contradiction. Hence, the language L cannot be generated by a network with finite filters only. □

In [11], it was shown that the language

$$L = \{w | w \in \{a, b, c\}^* \text{ and } |w|_a = |w|_b = |w|_c\}$$

belongs to the family $\mathcal{E}(REG_1)$. Together with Lemma 7 and the inclusion from Corollary 7, we obtain the proper inclusion.

Theorem 9 $\mathcal{E}(FIN) \subset \mathcal{E}(REG_1)$.

6 Further computationally non-complete cases

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

Theorem 10 $\mathcal{E}(MON) = \mathcal{E}(COMM)$.

Proof According to Lemma 1, we have the inclusion $\mathcal{E}(MON) \subseteq \mathcal{E}(COMM)$. We now show the converse inclusion $\mathcal{E}(COMM) \subseteq \mathcal{E}(MON)$.

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 Lemma 3, such a network can always be obtained).

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 $\hat{V}$:

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

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 $h(L)$.

The idea behind the simulation is as follows: Let $w \in V$ be the arriving word. The word w passes the input filter I if and only if the language $\tilde{I}$ contains a permutation of the word $h(w)$. The language $\tilde{I}$ is like I commutative and regular. The simulating network generates every word of the language $\tilde{I}$ scattered over one copy of w each and checks whether the generated word is letter-equivalent up to the isomorphism h to the word w. If so, then the language $\tilde{I}$ contains a permutation of the word $h(w)$ and hence $w \in I$ (then the original word w is restored by deleting all letters of $h(w)$ and is allowed to leave the subnetwork). If the generated word is not a permutation of the word w, then the whole word will be lost. If the subnetwork does not generate a permutation of w at all, then no copy of the word w leaves the network. Hence, the word w passes the subnetwork if and only if it passes the input filter I.

Let $G = (N, \tilde{V}, P, S)$ be a regular grammar that generates the language $\tilde{I}$. We now create the informally described network that simulates the grammar G. We assume that all rules in P have the form $A \to aB$ or $A \to a$ for non-terminals $A, B \in N$ 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 non-terminal $X \in N$, we set

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

as the set of symbols representing the right hand sides of the non-terminating rules,

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

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

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

as the set of all non-terminals that are generated by X, and

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

as the set of all terminal symbols that are generated at the same time as X. The terminating non-terminals (those non-terminals $X \in N$ 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 $R = \{\langle aY\rangle | \exists X \in N : X \to aY \in P\}$.

For every non-terminal $X \in N$, we define two nodes

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

as follows:

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

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 non-terminal $X \in N$. 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 non-terminal $X \in N_t$, we additionally define a node

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

by the sets $M_{X_t} = \{\lambda \to a | a \in T_t(X)\}$, $A_{X_t} = \emptyset$, 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. Any resulting word contains now the word w which entered the subnetwork as a scattered subword and a permutation of a word of the language $\tilde{I}$ as a scattered subword. Then it will be checked whether these two subwords are letter-equivalent. For this purpose, 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

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

for $1 \le i \le r$. 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 $1 \le i \le r$, and every node $N_{\tilde{x}_i}$ to $N_{x_{i+1}}$ for $1 \le i \le r-1$. 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 letter-equivalence is preserved by these insertions. 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 up to the isomorphism h to the word generated by the grammar G (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

$$I_{x_i'} = \left(\bigcup_{k=1}^{i-1} \{x_k', \tilde{x}_k'\} \cup \bigcup_{k=i}^{r} \{x_k, \tilde{x}_k, x_k', \tilde{x}_k'\} \right)^*$$

and

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

for $1 \le i \le r$. 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

$$N_{\tilde{V}'} = (\{\tilde{x}_i' \to \lambda | 1 \le i \le r\}, \emptyset, (V' \cup \tilde{V}')^*, (V')^*)$$

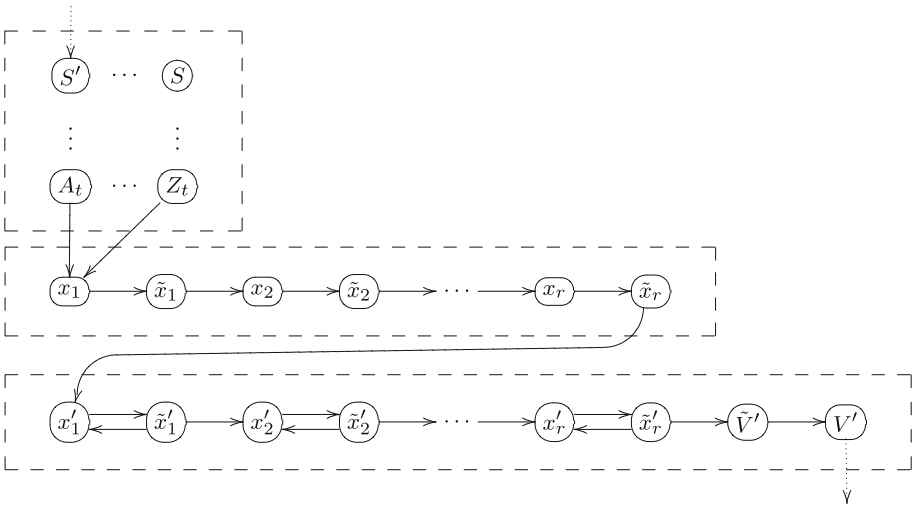
and replace the primed symbols by their unprimed counterparts in node

$$N_{V'} = (\{x_i' \to x_i | 1 \le i \le r\}, \emptyset, (V')^*, V^*).$$

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 $1 \le j \le r$, and every node $N_{\tilde{x}_i'}$ to $N_{x_{i+1}'}$ for $1 \le j \le r-1$ 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 index i then the word cannot move on because the trap symbol F is introduced. If for $1 \le i \le r$ 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 (the input filter I and output filter O 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 where each node is represented by its index according to the definition given above:



The filters are all monoidal. The entrance node of the network is $N_{S'}$. Hence, the edges that lead to N now lead to $N_{S'}$. The exit node of the network is $N_{V'}$. Hence, the edges that leave N 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 $\mathcal{E}(COMM) \subseteq \mathcal{E}(MON)$ follows which yields with $\mathcal{E}(MON) \subseteq \mathcal{E}(COMM)$ the equality. □

Theorem 11 $\mathcal{E}(MON) = \mathcal{E}(NIL)$.

Proof By Lemma 1, we already have the inclusion $\mathcal{E}(MON) \subseteq \mathcal{E}(NIL)$.

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

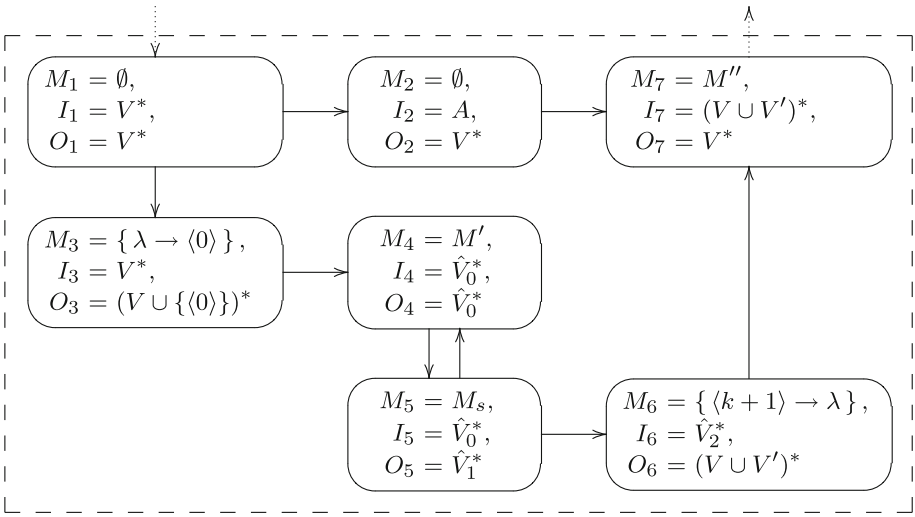
Let $\mathcal{N}$ be a NEP with a working alphabet V where all filters are nilpotent. The complement of a nilpotent language is also a nilpotent language. According to the considerations in the beginning of this section, we can assume that $\mathcal{N}$ has the property that all output filters are V^* 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

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

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 alternately 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 filters, which are monoidal, 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 know 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 $\mathcal{E}(MON)$. □

The next lemma states that the language family which is characterized by networks where all filters are from one of the families *MON*, *COMM*, or *NIL* is not computational complete.

Lemma 8 *The language* $L = \{wb | w \in \{a, b\}^*\}$ *cannot be generated by a network with monoidal filters only.*

Proof Suppose $L \in \mathcal{E}(MON)$. Then there is a NEP $\mathcal{N}$ which has only filters that belong to the class *MON* and which generates the language L. Since L is infinite and networks with only substitution and deletion nodes generate finite languages, the network $\mathcal{N}$ contains an inserting processor. The number of occurrences of the letter a is unbounded (for each natural number n, there is a word $w \in L$ with more than n occurrences of a). Hence, there are a natural number $s \geq 0$ and letters $x_0, x_1, \ldots, x_s$ with $x_s = a$ such that the network contains the rules $\lambda \to x_0$ and $x_i \to x_{i+1}$ for $0 \leq i \leq s-1$ and there is a word $w_1 a w_2 \in L$ which is derived from a word $v_1 v_2$ by applying these rules (possibly not only these rules), starting with the insertion of x_0 between v_1 and v_2. Instead, x_0 could also be inserted at the end of $v_1 v_2$. All words derived from $v_1 x_0 v_2$ are letter-equivalent to those derived from $v_1 v_2 x_0$. Thus, if a word derived from $v_1 x_0 v_2$ can pass a filter then also a word that is derived from $v_1 v_2 x_0$ in the same manner can pass that filter. Hence, in the same way how $w_1 a w_2$ is derived and communicated to the output node, also the word $w_1 w_2 a$ is derived and communicated to the output node. But $w_1 w_2 a \notin L$. Thus, the language L cannot be generated by a network where the filters belong to the family *MON*. □

According to the previous lemma, the language family which is characterized by networks where all filters are from one of the families *MON*, *COMM*, or *NIL* is a proper subset of the family of all recursively enumerable languages. Due to Theorem 5, it is sufficient to prove the proper inclusion $\mathcal{E}(MON) \subset \mathcal{E}(COMB)$.

Theorem 12 $\mathcal{E}(MON) \subset \mathcal{E}(COMB)$.

Proof The inclusion $\mathcal{E}(MON) \subseteq \mathcal{E}(COMB)$ follows from Lemma 1.

Let

$$L = \{wb | w \in \{a, b\}^*\}.$$

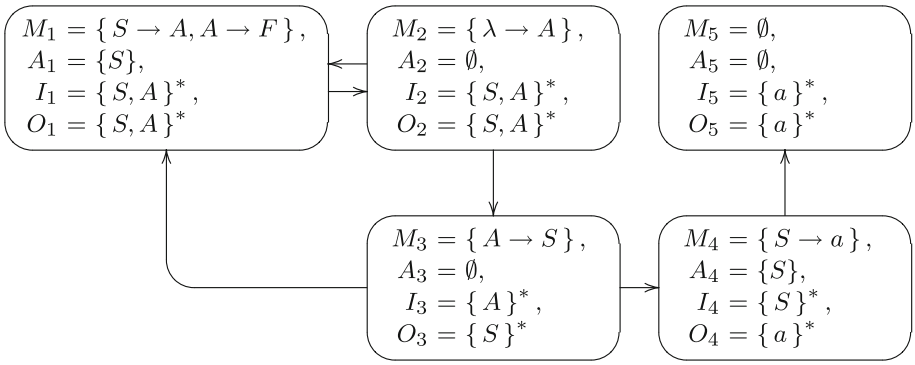
The language L can be written as V^*B with $V = \{a, b\}$ and $B = \{b\}$. Hence, the language L is combinational and, by Corollary 1, $L \in \mathcal{E}(COMB)$.

According to Lemma 8, $L \notin \mathcal{E}(MON)$ and therefore $\mathcal{E}(MON) \subset \mathcal{E}(COMB)$. □

We now present some properties of the family $\mathcal{E}(MON)$.

Lemma 9 *The language* $L = \{a^{2^n} | n \geq 0\}$ *belongs to the family* $\mathcal{E}(MON)$.

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 alternately 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, $L(\mathcal{N}) = \{a^{2^n} | n \geq 0\}$. □

The previous lemma implies the following statement.

Lemma 10 *The family* $\mathcal{E}(MON)$ *contains a non-semi-linear (hence non-regular and non-context-free) language.*

In [11], it was shown that every recursively enumerable language can be generated by a network where all the filters belong to a family REG_i for $i \geq 2$ (every filter is accepted by a deterministic finite automaton with at most i states and with the alphabet of the network as input alphabet). Furthermore, it was shown that if the number of states is bounded by one, then not every regular language but non-context-free languages can be generated.

We now relate the family $\mathcal{E}(REG_1)$ to other families.

Theorem 13 $\mathcal{E}(REG_1) \subset \mathcal{E}(MON)$.

Proof The family of languages that are accepted by deterministic finite automata with exactly one state consists of the empty set and all monoidal languages V^* for an alphabet V.

Let $\mathcal{N}$ be a network over an alphabet V with filters from the set REG_1. Then every filter is either the set V^* (if the only state of the representing automaton is accepting) or the empty set (if the only state is not accepting).

If the input filter of a node N is the empty set, then no word can enter this node. Thus, we can remove all edges that lead to the node N and set the input filter to an arbitrary monoidal language without changing the language generated. All nodes from which no directed path to the output node exists can be removed, too, because they do not contribute to the language. After these changes, we have a network $\mathcal{N}'$ that generates the same language as $\mathcal{N}$ but has only monoidal input filters. If the output filter of a node is the empty set, then no word can leave this node. If this is the case for a node N which is not the output node, then this node N is useless in that sense that it does not contribute to the language generated. So, we can eliminate this node together with all incident edges without changing the language. Again, we can also remove all nodes that are not connected to the output node anymore. After these changes, we have a network $\mathcal{N}''$ that generates the same language as $\mathcal{N}$, that has only monoidal input filters, and all nodes different from the output node have monoidal output filters. If the output node $N = (M, A, I, O)$ has as the output filter O the empty set, then we replace this filter by the language V^*, delete all outgoing edges, and add a new edge to this node itself (or to a new node $N' = (\emptyset, \emptyset, V^*, V^*)$ and back if loops should be avoided). This yields a network $\mathcal{N}'''$. All words that are present in the output node of one network after a communication step are also present in the output node of the other network after a communication step. Hence, the network $\mathcal{N}'''$ generates the same language as $\mathcal{N}$ and has only monoidal filters.

A witness language for the strictness of the inclusion is the language

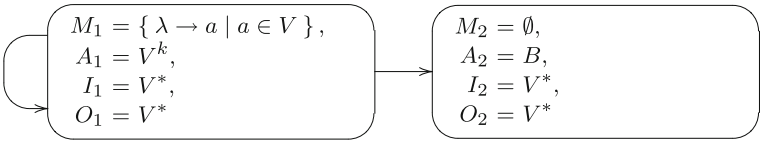
$$L = \{a^{2^n} | n \geq 0\}.$$

According to Lemma 9, we have $L \in \mathcal{E}(MON)$. In [11], it was shown that the length difference between two words is bounded by a constant. Hence, $L \notin \mathcal{E}(REG_1)$. □

Theorem 14 *$NIL \subset \mathcal{E}(REG_1)$.*

Proof According to the Theorems 2 and 9, every finite language can be generated by a network with filters from the family REG_1.

Let L be an infinite nilpotent language over an alphabet V. Then L can be expressed as $L = V^*V^{k+1} \cup B$ for some natural number $k \geq 0$ and set $B \subseteq V_k$. The language L can be generated by the network $\mathcal{N} = (V, N_1, N_2, E, 2)$ shown in the following picture:



In the node N_1, every word of the set V^{k+1} is generated in the first evolutionary step. In every further evolutionary step, the length of the words is increased by one. All words obtained there have a length of at least $k + 1$ and are sent to the output node which has the finite language B of words of length at most k belonging to L as its initial language.

Hence, any nilpotent language can be generated by a network with filters from the set REG_1. □

Theorem 15 *$COMM \subset \mathcal{E}(REG_1)$.*

Proof Let L be a commutative regular language over an alphabet V. Then there is a regular Grammar $G = (N, T, P, S)$ that generates the language L and where every rule has the form $A \to aB$ or $A \to a$ for non-terminal symbols A, B and a terminal symbol a.

For any word $w = a_1a_2 \dots a_n$ of the language L, also every permutation $a_{i_1}a_{i_2} \dots a_{i_n}$ with $\{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}$ belongs to L. Let us consider a derivation

$$S \Longrightarrow a_1A_1 \Longrightarrow a_1a_2A_2 \Longrightarrow \cdots \Longrightarrow a_1a_2 \dots a_{n-1}A_{n-1} \Longrightarrow a_1a_2 \dots a_n \tag{5}$$

of the word w. If we insert—starting from the empty word λ—successively the letters $a_1, a_2, \dots, a_n$ at arbitrary positions and keep in mind the current non-terminal symbol (such that we cannot insert a symbol b from a non-terminal symbol A if there is no rule $A \to bB$ or $A \to b$), then we obtain words of the language, too (namely permutations of the word w). Thus, we can consider a rule $A \to aB$ in such a way that the terminal symbol a is inserted somewhere into the current sentential form and B is the new current non-terminal symbol. Such a derivation process can be simulated by a network of evolutionary processors.

We now construct such a network $\mathcal{N}$. For each right hand side aB of a rule with $B \in N \cup \{\lambda\}$, we build a node $N_{\langle aB\rangle}$ by

$$N_{\langle aB\rangle} = (\{\lambda \to a\}, A_{\langle aB\rangle}, V^*, V^*)$$

where $A_{\langle aB\rangle} = \{\lambda\}$ if aB is the right hand side of a rule for the start symbol S, otherwise $A_{\langle aB\rangle} = \emptyset$. For each rule $A \to aB$, we connect any node $N_{\langle xA\rangle}$ (for $x \in T$) to the node $N_{\langle aB\rangle}$. Additionally, we set

$$N_{\langle\rangle} = (\emptyset, \emptyset, V^*, V^*),$$

declare this as the output node, and connect every node $N_{\langle a\rangle}$ for $a \in T$ to this node.

Let us consider again the derivation given in (5) for the word $a_1a_2 \dots a_n$. This word can be obtained in the network $\mathcal{N}$ as follows:

$$\begin{aligned}
\lambda \in C_0(\langle a_1A_1\rangle) &\Longrightarrow a_1 \in C_1(\langle a_1A_1\rangle) \vdash a_1 \in C_2(\langle a_2A_2\rangle) \\
&\Longrightarrow a_1a_2 \in C_3(\langle a_2A_2\rangle) \vdash a_1a_2 \in C_4(\langle a_3A_3\rangle) \\
&\ \ \vdots \\
&\Longrightarrow a_1a_2 \dots a_{n-1} \in C_{2(n-1)-1}(\langle a_{n-1}A_{n-1}\rangle) \\
&\qquad\qquad \vdash a_1a_2 \dots a_{n-1} \in C_{2(n-1)}(\langle a_n\rangle) \\
&\Longrightarrow a_1a_2 \dots a_{n-1}a_n \in C_{2n-1}(\langle a_n\rangle) \vdash a_1a_2 \dots a_{n-1}a_n \in C_{2n}(\langle\rangle).
\end{aligned}$$

Hence, we obtain the inclusion $L \subseteq L(\mathcal{N})$.

If a word w is generated by the network, then there is a moment t such that $w \in C_{2t+1}(\langle\rangle)$ or $w \in C_{2t}(\langle\rangle)$. If $w \in C_{2t+1}(\langle\rangle)$, then we have also $w \in C_{2t}(\langle\rangle)$ because the output node does not change any word ($M_{\langle\rangle} = \emptyset$). If $w \in C_{2t}(\langle\rangle)$, then $w \in C_{2t-1}(\langle a\rangle)$ for some terminal symbol $a \in T$ because no word remains in the output node during a communication step.

By induction on the time t, we can show that if a word w exists in a node $N_{\langle aA\rangle}$ for $a \in T$ and $A \in N \cup \{\lambda\}$ at a time $2t+1$ for $t \geq 0$ (after an evolutionary and before a communication step), then there exists a permutation v of the word w such that vA is a sentential form of the grammar G. In the beginning of the computation, we have $\{\lambda\} = C_0(\langle aA\rangle)$ for all $a \in T$ and $A \in N \cup \{\lambda\}$ such that $S \to aA$ is a rule in P and $C_0(x) = \emptyset$ for all other nodes N_x. Hence, we have $C_1(\langle aA\rangle) = \{a\}$ for all $a \in T$ and $A \in N \cup \{\lambda\}$ such that aA is a sentential form of the grammar G and $C_1(x) = \emptyset$ for all other nodes N_x. Let w be a word of the set $C_{2t+1}(\langle aA\rangle)$ for a time $t \geq 0$, a terminal symbol $a \in T$, and a symbol $A \in N \cup \{\lambda\}$. Let v be a permutation of the word w such that vA is a sentential form of the grammar G. Then we obtain in the network that w is communicated to any node $N_{\langle bB\rangle}$ such that $A \to bB$ is a rule of P and there the symbol b is inserted somewhere into the word w (thus, $w_1bw_2 \in C_{2t+3}(\langle bB\rangle)$

for any words w_1 and w_2 with $w_1w_2 = w$) and we have in the grammar G the derivation $vA \Longrightarrow vbB$. Hence, for every word w' in $C_{2t+3}(\langle bB\rangle)$ there is a permutation v' of w' such that $v'B$ is a sentential form of the grammar G.

We summarize: If a word w is generated by the network, then there is a moment t such that $w \in C_{2t+1}(\langle a\rangle)$ for some terminal symbol $a \in T$ and then there exists a permutation v of the word w such that v is a sentential form of the grammar G (and since v is a terminal word it is a word of the language $L(G)$). The language $L(G)$ is commutative and therefore not only the word v but also the word w itself belong to this language.

Hence, $L(\mathcal{N}) \subseteq L$ which leads together with the converse inclusion to the equality $L = L(\mathcal{N})$. Thus, we have shown how one can construct a network which has only filters from the family REG_1 for generating a regular commutative language. This yields the inclusion $COMM \subseteq \mathcal{E}(REG_1)$.

The language $L = \{ab\}$ is finite but not commutative. According to Corollary 1 and Theorem 9, we have $L \in \mathcal{E}(REG_1)$. Thus, we obtain $L \in \mathcal{E}(REG_1) \setminus COMM$ which yields the proper inclusion $COMM \subset \mathcal{E}(REG_1)$. □

Not only regular commutative languages can be generated by networks with filters from the family REG_1 but also arbitrary semi-linear commutative languages.

Theorem 16 *Let L be a semi-linear language. Then $Comm(L) \in \mathcal{E}(REG_1)$.*

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., $\psi(L(G)) = \psi(L)$ (see [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 filters from the family REG_1 only that generates the language $Comm(L(G))$ can be constructed analogously to the construction in the proof of Theorem 15. □

Finally, we give some incomparability results.

Theorem 17 *The language families NIL and COMM are both incomparable to the family $\mathcal{E}(FIN)$.*

Proof According to Lemma 4, the language

$$L = \{a\} \cup \{a^n | n \geq 3\}$$

cannot be generated by a network with finite filters only. Since the language L is nilpotent and commutative, we obtain $NIL \not\subseteq \mathcal{E}(FIN)$ and $COMM \not\subseteq \mathcal{E}(FIN)$.

Let V be the alphabet $\{a, b, c\}$. The language

$$L = \{c^k ac^m bc^n | k \geq 0, m \geq 0, n \geq 0\}$$

is neither nilpotent nor commutative but is generated by the NEP $\mathcal{N} = (V, N_1, \emptyset, 1)$ with the node $N_1 = (\{\lambda \to c\}, \{ab\}, \emptyset, \emptyset)$. Hence, we also obtain $\mathcal{E}(FIN) \not\subseteq NIL$ and $\mathcal{E}(FIN) \not\subseteq COMM$.

Theorem 18 *The language families REG and CF are both incomparable to $\mathcal{E}(MON)$.*

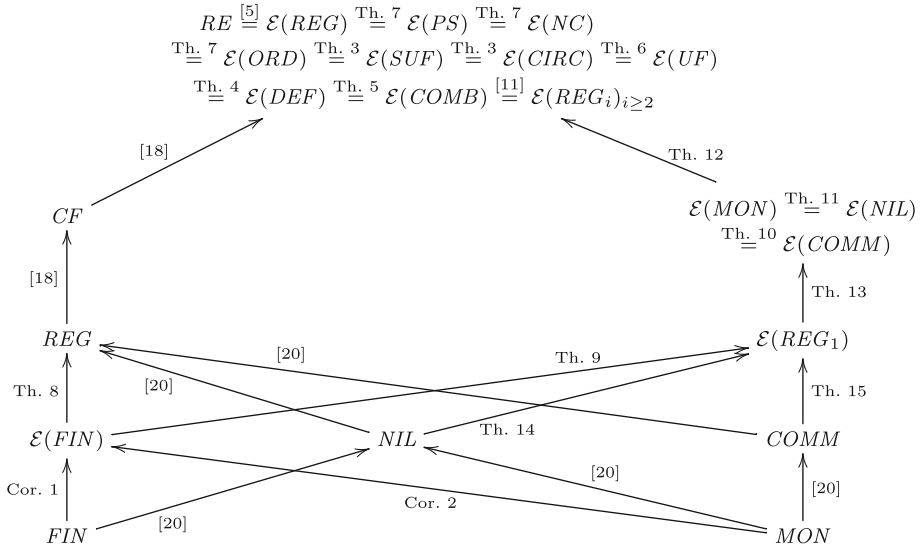


Fig. 2 Hierarchy of the language families generated by networks of evolutionary processors with filters of a certain subregular set

Proof According to Lemma 8, the regular language

$$L = \{wb | w \in \{a, b\}^*\}$$

does not belong to the family $\mathcal{E}(MON)$.

According to Lemma 10, the family $\mathcal{E}(MON)$ contains a non-context-free language. □

As shown in [11], both language families *REG* and *CF* are also incomparable to the family $\mathcal{E}(REG_1)$.

7 Conclusion

If we combine all the results of the preceding sections and [11], we get the diagram presented in Fig. 2.

Theorem 19 *The relations shown in Fig. 2 hold.*

In Fig. 2, an arrow from a language family X to a language family Y stands for the proper inclusion $X \subset Y$. If two families X and Y are not connected by a directed path, then the families are incomparable. A label at an edge or equality sign indicates where the inclusion or equality is shown.

Acknowledgments Florin Manea's work was partially supported by the *Alexander von Humboldt Foundation*, through a two-years Research Fellowship at the Otto-von-Guericke University Magdeburg, Germany, that ended in June 2011 and a subsequent Return Fellowship at the University of Bucharest, Romania, between July 2011 and October 2011.

References

1. Alhazov, A., Dassow, J., Martín-Vide, C., Rogozhin, Y., Truthe, B.: On networks of evolutionary processors with nodes of two types. Fundamenta Informaticae **91**, 1–15 (2009)
2. Bordihn, H., Holzer, M., Kutrib, M.: Determination of finite automata accepting subregular languages. Theor. Comput. Sci. **410**, 3209–3222 (2009)
3. Brzozowski, J., Jirásková, G., Li, B.: Quotient Complexity of Ideal Languages. In: LATIN 2010, LNCS, vol. 6034, pp. 208–221. Springer, Berlin (2010)
4. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: 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, pp. 621–628. Springer, Berlin (2001)
5. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Informatica **39**(6–7), 517–529 (2003)
6. Csuhaj-Varjú, E., Salomaa, A.: Networks of Parallel Language Processors. In: New Trends in Formal Languages—Control, Cooperation, and Combinatorics, LNCS, vol. 1218, pp. 299–318. Springer, Berlin (1997)
7. Dassow, J.: Subregularly controlled derivations: the context-free case. Rostock. Math. Kolloq. **34**, 61–70 (1988)
8. Dassow, J.: 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, pp. 27–37. University of Szeged (2009)
9. Dassow, J., Hornig, H.: Conditional Grammars with Subregular Conditions. In: Proceedings of the International Confernece Words, Languages and Combinatorics II, pp. 71–86. World Scientific Singapore (1994)
10. Dassow, J., Stiebe, R., Truthe, B.: Generative capacity of subregularly tree controlled grammars. Int. J. Found. Comput. Sci. **21**, 723–740 (2010)
11. Dassow, J., Truthe, B.: On networks of evolutionary processors with filters accepted by two-state-automata. Fundamenta Informaticae **112**(2–3), 157–170 (2011)
12. Han, Y.S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theor. Comput. Sci. **410**(27–29), 2537–2548 (2009)
13. Han, Y.S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-suffix-free regular languages. Fundamenta Informaticae **90**(1–2), 93–106 (2009)
14. Havel, I.M.: The theory of regular events II. Kybernetika **5**(6), 520–544 (1969)
15. Holzer, M., Jakobi, S., Kutrib, M.: The magic number problem for subregular language families. In: Proceedings of 12th International Workshop Descriptional Complexity of Formal Systems, pp. 135–146. University of Saskatchewan, Saskatoon (2010)
16. Jirásková, G., Masopust, T.: Complexity in union-free languages. In: Developments in Language Theory, LNCS, vol. 6224, pp. 255–266. Springer, Berlin (2010)
17. Martín-Vide, C., Mitrana, V.: Networks of evolutionary processors: results and perspectives. In: Molecular Computational Models: Unconventional Approaches, pp. 78–114 (2005)
18. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages. Springer, Berlin (1997)
19. Salomaa, A.: Formal Languages. Springer, Berlin (1978)
20. Wiedemann, B.: Vergleich der Leistungsfähigkeit endlicher determinierter Automaten. Universität Rostock, Diplomarbeit (1978)