




A Survey on Computationally Complete Accepting and Generating Networks of Evolutionary Processors

Bianca Truthe^(✉) 

Institut Für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
bianca.truthe@informatik.uni-giessen.de

Abstract. In this survey, we discuss accepting and generating networks of evolutionary processors in their various characteristics as presented in the literature over the years. We show several research directions with respect to reducing the resources needed for still being computationally complete and gather results obtained in these areas so far.

Keywords: Network · Evolutionary Processor · Computational Power

1 Introduction

Based on the idea of processing languages in a distributed and parallel manner by a system of simple agents, several models of language generating or accepting devices have been developed (for instance, grammar systems, evolutionary systems, networks of language processors, networks of splicing systems, networks of Watson–Crick D0L systems). Here, we focus on networks of evolutionary processors.

Starting from networks of language processors which have been introduced in [6] by E. CSUHAJ-VARJÚ and A. SALOMAA, networks of evolutionary processors have been developed in [4] by J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, and J. M. SEMPERE inspired by biological processes.

Such a network can be considered as a graph where the nodes represent processors which apply production rules to the words they contain and the edges are considered as communication channels for exchanging words between processors.

The computation consists of alternating derivation (evolutionary) and communication steps. In an evolutionary step, any node derives from its language all possible words according to its production rules as its new language (any word is assumed to exist in an arbitrary number such that there are enough words for the application of rules; only one rule is applied in one step at one place at most; if no rule is applicable, then the word itself will survive this derivation step, otherwise the original word will not exist anymore after the derivation). The allowed production rules are that one letter is substituted by a letter, a letter is inserted, or a letter is deleted; the nodes are then called substitution

nodes, insertion nodes, or deletion nodes, respectively. In a communication step, any node sends copies of those words to other nodes which satisfy an output condition given as a regular language (called the output filter) and any node adopts (copies of) words sent by the other nodes if the words satisfy an input condition also given by a regular language (called the input filter). Words not passing an output filter remain in the node for the next derivation step; words which have left node but do not pass an input filter to enter some node get lost (disappear from the network).

In the meantime, also other variants have been introduced and investigated, e. g., networks where the filters belong to edges not nodes (e. g., [19]) or networks where the filtering is realized by polarization (e. g., [17]).

Networks of evolutionary processors can be defined as language generating or language accepting devices. In case of a generating device, the processors start working with finite sets of axioms and all words which are in a designated processor at some time form the generated language. In case of an accepting device, input words are accepted if there is a computation which leads to a word in a designated processor.

Early results on generating networks of evolutionary processors can be found, e. g., in [4, 5, 23]. In [11] and [1], the generative capacity of networks of evolutionary processors was investigated where at most two types of rules occur. In [7], the generative capacity of networks of evolutionary processors was investigated for cases that all filters belong to a certain subfamily of the set of all regular languages. In [27], networks of evolutionary processors were investigated where the filters are restricted by bounded resources, namely the number of non-terminal symbols or the number of production rules which are necessary for generating the languages or the number of states of a minimal deterministic finite automaton over an arbitrary alphabet which are necessary for accepting the filters. In [15], the use of codes and ideals as filters was studied. In [14], the hierarchies of the language classes obtained before were merged.

Accepting networks of evolutionary processors were introduced in [22]. Further results, especially on accepting networks where the filters belong to certain subclasses of the family of the regular languages, were published in [8] and [20]. In [28], accepting networks of evolutionary processors were investigated where the filters are restricted by bounded resources (number of non-terminal symbols, number of production rules necessary for generating the languages or number of states of a minimal deterministic finite automaton over an arbitrary alphabet necessary for accepting the filters). In [16], the use of codes and ideals as filters was studied and compared to the impact of other filters.

In the present paper, we give an overview about classes of generating or accepting networks of evolutionary processors which generate or accept all recursively enumerable languages.

2 Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see, e. g., [25]). and recall here only 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 cardinality of a set A is denoted by $|A|$.

A phrase structure grammar is a quadruple $G = (N, T, P, S)$ where N is a finite set of non-terminal symbols, T is a finite set of terminal symbols, P is a finite set of production rules which are written as $\alpha \rightarrow \beta$ with $\alpha \in (N \cup T)^* \setminus T^*$ and $\beta \in (N \cup T)^*$, and $S \in N$ is the axiom. A grammar is right-linear if, for any rule $\alpha \rightarrow \beta$, the left-hand side α consists of a non-terminal symbol only and the right-hand side β contains at most one non-terminal symbol and this is at the right end of the word: $\alpha \in N$ and $\beta \in T^* \cup T^*N$. A special case of right-linearity is regularity where each rule contains exactly one terminal symbol (with the only possible exception $S \rightarrow \lambda$). Let $G = (N, T, P, S)$ be a grammar. A word $u \in (N \cup T)^*$ is derived in one step to a word $v \in (N \cup T)^*$ by the grammar G , written as $u \Rightarrow v$, if there are a rule $\alpha \rightarrow \beta \in P$ and two subwords x and y of u such that $u = x\alpha y$ and $y = x\beta y$. By \Rightarrow^* , we denote the reflexive and transitive closure of the derivation relation \Rightarrow . The language $L(G)$ generated by the grammar G is the set of all words which consist of terminal symbols and which are derivable from the axiom S :

$$L(G) = \{ w \mid w \in T^* \text{ and } S \Rightarrow^* w \}.$$

Regular and right-linear grammars generate the same family of languages (the regular languages). Therefore, also right-linear grammars are often called regular. In the context of descriptonal complexity, when the number of non-terminal symbols or the number of production rules which are necessary for generating a language are considered then there is a difference whether a language is generated by means of regular or right-linear rules. We use in this paper right-linear grammars.

By *REG* and *RE*, we denote the families of languages generated by regular and arbitrary phrase structure grammars, respectively.

A finite automaton is a quintuple $\mathcal{A} = (V, Z, z_0, F, \delta)$ where V is an alphabet called the input alphabet, Z is a non-empty finite set of elements which are called states, $z_0 \in Z$ is the so-called start state, $F \subseteq Z$ is the set of accepting states, and $\delta : Z \times V \rightarrow \mathcal{P}(Z)$ is a mapping which is also called the transition function where $\mathcal{P}(Z)$ denotes the power set of Z (the set of all subsets of Z). A finite automaton is called deterministic if every set $\delta(z, a)$ for $z \in Z$ and $a \in V$ is a singleton set.

The transition function δ can be extended to a function $\delta^* : Z \times V^* \rightarrow \mathcal{P}(Z)$ where $\delta^*(z, \lambda) = \{z\}$ and

$$\delta^*(z, va) = \bigcup_{z' \in \delta^*(z, v)} \delta(z', a).$$

We will use the same symbol δ in both the original and extended version of the transition function.

Let $\mathcal{A} = (V, Z, z_0, F, \delta)$ be a finite automaton. A word w is accepted by the finite automaton \mathcal{A} if and only if the automaton has reached an accepting state after reading the input word w :

$$L(A) = \{ w \mid \delta(z_0, w) \cap F \neq \emptyset \}.$$

The family of the languages accepted by finite automata is also the family of the regular languages.

In the sequel, let V be an alphabet. For a language L over V , we set

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

We consider the following restrictions for regular languages (which yield so-called subregular families of languages). Let L be a language and $V = \text{alph}(L)$ the minimal alphabet of L . We say that the language L , with respect to the alphabet V , is

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

Among the commutative, circular, suffix-closed, non-counting, and power-separating languages, we consider only those which are also regular.

By *FIN*, *MON*, *COMB*, *DEF*, *NIL*, *COMM*, *CIRC*, *SUF*, *NC*, *PS*, *ORD*, and *UF*, we denote the families of all finite, monoidal, combinational, definite, nilpotent, regular commutative, regular circular, regular suffix-closed, regular non-counting, regular power-separating, ordered, and union-free languages, respectively.

In several papers, also regular languages have been considered which are based on some kind of random context: A language over an alphabet V is defined by two subsets $P \subseteq V$ and $F \subseteq V$ and a mode (strong or weak) as the set of all

words $w \in (V \setminus F)^*$ which contain every symbol of the permitting set P (in the strong mode) or at least one symbol (in the weak mode, if P is not empty) but, in any case, no symbol of the forbidding set F .

Additionally, families of languages are considered which are defined by bounding the resources which are necessary for accepting or generating these languages.

Let RLG be the set of all right-linear grammars and DFA the set of all deterministic finite automata. Further, let

$$G = (N, T, P, S) \in RLG \quad \text{and} \quad \mathcal{A} = (V, Z, z_0, F, \delta) \in DFA.$$

Then we define the following measures of descriptonal complexity:

$$Var(G) = |N|, \quad Prod(G) = |P|, \quad State(\mathcal{A}) = |Z|.$$

For these complexity measures, we define the following families of languages (we abbreviate the measure Var by V , the measure $Prod$ by P , and the measure $State$ by Z):

$$\begin{aligned} RL_n^V &= \{ L \mid \exists G \in RLG : L = L(G) \text{ and } Var(G) \leq n \}, \\ RL_n^P &= \{ L \mid \exists G \in RLG : L = L(G) \text{ and } Prod(G) \leq n \}, \\ REG_n^Z &= \{ L \mid \exists \mathcal{A} \in DFA : L = L(\mathcal{A}) \text{ and } State(\mathcal{A}) \leq n \}. \end{aligned}$$

We now introduce the notion of an ideal in V^* from the theory of rings and semigroups.

A non-empty language $L \subseteq V^*$ is called a right (left) ideal if and only if, for any word $v \in V^*$ and any word $u \in L$, we have $uv \in L$ ($vu \in L$, respectively). It is easy to see that the language L is a right (left) ideal if and only if there is a language L' such that $L = L'V^*$ ($L = V^*L'$, respectively).

We now present some notions from coding theory, especially some special codes. For details, we refer to [18] and [26].

For a word $x \in V^*$, let

$$E(x) = \{ y \mid y \in V^+, \quad yv' = x \text{ for some } v, v' \in V^* \},$$

(i. e., $E(x)$ is the set of all non-empty subwords of x).

A language $L \subseteq V^*$ is called

- a code if and only if, for any numbers $n \geq 1$, $m \geq 1$, and words

$$x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m \in L$$

such that

$$x_1x_2 \dots x_n = y_1y_2 \dots y_m,$$

we have the equalities $n = m$ and $x_i = y_i$ for $1 \leq i \leq n$ (i. e., a word of L^* has a unique decomposition into code words).

– a solid code if and only if, for any numbers $n \geq 1$, $m \geq 1$, words

$$x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m \in L,$$

and words

$$v_1, v_2, \dots, v_{n+1}, w_1, w_2, \dots, w_{m+1}$$

with $E(v_i) \cap L = \emptyset$ for $1 \leq i \leq n+1$, and $E(w_j) \cap L = \emptyset$ for $1 \leq j \leq m+1$ such that

$$v_1 x_1 v_2 x_2 \dots v_n x_n v_{n+1} = w_1 y_1 w_2 y_2 \dots w_m y_m w_{m+1},$$

we have $n = m$, $x_i = y_i$ for $1 \leq i \leq n$, and $v_j = w_j$ for $1 \leq j \leq n+1$;

- uniform if and only if $L \subseteq V^n$ for some $n \geq 1$ (all words have the same length);
- prefix if and only if, for any words $u \in L$ and $v \in V^*$ such that $uv \in L$, we have $v = \lambda$ (i. e., any proper prefix of a word in L is not in L);
- suffix if and only if, for any words $u \in L$ and $v \in V^*$ such that $vu \in L$, we have $v = \lambda$ (i. e., any proper suffix of a word in L is not in L);
- bifix if and only if it is prefix as well as suffix;
- infix if and only if, for any $u \in L$, and $v, v' \in V^*$ such that $vuv' \in L$, we have $v = v' = \lambda$ (i. e., any proper subword of a word in L is not in L).

Note that uniform, prefix, suffix, bifix, and infix languages are codes.

A code $L \subseteq V^*$ is called

- outfix if and only if, for any words $u \in V^*$ and $v, v' \in V^*$ such that $vv' \in L$ and $vuv' \in L$, we have $u = \lambda$;
- reflective if and only if, for any words $u, v \in V^*$ such that $uv \in L$, we have $vu \in L$.

By *rId*, *lId*, *C*, *SC*, *PfC*, *SfC*, *BfC*, *IfC*, *OfC*, *RC*, and *UC*, we denote the families of regular right ideals, regular left ideals, regular codes, regular solid codes, regular prefix codes, regular suffix codes, regular bifix codes, regular infix codes, regular outfix codes, regular reflective codes and uniform codes, respectively.

We now present networks of evolutionary processors. We call a production $\alpha \rightarrow \beta$ a substitution if $|\alpha| = |\beta| = 1$ and 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 $\alpha \rightarrow \beta$ such that $v = x\alpha y$ and $w = x\beta y$.

We introduce insertion as a counterpart of deletion. We write $\lambda \rightarrow a$, where a is a letter. The application of an insertion $\lambda \rightarrow a$ 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 .

If the applied rule p should be mentioned, we write $v \Longrightarrow_p w$. For a set P of rules, we write $v \Longrightarrow_P w$ if and only if $v \Longrightarrow_p w$ for some rule $p \in P$. The reflexive and transitive closure of the relation is denoted by \Longrightarrow_P^* : we write $x \Longrightarrow_P^* y$ if there are rules p_1, \dots, p_n ($n \geq 0$) in P and words w_0, \dots, w_n

such that $x = w_0$, $w_i \xRightarrow{p_{i+1}} w_{i+1}$ for $i = 0, \dots, n-1$, and $w_n = y$. If at least one rule is applied, we write $x \xRightarrow{p}^+ y$. For an alphabet V , we denote by SUB_V , DEL_V , and INS_V the sets of all substitution, deletion, or insertion rules, respectively, over the alphabet V .

We first define networks of evolutionary processors for generating languages. In the literature, they are abbreviated as NEPs but in order to better distinguish them from accepting networks, for say here GNEPs.

Definition 1.

1. A generating network of evolutionary processors (of size n) is an $(n+3)$ -tuple $\mathcal{N} = (V, N_1, N_2, \dots, N_n, E, n_o)$ where
 - V is a finite alphabet (the working alphabet of the network),
 - for $1 \leq i \leq n$, there is a processor $N_i = (M_i, A_i, I_i, O_i)$ where
 - M_i is a set of evolution rules of a certain type, i. e., $M_i \subseteq SUB_V$ or $M_i \subseteq DEL_V$ or $M_i \subseteq INS_V$,
 - A_i is a finite subset of V^* (the language of axioms, from where the processing starts in this processor),
 - I_i and O_i are regular sets over V (called the input and output filter, respectively),
 - E is a subset of $\{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$, and
 - n_o is a natural number from the set $\{1, 2, \dots, n\}$; the processor N_{n_o} is called the output node of the network.
2. A configuration C of \mathcal{N} is an n -tuple $C = (C(1), C(2), \dots, C(n))$ where $C(i)$ is a subset of V^* for $1 \leq i \leq n$.
3. Let $C = (C(1), C(2), \dots, C(n))$ and $C' = (C'(1), C'(2), \dots, C'(n))$ be two configurations of \mathcal{N} . We say that C derives C' in one
 - evolution step (written as $C \Longrightarrow C'$) if, for $1 \leq i \leq n$, the set $C'(i)$ 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 $v \xRightarrow{p} w$ 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 \mathcal{N} is a sequence of configurations

$$C_t = (C_t(1), C_t(2), \dots, C_t(n)), \text{ for } t \geq 0,$$

such that

- $C_0 = (A_1, A_2, \dots, A_n)$,
- for any $t \geq 0$, C_{2t} derives C_{2t+1} in one evolution step:

$$C_{2t} \Longrightarrow C_{2t+1},$$

– for any $t \geq 0$, C_{2t+1} derives C_{2t+2} in one communication step:

$$C_{2t+1} \vdash C_{2t+2}.$$

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

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

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

Before we define accepting networks, we briefly describe how such a network works. The underlying structure of a GNEP is a graph consisting of some, say n , nodes N_1, N_2, \dots, N_n (called processors) and 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 M_i of evolution rules (also called mutation rules), a set A_i of start words, an input filter I_i and an output filter O_i . We say that N_i is a substitution node or a deletion node or an insertion node if $M_i \subseteq \{a \rightarrow b \mid a, b \in V\}$ or $M_i \subseteq \{a \rightarrow \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \rightarrow b \mid b \in V\}$, respectively. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively. With any node N_i and any time moment $t \geq 0$, we associate a set $C_t(i)$ of words (the words contained in the node N_i 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 (each word occurs in a sufficiently large number, each rule is applied at an arbitrary possible position in a word, but only one rule at one place is applied in one step, if no rule can be applied to a word, then it remains unchanged, if a rule can be applied, then the original word will be consumed). In a communication step, any processor N_i sends out all words $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i , i. e., the processor N_i gets in addition all words of $(C_t(k) \cap O_k) \cap I_i$. The computation starts with an evolutionary step and then communication steps and evolutionary steps are alternately performed. The language generated consists of all words which are in the output node N_{n_o} at some moment t with $t \geq 0$.

We now define networks of evolutionary processors for accepting languages (ANEPs for short).

Definition 2. 1. An accepting network of evolutionary processors (of size n) is a $(n + 5)$ -tuple $\mathcal{N} = (V, U, N_1, N_2, \dots, N_n, E, n_i, n_o)$ where

- V is a finite alphabet, called the input alphabet of the network,
- U is a finite alphabet with $V \subseteq U$, called the working alphabet of the network,
- $N_i = (M_i, I_i, O_i)$ for $1 \leq i \leq n$ are the processors where

- M_i is a set of rules of a certain type: $M_i \subseteq SUB_U$ or $M_i \subseteq DEL_U$ or $M_i \subseteq INS_U$,
 - I_i and O_i are regular sets over U (called the input and output filter, respectively),
- E is a subset of $\{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$, and
 - n_i and n_o are two natural numbers from the set $\{1, 2, \dots, n\}$; the processor N_{n_i} is called the input node and N_{n_o} the output node of the network.
2. Configuration, evolutionary steps, and communication steps are defined as for generating networks.
- The computation of an evolutionary network \mathcal{N} on an input word $w \in V^*$ is a sequence of configurations $C_t^w = (C_t^w(1), C_t^w(2), \dots, C_t^w(n))$ with $t \geq 0$, such that
- $C_0^w(n_i) = \{w\}$ and $C_0^w(j) = \emptyset$ for $j \in \{1, \dots, n\} \setminus \{n_i\}$,
 - for any $t \geq 0$, C_{2t}^w derives C_{2t+1}^w in one evolutionary step,
 - for any $t \geq 0$, C_{2t+1}^w derives C_{2t+2}^w in one communication step.
- The computation of an evolutionary network \mathcal{N} on an input word $w \in V^*$ is said to be accepting if there exists a step $t \geq 0$ in which the component $C_t^w(n_o)$ of the configuration representing the content of the output node is not empty.
3. The language $L(\mathcal{N})$ accepted by \mathcal{N} is defined as

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

An accepting network of evolutionary processors works in the same manner as a generating one. The differences are, that, in an ANEP, only one processor has a word in the beginning (called the input word) and that a word in the output processor in an ANEP indicates the acceptance of the input word whereas in a GNEP it belongs to the generated language.

For a language class X , we denote the class of languages accepted by networks of evolutionary processors where all filters belong to the class X by $\mathcal{A}(X)$ and the class of languages generated by networks of evolutionary processors where all filters belong to the class X by $\mathcal{E}(X)$. We consider the filters independently from the environment. A filter language belongs to some family X if it belongs to it with respect to its smallest alphabet, not necessarily to the the alphabet of all letters which might occur in the node or even in the entire network. A word passes a filter if it is an element of the language representing the filter otherwise it does not pass the filter.

In the literature, also other definitions have been used. In the first paper [4] and subsequent ones, the underlying graph was a complete one and the insertion and deletion rules were allowed only to be applied at the end of a word (substitution was allowed everywhere). Later, the so-called hybrid networks of evolutionary processors were introduced (generating HNEPs in [24] and accepting HNEPs in [22]) where each processor is equipped also with a position where its rules must be applied (either all to the left end of a word or all to the right end of a word, or at an arbitrary place). Another difference regards what happens with words to which not all rules or even no rule can be applied in an evolutionary step. Consider, for instance, the word a . If it is in a processor with

the deletion rule $a \rightarrow \lambda$ and only this rule, then the word is derived to λ . If further a deletion rule $b \rightarrow \lambda$ is present, then this rule does not change the word. Hence, the words obtained are a (by the b -rule) and λ (by the a -rule). In [2], a variant has been introduced, called obligatory HNEPs, where only the evolved words belong to the derived language (in this example, only λ because a together with the b -rule yields the empty set). Hence, in OHNEPs, the original words disappear in an evolutionary step (their evolution is obligatory). In [11], a weaker variant has been introduced where in an evolutionary step a word survives if no rule can be applied to it but if a rule can be applied, then the original word will not be present any longer (after all rules have been applied at every possible position). In all these cases, computational completeness was obtained (for every recursively enumerable language, there is a network of evolutionary processors generating or accepting it, no matter what details have been used).

In the sequel, we refer here to papers where the same definition as above has been used. Some results appeared already earlier in other publications but based on another definition.

The following theorem is known (see [7] and [20]).

Theorem 1 ([7], [20]). *We have $\mathcal{E}(REG) = \mathcal{A}(REG) = RE$.*

As usual for powerful models, one asks what can be reduced to what extend without leaving this power.

3 Restrictions Without Decreasing Computational Power

The size of a system is always interesting since smaller systems need fewer resources (space, time for the construction). Regarding resources, the size is not only the number of components but also the sum of the sizes of the components. So, there are many aspects to consider.

3.1 Number of Processors

While in the introductory paper to GNEPs [4] the size of the constructed network was still unbounded (depending on the size of the problem), it has been shown in [5] that for any recursively enumerable language, there is a GNEP with five processors generating the language. Already between these two papers, the definition changed (in the first one, deletion and insertion rules were allowed to be applied only at the ends of a word whereas in the second paper, they could be applied everywhere in a word). In [3], the number was reduced to four (if one takes only terminal words of the generated language (the intersection of the language with a monoid), then even three processors are sufficient). In the same paper, it was shown that, with again another definition (any type of rule is allowed in any processor), any recursively enumerable language can be generated by a network with two processors (even one with intersection with a monoid). Back to processors specialized in one type of rules only, the network with four/three nodes used each type of rules. Hence, a natural question was

whether the number of rule types could be reduced. This question was investigated in [1] where it was shown that any recursively enumerable language can be generated by a network which has one node for insertion rules, one node for deletion rules, and one node without rules (which is for collecting the terminal words) only. If one allows intersection with a monoid, the node without rules can be omitted. This yielded also an optimal result for the total number of processors needed to be still computational complete.

Regarding accepting NEPs, any recursively enumerable language can be accepted by a network with three nodes: one substitution or deletion node, one insertion node and one without rules [10].

3.2 Number of Production Rule Types

As mentioned above, in [1], it was shown that any recursively enumerable language can be generated by a network which has one node for insertion rules one node for deletion rules, and one node without rules. In the same paper, it was shown that networks with an arbitrary number of deletion and substitution nodes only generate finite languages (and, for each finite language, one deletion node or one substitution node is sufficient) and networks with an arbitrary number of insertion and substitution nodes only generate context-sensitive languages (and, up to an intersection with a monoid, every context-sensitive language can be generated by a network with one substitution node and one insertion node). So, one type alone would not suffice for computational completeness.

Also ANEPs do not need all three types of rules for being computationally complete; two types are sufficient: one node with substitution or deletion rules, one node with insertion rules, and one node without rules [10]. In [9], it has been shown that networks with only substitution and deletion rules accept context-sensitive languages only (and that every context-sensitive language is accepted by such a network). So, insertion rules are essential for being computational complete.

3.3 Restrictions to the Filters

In several papers, generating or accepting networks haven been investigated where the filters are not arbitrary regular languages but special ones like finite, union-free, suffix-closed languages or languages which are codes or languages which can be generated with a certain number of variables or rules in their generating right-linear grammars or accepted by deterministic finite automata with a certain number of states. Also networks where the filters are given by permitting and forbidding sets (random-context filters) belong to this area.

For generating NEPs, we refer to the papers [7] and [14] for overviews about hierarchies where the language classes obtained are put into set theoretic relations.

In [7], it was shown 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 filters are used which 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.

In [12] and [13], generating networks have been investigated where the minimal deterministic finite automata accepting the filter languages are restricted with respect to the number of their states. It was shown that if the number of states is bounded by two, then every recursively enumerable language can be generated by such a network. If the number of states is bounded by one, then not all regular languages but non-context-free languages can be generated.

In [27], other restrictions on the resources needed for the filters, namely the number of variables or production rules which are needed by a right-linear grammar generating a filter were investigated and set into relation with the restrictions of the papers [12] and [13].

In [15], the generative capacity of GNEPs has been studied where the filters are codes (arbitrary and special ones) or ideals. The hierarchy of the generated language classes obtained there has been merged which that one from [27] in the paper [14].

If the filters are all taken from one of the classes PS , NC , ORD , DEF , $CIRC$, UF , SUF , $COMB$, Id , rId , $(RL_i^V)_{i \geq 1}$, $(REG_i^Z)_{i \geq 2}$, then any recursively enumerable language can be generated. With filters from the other considered classes, the GNEPs are less powerful.

Theorem 2 ([7, 14, 15]). *We have*

$$\begin{aligned} RE &= \mathcal{E}(PS) = \mathcal{E}(NC) = \mathcal{E}(ORD) = \mathcal{E}(DEF) = \mathcal{E}(CIRC) \\ &= \mathcal{E}(UF) = \mathcal{E}(SUF) = \mathcal{E}(COMB) \\ &= \mathcal{E}(Id) = \mathcal{E}(rId) = \mathcal{E}((RL_i^V)_{i \geq 1}) = \mathcal{E}((REG_i^Z)_{i \geq 2}). \end{aligned}$$

For accepting NEPs, we refer to the papers [21] and [16] for overviews about hierarchies where the language classes obtained are put into set theoretic relations.

In [21], it was shown that the use of filters from the class of non-counting, ordered, power-separating, suffix-closed regular, union-free, definite and combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can accept all the recursively enumerable languages. On the other hand, by using filters that are only finite languages, monoids, nilpotent languages, commutative regular languages, or circular regular languages, one cannot generate all recursively enumerable languages. Hence, for such filters, the only difference between generating and accepting NEPs is the class $CIRC$. Networks with circular filters only can still generate every recursively enumerable language whereas they cannot accept every such language.

In [28], the aforementioned restrictions on the resources needed for the filters, namely the number of variables or production rules which are needed by a right-linear grammar generating a filter and the number of states which are needed by a minimal deterministic finite automaton accepting a filter were investigated and set into relation with the restrictions of the paper [21]. Here, the results for GNEPs and ANEPs coincide: In both cases, the classes $(RL_i^V)_{i \geq 1}$ and $(REG_i^Z)_{i \geq 2}$ yield computationally complete networks whereas filter restrictions to the classes REG_1^Z or RL_i^P for any number $i \geq 1$ are less powerful.

In [16], the generative capacity of ANEPs has been studied where the filters are codes (arbitrary and special ones) or ideals. The hierarchy of the generated language classes obtained there has been merged which that one from [28]. For ideals, the results for GNEPs and ANEPs coincide: In both cases, the classes Id and rId yield computationally complete networks. For codes, the situation is different: Whereas filter restrictions to the classes C , PfC , SfC , BfC , IfC , or SC do not decrease the computational power of accepting networks (such filters are equally powerful as arbitrary regular languages), generating networks with filters from such classes are less powerful.

Summarizing, for accepting networks with filters from subregular language classes, we have the following results.

Theorem 3 ([16, 21]). *We have*

$$\begin{aligned} RE &= \mathcal{A}(PS) = \mathcal{A}(NC) = \mathcal{A}(ORD) = \mathcal{A}(DEF) \\ &= \mathcal{A}(UF) = \mathcal{A}(SUF) = \mathcal{A}(COMB) \\ &= \mathcal{A}(Id) = \mathcal{A}(rId) = \mathcal{A}((RL_i^V)_{i \geq 1}) = \mathcal{A}((REG_i^Z)_{i \geq 2}) \\ &= \mathcal{A}(C) = \mathcal{A}(PfC) = \mathcal{A}(SfC) = \mathcal{A}(BfC) = \mathcal{A}(IfC) = \mathcal{A}(SC). \end{aligned}$$

4 Further Research

Various kinds of generating and accepting networks of evolutionary processors have been developed. Every kind has its own motivation (for instance, biological background). However, from the theoretical point of view, it would be interesting to close the gaps such that networks and their properties are better comparable.

Further, there are still open questions about the computational power of certain networks (e. g., networks with insertion processors only or where the filters belong to a class such that not every recursively enumerable language can be generated or accepted by such a network). Also other restrictions could be considered (other subregular language classes for the filters) or restrictions regarding other resources or combinations thereof.

References

1. Alhazov, A., Dassow, J., Martín-Vide, C., Rogozhin, Y., Truthe, B.: On networks of evolutionary processors with nodes of two types. *Fundam. Informaticae* **91**, 1–15 (2009)

2. Alhazov, A., Enguix, G.B., Rogozhin, Y.: Obligatory hybrid networks of evolutionary processors. In: Filipe, J., Fred, A.L.N., Sharp, B. (eds.) ICAART 2009 - Proceedings of the International Conference on Agents and Artificial Intelligence, Porto, Portugal, 19–21 January 2009, pp. 613–618. INSTICC Press (2009)
3. Alhazov, A., Martín-Vide, C., Rogozhin, Y.: On the number of nodes in universal networks of evolutionary processors. *Acta Informatica* **43**(5), 331–339 (2006). <https://doi.org/10.1007/s00236-006-0024-x>
4. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Solving NP-complete problems with networks of evolutionary processors. In: Mira, J., Prieto, A. (eds.) IWANN 2001. LNCS, vol. 2084, pp. 621–628. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45720-8_74
5. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. *Acta Informatica* **39**(6–7), 517–529 (2003). <https://doi.org/10.1007/s00236-003-0114-y>
6. Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In: Păun, G., Salomaa, A. (eds.) *New Trends in Formal Languages*. LNCS, vol. 1218, pp. 299–318. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62844-4_22
7. Dassow, J., Manea, F., Truthe, B.: Networks of evolutionary processors: the power of subregular filters. *Acta Informatica* **50**(1), 41–75 (2013). <https://doi.org/10.1007/s00236-012-0172-0>
8. Dassow, J., Mitrana, V.: Accepting networks of non-inserting evolutionary processors. In: Petre, I., Rozenberg, G. (eds.) *Proceedings of NCGT 2008 - Workshop on Natural Computing and Graph Transformations*, Leicester, United Kingdom, 8 September 2008, pp. 29–41. University of Leicester (2008)
9. Dassow, J., Mitrana, V.: Accepting networks of non-inserting evolutionary processors. In: Priami, C., Back, R.-J., Petre, I. (eds.) *Transactions on Computational Systems Biology XI*. LNCS, vol. 5750, pp. 187–199. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04186-0_9
10. Dassow, J., Mitrana, V., Truthe, B.: The role of evolutionary operations in accepting hybrid networks of evolutionary processors. *Inf. Comput.* **209**(3), 368–382 (2011)
11. Dassow, J., Truthe, B.: On the power of networks of evolutionary processors. In: Durand-Lose, J., Margenstern, M. (eds.) *MCU 2007*. LNCS, vol. 4664, pp. 158–169. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74593-8_14
12. Dassow, J., Truthe, B.: On networks of evolutionary processors with state limited filters. In: Bordihn, H., Freund, R., Hinze, T., Holzer, M., Kutrib, M., Otto, F. (eds.) *Second Workshop on Non-Classical Models of Automata and Applications (NCMA)*, Jena, Germany, 23–24 August 2010, *Proceedings*. books@ocg.at, vol. 263, pp. 57–70. Österreichische Computer Gesellschaft, Austria (2010)
13. Dassow, J., Truthe, B.: On networks of evolutionary processors with filters accepted by two-state-automata. *Fundam. Informaticae* **112**(2–3), 157–170 (2011)
14. Dassow, J., Truthe, B.: Generating networks of evolutionary processors with resources restricted and structure limited filters. *J. Automata Lang. Comb.* **25**(2–3), 83–113 (2020)
15. Dassow, J., Truthe, B.: Networks with evolutionary processors and ideals and codes as filters. *Int. J. Found. Comput. Sci.* **31**(1), 73–89 (2020)
16. Dassow, J., Truthe, B.: Accepting networks of evolutionary processors with resources restricted and structure limited filters. *RAIRO Theor. Inform. Appl.* **55**, 8 (2021)

17. Freund, R., Rogojin, V., Verlan, S.: Variants of networks of evolutionary processors with polarizations and a small number of processors. *Int. J. Found. Comput. Sci.* **30**(6–7), 1005–1027 (2019)
18. Jürgensen, H., Konstantinidis, S.: Codes¹. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, pp. 511–607. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_8
19. Loos, R., Manea, F., Mitrana, V.: Small universal accepting networks of evolutionary processors with filtered connections. *J. Automata Lang. Comb.* **15**(1–2), 155–174 (2010)
20. Manea, F., Truthe, B.: Accepting networks of evolutionary processors with subregular filters. *Theory of Computing Systems* **55**(1), 84–109 (2014)
21. Manea, F., Truthe, B.: Accepting networks of evolutionary processors with subregular filters. *Theor. Comput. Syst.* **55**(1), 84–109 (2013). <https://doi.org/10.1007/s00224-013-9502-z>
22. Margenstern, M., Mitrana, V., Pérez-Jiménez, M.J.: Accepting hybrid networks of evolutionary processors. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) *DNA 2004*. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005). https://doi.org/10.1007/11493785_21
23. Martín-Vide, C., Mitrana, V.: Networks of evolutionary processors: results and perspectives. In: *Molecular Computational Models: Unconventional Approaches*, pp. 78–114 (2005)
24. Martín-Vide, C., Mitrana, V., Pérez-Jiménez, M.J., Sancho-Caparrini, F.: Hybrid networks of evolutionary processors. In: Cantú-Paz, E. (ed.) *GECCO 2003*. LNCS, vol. 2723, pp. 401–412. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_49
25. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. Springer, Heidelberg (1997). <https://doi.org/10.1007/978-3-642-59136-5>
26. Shyr, H.J.: *Free Monoids and Languages*. Hon Min Book Company, Taichung, Taiwan (1991)
27. Truthe, B.: Networks of evolutionary processors with resources restricted filters. In: Freund, R., Hospodár, M., Jirásková, G., Pighizzini, G. (eds.) *Tenth Workshop on Non-Classical Models of Automata and Applications (NCMA)*, Košice, Slovakia, 21–22 August 2018, Proceedings. *books@ocg.at*, vol. 332, pp. 165–180. Österreichische Computer Gesellschaft (2018)
28. Truthe, B.: Accepting networks of evolutionary processors with resources restricted filters. In: Freund, R., Holzer, M., Sempere, J.M. (eds.) *Eleventh Workshop on Non-Classical Models of Automata and Applications (NCMA)*, Valencia, Spain, 2–3 July 2019, Proceedings. *books@ocg.at*, vol. 336, pp. 187–202. Österreichische Computer Gesellschaft (2019)