Genetic Programming and Evolvable Machines (2022) 23:133–155
https://doi.org/10.1007/s10710-021-09423-7

Generating networks of genetic processors

Marcelino Campos[1] · José M. Sempere[1]

Received: 8 April 2020 / Revised: 6 October 2021 / Accepted: 9 October 2021 /
Published online: 21 October 2021

Abstract

The Networks of Genetic Processors (NGPs) are non-conventional models of computation based on genetic operations over strings, namely mutation and crossover operations as it was established in genetic algorithms. Initially, they have been proposed as acceptor machines which are decision problem solvers. In that case, it has been shown that they are universal computing models equivalent to Turing machines. In this work, we propose NGPs as enumeration devices and we analyze their computational power. First, we define the model and we propose its definition as parallel genetic algorithms. Once the correspondence between the two formalisms has been established, we carry out a study of the generation capacity of the NGPs under the research framework of the theory of formal languages. We investigate the relationships between the number of processors of the model and its generative power. Our results show that the number of processors is important to increase the generative capability of the model up to an upper bound, and that NGPs are universal models of computation if they are formulated as generation devices. This allows us to affirm that parallel genetic algorithms working under certain restrictions can be considered equivalent to Turing machines and, therefore, they are universal models of computation.

Keywords Natural computing · Networks of bio-inspired processors · Parallel genetic algorithms · Formal languages · Descriptive complexity

Mathematics Subject Classification MSC 68Q07 · MSC 68Q19 · MSC 68Q45

This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

José M. Sempere
jsempere@dsic.upv.es

Marcelino Campos
mcampos@dsic.upv.es

[1] Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València, València, Spain

1 Introduction

In the framework of non-conventional computing, new models of computation have been proposed by introducing new operations on data inspired by the nature [11]. In this work, we focus our attention on biologically inspired models of computation. This approach is not new in the history of computing: Artificial Neural Networks and Genetic Algorithms were formulated many years ago by taking into account biological aspects of information processing in nature. Currently, these models can be considered classical models of computation. We are mostly interested in the new models that have been formulated with new operations based at the molecular level (mainly, the DNA recombination and mutation) or cellular level (mainly, by using the structure and organization of the living cell, and the way it processes information).

This work is highly related to the *Networks of Evolutionary Processors* (NEP) [9, 10]. That model was inspired by point mutations and evolutive selection on DNA, in a similar way as in genetic algorithms mutation and INSDEL genome evolution. The *Networks of Splicing Processors* (NSPs) [13] directly use splicing operations over strings instead of point mutation of NEPs. Finally, the *Networks of Genetic Processors* (NGPs) [6], use only substitution mutations together with recombination (crossover), so it is an alternative way of formulating genetic algorithms working in parallel, in a distributed way. All these models can be considered as Networks of Bio-inspired Processors (NBPs) [4], and all these models have been proved to be equivalent to Turing machines, and they have been used to solve NP-complete problems in polynomial time (in several cases, with a constant number of processors) [7, 9, 12–14].

In the last few years, the *Networks of Polarized Evolutionary Processors* (NPEP) have been proposed [1, 5]. In this case, the operations over strings are those proposed in the NEP model, the processors are positively, negatively or neutral polarized and all the strings in the network are numerically valuated. Again, these models have been proved to be computationally complete, and they have been proposed to solve hard problems efficiently.

In this work, we study the computational power of Networks of Genetic Processors as generating devices. It has been proved that this model is equivalent to Turing machines as an accepting device [6]. In addition, it was proved that they are equivalent to parallel genetic algorithms with full migration rates and synchronization, and they can solve hard problems efficiently [7].

The structure of this work is the following: First, we introduce basic concepts on classical language theory and generative grammars. Then, we formally define the Generating Networks of Genetic Processors (GNGP), and we relate them directly to genetic algorithms showing the evidences about why GNGPs can be considered parallel and distributed genetic algorithms with certain restrictions in the migration and selection processes through fitness functions. We propose different network structures to characterize each of the language classes defined in the Chomsky hierarchy, and we define each of the processors involved in the mentioned structures. In addition, we formally establish, through an inductive proof, that the proposed topologies effectively characterize upper limits in the language classes under study. Finally, we discuss our results, and we describe future research on this topic.

2 Basic concepts and notation

In the following, we introduce basic concepts about generative formal grammars and formal language theory [15, 18].

An *alphabet* is a finite non-empty set of elements named *symbols*. A *string* is an ordered finite sequence of symbols of an alphabet. The empty string is denoted by ε and it is defined as the string with no symbols. Given a string w, the length of the string is the number of symbols that it contains and it is denoted by $|w|$ (observe that $|\varepsilon| = 0$). The infinite set of all the strings defined over a given alphabet V is denoted by V^*. Given the alphabet V, the set V^+ is defined as $V^+ = V^* - \{\varepsilon\}$. Given the string $x \in V^*$, $alph(x)$ denotes the minimal subset $W \subseteq V$ such that $x \in W^*$. Given the string $x \in V^*$, the set of segments of x is defined by $seg(x) = \{\beta \in V^* : x = \alpha\beta\gamma$ with $\alpha, \gamma \in V^*\}$. Obviously, given any string $x \in V^*$ the set $alph(x)$ is a subset of $seg(x)$. A *language* defined over an alphabet V is a subset of V^*.

A *grammar* is a tuple $G = (N, T, P, S)$, where N is an alphabet of *auxiliary* symbols, T is an alphabet of *terminal* symbols, with $N \cap T = \emptyset$, $S \in N$ is an *axiom* or *initial* symbol, and P is a finite set of *production rules*. Every production rule is a pair (α, β) (also written as $\alpha \to \beta$), with $\alpha \in (N \cup T)^* N (N \cup T)^*$ and $\beta \in (N \cup T)^*$. Given two strings of terminal and auxiliary symbols, v and w, we say that w can be obtained from v in a direct derivation according to G, and we denote it by $v \underset{G}{\Rightarrow} w$, if $v = v_1 \alpha v_2$, $w = v_1 \beta v_2$ and $\alpha \to \beta \in P$. Observe that the direct derivation is a relation between strings formed by terminal and auxiliary symbols, and we can define the reflexive and transitive closure of $\underset{G}{\Rightarrow}$ as the derivation relation between any pair of strings v and w, that is denoted by $v \overset{*}{\underset{G}{\Rightarrow}} w$, and it is defined iff one of the following conditions hold:

1. $v = w$ (no production rule is applied over v), or,
2. $v \underset{G}{\Rightarrow} u \overset{*}{\underset{G}{\Rightarrow}} w$ (a positive number of rules are applied to obtain w from u).

In addition, we denote i derivation steps by the symbol $\overset{i}{\underset{G}{\Rightarrow}}$.

The language generated by $G = (N, T, P, S)$ is defined as follows

$$L(G) = \{w \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}.$$

The grammars G_1 and G_2 are *equivalent* if $L(G_1) = L(G_2)$, and G_1 is *quasi-equivalent* to G_2 if $L(G_1) = L(G_2) - \{\varepsilon\}$.

The Chomsky hierarchy is a framework to study large formal language classes. It is based on the classification of generative grammars according to the forms of the production rules. It establishes four classes of grammars that we can enumerate as follows:

1. Regular grammars (right linear grammars)
   The productions of the grammar must be in one of the following forms
   - $A \rightarrow aB$, with $A, B \in N$ and $a \in T$
   - $A \rightarrow a$, with $A \in N$ and $a \in T \cup \{\varepsilon\}$
2. Context-free grammars
   The productions of the grammar must be in the form $A \rightarrow \alpha$, with $A \in N$, and $\alpha \in (N \cup T)^*$.
   The Chomsky Normal Form for context-free grammars is defined whenever the productions are in one of the following forms:
   - $A \rightarrow BC$, with $A, B, C \in N$
   - $A \rightarrow a$, with $A \in N$ and $a \in T$

   It is well known that for every context-free grammar there exists a quasi-equivalent grammar in Chomsky Normal Form.
3. Context-sensitive grammars
   The derivations of the grammar are length increasing (with the exception of the derivation of ε). We can establish the Kuroda Normal Form for context-sensitive grammars. It is defined by the following production forms:
   - $A \rightarrow a$, with $A \in N$ and $a \in T$
   - $A \rightarrow B$, with $A, B \in N$
   - $A \rightarrow BC$ with $A, B, C \in N$
   - $AB \rightarrow CD$ with $A, B, C, D \in N$

   In addition, we can add the production rule $S \rightarrow \varepsilon$, whenever S does not appear in the right side of any production rule. In such a case, the grammar can generate the empty string.
4. Phrase structure grammars
   There are no restrictions in the form of the production rules. Nevertheless, we can establish the following production rules, as an extended Kuroda Normal Form:
   - $S \rightarrow \varepsilon$
   - $A \rightarrow a$, with $A \in N$ and $a \in T$
   - $A \rightarrow B$, with $A, B \in N$
   - $A \rightarrow BC$ with $A, B, C \in N$
   - $AB \rightarrow AC$ with $A, B, C, D \in N$
   - $AB \rightarrow CB$, with $A, B, C \in N$
   - $AB \rightarrow B$, with $A, B \in N$

   The initial symbol S may appear only in the left-hand sides of the production rules.

The Chomsky hierarchy establishes the relationship of the language classes defined by the previously established classes of formal grammars. So, REG, CF, CS and RE refer to the languages generated by the previously defined grammars, and we have the following inclusions (the Chomsky hierarchy):

$$REG \subset CF \subset CS \subset RE.$$

3 Generating networks of genetic processors

In the following, we define the *Generating Networks of Genetic Processors*. The basic elements of the model are inspired by previous works on Networks of Evolutionary Processors (NEPs) [9, 10], and Networks of Splicing Processors (NSPs) [13, 14]. In addition, the main ingredients of Generating Networks of Genetic Processors were previously defined as Accepting Networks of Genetic Processors [6].

Given the alphabet V, a *mutation rule* $a \rightarrow b$, with $a, b \in V$, can be applied over the string xay to produce the new string xby. A mutation rule can be viewed as a substitution rule introduced in [10].

A *crossover operation* is an operation over strings defined as follows: Let x and y be two strings, then $x \bowtie y = \{x_1y_2, y_1x_2 : x = x_1x_2 \text{ and } y = y_1y_2\}$. Observe that $x, y \in x \bowtie y$ given that ϵ is a prefix and a suffix of any string. The operation is extended over languages as $L_1 \bowtie L_2 = \bigcup_{x \in L_1, y \in L_2} x \bowtie y$. The crossover operation is a splicing operation over strings where the contexts of the strings are empty [17].

Let P and F be two disjoint subsets of an alphabet V, and let $w \in V^*$. We define the predicate φ as follows:[1]

$$\varphi(w, P, F) \equiv (P = \emptyset \vee alph(w) \cap P \neq \emptyset) \wedge (F \cap alph(w) = \emptyset)$$

We can extend the predicate to act over segments instead of symbols. Let P and F be two disjoint sets of finite strings over V, and let $w \in V^*$, then

$$\varphi(w, P, F) \equiv (P = \emptyset \vee seg(w) \cap P \neq \emptyset) \wedge (F \cap seg(w) = \emptyset).$$

In the following, we work with this extension over segments instead of symbols. The predicate φ is based on random-context conditions defined by the sets P (*permitting contexts*) and F (*forbidding contexts*). Let V be an alphabet and $L \subseteq V^*$, then $\varphi(L, P, F) = \{w \in L : \varphi(w, P, F)\}$.

In the following, we define a *genetic processor* that can be viewed as an abstract machine that can apply mutation rules or crossover operations over a multiset of strings.

[1] In other works such as [6] there have been proposed other predicates where the definition is different from the one proposed here.

Definition 1 Let V be an alphabet. A genetic processor over V is defined by the tuple $(M_R, A, PI, FI, PO, FO, \alpha)$, where

- M_R is a finite set of mutation rules over V
- A is a multiset of strings over V with a finite support and an arbitrary large number of copies of every string.[2]
- $PI, FI \subseteq V^*$ are finite sets with the input permitting/forbidding contexts
- $PO, FO \subseteq V^*$ are finite sets with the output permitting/forbidding contexts
- $\alpha \in \{m, c\}$ defines the function mode as follows:
  - If $\alpha = m$ the processor applies mutation rules.
  - If $\alpha = c$ the processor applies crossover operations, and $M_R = \emptyset$.

In any genetic processor, and for any word $w \in V^*$, there is an input filter $\rho(w) = \varphi(w, PI, FI)$ and an output filer $\tau(w) = \varphi(w, PO, FO)$. That is, $\rho(w)$ (resp. $\tau(w)$) indicates whether or not the word w passes the input (resp. the output) filter of the processor. We can extend the filters to act over languages. So, $\rho(L)$ (resp. $\tau(L)$) is the subset of L with the words that can pass the input (resp. output) filter of the processor.

Once we have defined the main component of the model, that is the genetic processor, we can formulate the Generating Networks of Genetic Processors as follows:

Definition 2 A Generating Network of Genetic Processors (GNGP) is defined by the tuple $\Pi = (V, V_{out}, N_1, N_2, \ldots, N_n, G, \mathcal{N}, N_{out})$, where V is an alphabet, $V_{out} \subseteq V$ is an output alphabet, $G = (X_G, E_G)$ is a graph, $N_i (1 \leq i \leq n)$ is a genetic processor over V, $\mathcal{N} : X_G \to \{N_1, N_2, \ldots, N_n\}$ is a mapping that associates the genetic processor N_i to the node $i \in X_G$, and the processor $N_{out} \in \{N_1, \cdots, N_n\}$ is the output processor.

A *configuration* of a GNGP $\Pi = (V, V_{out}, N_1, N_2, \ldots, N_n, G, \mathcal{N}, N_{out})$ is defined by the tuple $C = (L_1, L_2, \ldots, L_n)$, where L_i is a multiset of strings defined over V for all $1 \leq i \leq n$. A configuration represents the multisets of strings that every processor holds at a given time (remember that every string appears in an arbitrarily large number of copies). The initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$.

Every copy of any string in L_i can be changed by applying a *genetic step*, according to the mutation rules or the crossover operations in the processor N_i. Formally, we say that the configuration $C_1 = (L_1, L_2, \ldots, L_n)$ directly changes into the configuration $C_2 = (L'_1, L'_2, \ldots, L'_n)$ by a genetic step, denoted by $C_1 \Rightarrow C_2$, if L'_i is the multiset of strings obtained by applying the mutation rules or the crossover operations of N_i to the strings in L_i. Since an arbitrarily large number of copies of each string is available in every processor, after a genetic step, in each processor, one gets an arbitrarily large number of copies of any string, that can be obtained by using all

[2] A *multiset* is a set where each element can appear a number of times greater than one. In our case, each element of the multiset A appears a non-bounded number of times but the number of distinct elements defined in A is finite, that is, A has a finite support.

possible mutation rules or crossover operations associated with that processor. If L_i is empty for some $1 \leq i \leq n$, then L'_i is empty as well.

In a *communication step*, each processor N_i sends all copies of the strings to all the processors connected to N_i according to G, provided that they are able to pass its output filter. In addition, it receives all the copies of the strings sent by the processors connected to N_i according to G, provided that they can pass its input filter. Formally, we say that the configuration C' is obtained in one communication step from configuration C, denoted by $C \vdash C'$, iff

$$(\forall x \in X_G)\ C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))).$$

Observe that, if one string leaves any processor together with all its copies, and it cannot enter into any new processor due to its filter restrictions, then the string and all its copies are lost.

Let $\Pi = (V, V_{out}, N_1, N_2, \dots, N_n, G, \mathcal{N}, N_{out})$ be a GNGP. A *computation* in Π is a sequence of configurations $C_0, C_1, \dots$, where C_0 is the initial configuration of Π, $C_{2i} \Rightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$. In the following, we will use the symbol $\mapsto$ to denote a genetic step followed by a communication step. That is, $C \mapsto C'$ iff $C \Rightarrow C'' \vdash C'$. In addition, to emphasize that the computation is carried out in the network Π we denote it by the symbol $\underset{\Pi}{\mapsto}$. Consequently, the reflexive and transitive closure of $\mapsto$ is denoted by $\overset{*}{\mapsto}$. In addition, the symbol $\overset{i}{\underset{\Pi}{\mapsto}}$ denotes i genetic steps plus i communication steps in the usual alternating way. All the strings defined over V_{out} which are in the output processor N_{out} at any computation step belong to the language generated by the network. Let $\Pi = (V, V_{out}, N_1, \dots, N_n, G, \mathcal{N}, N_{out})$ be a GNGP with $N_{out} = N_k$, the language generated by Π is denoted by $L(\Pi)$, and it is defined as follows.

$$L(\Pi) = \{x \in V_{out}^* \cap L_k \,:\, (A_0, \dots, A_k, \dots A_n) \overset{*}{\underset{\Pi}{\mapsto}} (L_0, \dots, L_k, \dots, L_n)\}$$

Observe that according to the definition any finite language L can be trivially generated by a GNGP by defining $A_{out} = L$. Hence, the empty string ε can be generated by including it as an element of A_{out}. In the following section we will not consider the empty string since, according to the criteria defined above, it can be generated in a trivial way.

3.1 Generating networks of genetic processors are parallel genetic algorithms

Once we have defined the Networks of Genetic Processors as generating devices, we are going to relate them to the classical concept of genetic algorithms and, particularly, to the case of parallel genetic algorithms. We followed this approach in a previous work [6] where we could also formulate genetic algorithms as decision problem solvers, as opposed to the more classical view that implies their definition as optimization algorithms. We have followed fundamentally the reference [16] for the case of genetic algorithms, and [8] for the case of parallel genetic algorithms.

We can see that every string inserted in a genetic processor is an individual of its population with an undefined number of clones. In addition, the mutation and crossover operations are applied in this case in a non-uniform way (the genetic crossover of two individuals is considered as a case where the new individuals can extend their genetic code indefinitely). In our case, the fitness function, used for the selection of different individuals to generate new populations, is limited to the selection of individuals for the migration rates discussed below. In the case of GNGPs, the genetic operators are applied to the entire population without making any exceptions. Mutation and crossover ratios are kept uniform throughout the process and no elitism technique is applied to the selection of individuals. Finally, we want to note that the output processor of the network would contain individuals resulting from computation which, in the case of genetic algorithms, would be the population with all feasible solutions to a given problem.

For the case of parallel and distributed genetic algorihms, [3] and [2] define the main components to be established. We can enumerate these components as follows:

1. The distribution of the individuals in different populations. They can be organized in different topologies: master– slave, multiple populations or islands, fine-grained populations or hierarchical and hybrid populations. In addition, the neighborhood connections can be rings, (m, n)-complete, ladders, grids, etc.
2. The synchronicity of evolution and communication of the populations
3. The migration phenomena: migration rates (the percentage of individuals that migrate from one population to a different one), migration selection (the selections of the individuals that migrate) and migration frequency.

The above three aspects are covered in the definition of the GNGPs. The topology of connection of populations, and their initial distribution is made by means of the configuration of the processors connection graph. The evolution and communication of the populations is carried out by means of the universal clock underlying the definition of the operating mode of the networks. Finally, the migration processes are regulated by the definition of the input and output filters of each processor. In other words, the filters defined in the processors are effective procedures for selecting the individuals who can migrate from one population to another.

Therefore, we can conclude that the GNGPs definition meets the main ingredients of parallel and distributed genetic algorithms. In this way, we can initiate a formal study about the generative capacity of the genetic algorithms and how many populations acting in parallel are necessary to be able to generate different populations formalized under the paradigm of the theory of formal languages. This study will be addressed in the following section.

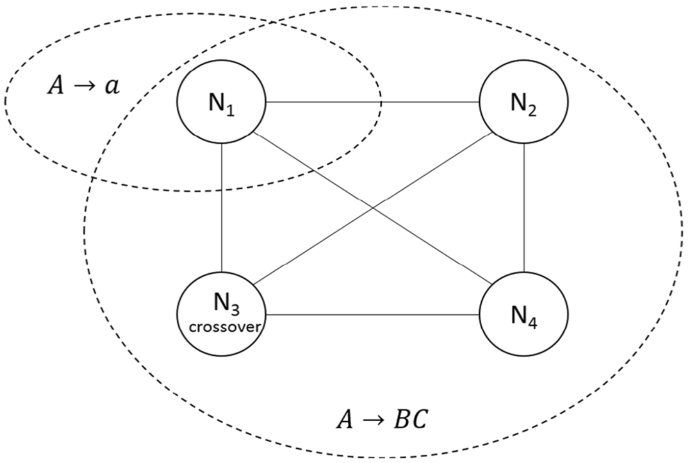

Fig. 1 GNGP structure for regular grammars

4 Generating formal languages through generating networks of genetic processors

In this section, we propose different Generating Networks of Genetic Processors to generate the formal languages according to the Chomsky hierarchy. The number of processors of every proposed network structure is important in order to generate every language class. A general criterion that has been followed to formulate the different topologies and their use in the theorem proofs is that processors that use crossover are used to increase the lengths of the strings that are generated in the grammar, while processors that use mutation are used to make effective the substitution of non-terminal symbols by new terminal or non-terminal symbols.

In the following, we denote a complete graph with n vertexes by K_n and the identity function from processors to vertexes by *id*.

Our first result is related to the regular language class, defined by right linear grammars. We provide the following theorem.

Theorem 1 *Every regular language can be generated by a GNGP with* 3 *processors.*

Proof Let $L = L(G)$ and $G = (N, T, P, S)$ be a right linear grammar. We define the GNGP $\Pi = (V, T, N_1, N_2, N_3, K_3, id, N_1)$, where $V = N \cup T \cup \hat{N} \cup [TN]$, with $\hat{N} = \{\hat{A} : A \in N\}$ and $[TN] = \{[aB] : a \in T, B \in N\}$. The processors in Π are defined as follows

1. $N_1 = (M_1, \{S\}, N, FI_1, PO_1, \emptyset, m)$
   $M_1 = \{A \to [bC] : (A \to bC) \in P\} \cup \{A \to a : (A \to a) \in P\}$
   $FI_1 = [TN] \cup \hat{N}$
   $PO_1 = [TN]$
2. $N_2 = (\emptyset, A_2, PI_2, \emptyset, PO_2, \emptyset, c)$

$A_2 = \hat{N}$
$PI_2 = [TN]$
$PO_2 = VV \cup (V - \hat{N})$

3. $N_3 = (M_3, \emptyset, PI_3, \emptyset, V, FO_3, m)$
$M_3 = \{[aA] \to a : A \in N \wedge a \in T\} \cup \{\hat{A} \to A : A \in N\}$
$PI_3 = \{[aA]\hat{A} : a \in T \wedge A \in N\}$
$FO_3 = [TN] \cup \hat{N}$

The network structure is shown in Fig. 1, and it simulates the derivation process in the regular grammar. Processor N_1 collects the output strings and it applies the productions of the grammar. Observe that the strings of the language cannot leave the processor due to the output filter definition. Processor N_2 is used for the case when a production in the form $A \to bC$ has been applied. It applies crossover in order to add a new symbol from $\hat{N}$ at the end of the string. Observe that, due to the PO_2 filter definition, all the symbols from $\hat{N}$ remain in the processor N_2. Finally, processor N_3 substitutes the symbols from $\hat{N}$ and $[TN]$ used by N_1 and N_2 in order to keep the derivation process of the grammar. We propose the following enunciate that allows the formal proof of the theorem:

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \text{ iff } (\{S\}, A_2, \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3)$$

with $\alpha \in L_1 \cap (N \cup T)^*$

In order to prove the statement, we carry out an induction process over the number of derivation steps in the grammar G to obtain α. So, we prove that if $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ then a configuration (L_1, L_2, L_3) exists such that $\alpha \in L_1 \cap (N \cup T)^*$

Induction base

In this case, only one derivation step in the grammar is applied as $S \underset{G}{\Rightarrow} \alpha$ (observe that we have omitted the trivial case $S \overset{*}{\underset{G}{\Rightarrow}} S$). If one derivation step is carried out, the production $S \to \alpha \in P$. If $\alpha = a$ then $S \to a \in M_1$, and in the first genetic step S mutates to a and, in the following communication step, a does not leave the processor N_1. So, $(\{S\}, A_2, \emptyset) \underset{\Pi}{\mapsto} (L_1, L_2, L_3)$, $a \in L_1$ and the statement is true.

If $\alpha = aB$ then $S \to [aB] \in M_1$ and $[aB] \to a \in M_3$. The following steps are carried out in Π: First, S mutates to $[aB]$ in N_1 during the first genetic step, then the string $[aB]$ is communicated to N_2. Here, by applying the crossover operation, the string $[aB]\hat{B}$ is obtained and it is communicated to N_3. The other strings that can be obtained by applying the crossover operations are sent out the processor N_2, and they cannot enter into a new processor, so they are lost. In the processor N_3, the symbol $[aB]$ mutates to a and the symbol $\hat{B}$ mutates to B in the next two genetic steps. Finally, the string aB is communicated to N_1 and the statement is true.

Induction hypothesis

Let us suppose that for every integer p such that $S \overset{p}{\underset{G}{\Rightarrow}} \alpha$, with $1 \leq p$, there is a number k such that $(\{S\}, A_2, \emptyset) \overset{k}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3)$ with $\alpha \in L_1 \cap (N \cup T)^*$.

Induction step

Finally, let us suppose that $S \overset{p}{\underset{G}{\Rightarrow}} \beta \underset{G}{\Rightarrow} \alpha$. Here, $\beta = wA$ with $w \in T^*$ and $A \in N$ and, by our induction hypothesis, $(\{S\}, A_2, \emptyset) \overset{k}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3)$ with $wA \in L_1$. Now, we consider two cases to obtain α from wA: First, $A \to b \in P$, with $b \in T$, and $\alpha = wb$. In this case, the string wA mutates to wb in N_1 given that $A \to b \in M_1$. In the following communication step, wb does not leave the processor N_1 and the statement is true. The second case is established whenever $A \to bC \in P$ and $\alpha = wbC$. Here, the string wA is in processor N_1 by our induction hypothesis. Then, the string wA mutates to $w[bC]$ given that $A \to [bC] \in M_1$. The string $w[bC]$ is then communicated to N_2 and, in the next genetic step, by applying the crossover operation in N_2 the string $w[bC]\hat{C}$ is obtained and communicated to N_3. In the next two genetic steps, the string $w[bC]\hat{C}$ mutates to wbC which is finally communicated to N_1, and the statement is true.

The second part of the statement can be established as follows:

$$\text{if } (\{S\}, A_2, \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3) \text{ with } \alpha \in L_1 \cap (N \cup T)^* \text{ then } S \overset{*}{\underset{G}{\Rightarrow}} \alpha.$$

Here, we can carry out the induction proof in a way similar to the previous one.

Given that for every string $w \in L(G)$, $S \overset{*}{\underset{G}{\Rightarrow}} w$, we have proved that there exists a configuration (L_1, L_2, L_3) with $w \in L_1$ which can be obtained during the network computation and, consequently, $w \in L(\Pi)$. So, the theorem is proved to be true. □

Example 1 Let G be the regular grammar defined by the following productions

$$S \to aA \mid bB \quad A \to aA \mid a \quad B \to bB \mid b$$

The grammar G generates the language $L(G) = \{a^n : n \geq 2\} \cup \{b^n : n \geq 2\}$.

We define a GNGP $\Pi = (V, T, N_1, N_2, N_3, K_3, id, N_1)$ that generates $L(G)$ as follows:

$$\begin{aligned} V &= \{S, A, B, a, b, \hat{S}, \hat{A}, \hat{B}, [aA], [aB], [aS], [bA], [bB], [bS]\} \\ T &= \{a, b\} \end{aligned}$$

The processor $N_1 = (M_1, \{S\}, \{S, A, B\}, FI_1, PO_1, \emptyset, m)$ where M_1 is defined by the rules:

$$\begin{array}{lll} S \to [aA] & S \to [bB] & A \to [aA] \\ A \to a & B \to [bB] & B \to b \end{array}$$

$$FI_1 = \{\hat{S}, \hat{A}, \hat{B}, [aA], [aB], [aS], [bA], [bB], [bS]\}, \text{ and}$$

$$PO_1 = \{[aA], [aB], [aS], [bA], [bB], [bS]\}$$

The processor $N_2 = (\emptyset, A_2, PI_2, \emptyset, PO_2, \emptyset, c)$ where

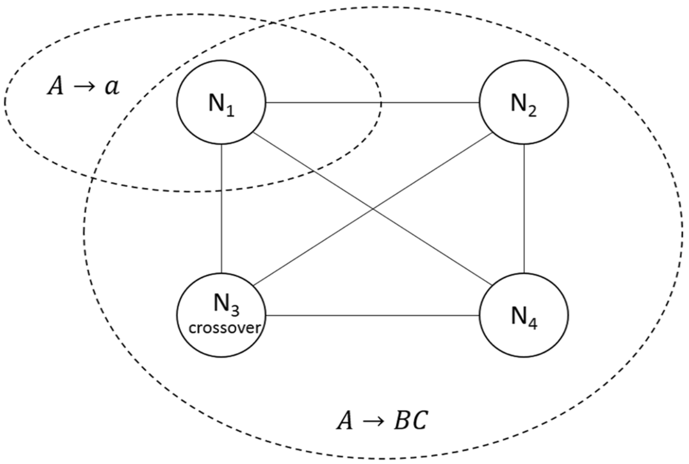

Fig. 2 GNGP structure for context-free grammars

$$\begin{aligned}
A_2 &= \{\hat{S}, \hat{A}, \hat{B}\}, \\
PI_2 &= \{[aA], [aB], [aS], [bA], [bB], [bS]\}, \text{ and} \\
PO_2 &= VV \cup \{S, A, B, a, b, [aA], [aB], [aS], [bA], [bB], [bS]\}
\end{aligned}$$

The processor $N_3 = (M_3, \emptyset, PI_3, \emptyset, V, FO_3, m)$ where M_3 is defined by the rules:

$$\begin{array}{lll}
[aS] \to a & [bS] \to b & [aA] \to a \\
[bA] \to b & [aB] \to a & [bB] \to b \\
\hat{S} \to S & \hat{A} \to A & \hat{B} \to B
\end{array}$$

$$PI_3 = \{[aS]\hat{S}, [bS]\hat{S}, [aA]\hat{A}, [bA]\hat{A}, [aB]\hat{B}, [bB]\hat{B}\}, \text{ and}$$

$$FO_3 = \{\hat{S}, \hat{A}, \hat{B}, [aA], [aB], [aS], [bA], [bB], [bS]\}$$

The functioning of the network is explained in the following: In processor N_1, the productions of the grammar are effectively applied. For example, if the production is $A \to a$ then directly the mutation $A \to a$ is applied, while if the production $A \to aA$ is applied then the mutation $A \to [aA]$ mutates the auxiliary symbol. Then the mutated string with the $[aA]$ symbol is sent to the N_2 processor. In processor N_2, by means of crossover operations, strings with the segment $[aA]\hat{A}$ are obtained. These strings are sent to processor N_3. In processor N_3 the mutation symbols $[aA]$ are changed to a and the mutation symbols $\hat{A}$ are changed to A. The string, which already contains only auxiliary and terminal symbols of the grammar G, is sent back to processor N_1 and a new grammar derivation cycle can be applied. Note that those strings containing only terminal symbols do not leave the processor N_1 and they are the strings generated by the grammar G. □

For the class of context-free languages we use an additional processor with respect to the network structure used in the regular case. In Fig. 2, we show the network structure that we use in the following proof.

Theorem 2 *Every context-free language can be generated by a GNGP with* 4 *processors.*

Proof Let $G = (N, T, P, S)$ be a context-free grammar in Chomsky Normal Form and $L = L(G)$. We define the GNGP $\Pi = (V, T, N_1, N_2, N_3, N_4, K_4, id, N_1)$, where $V = N \cup T \cup \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT] \cup [NT]' \cup [NN]' \cup [TN]' \cup [TT]' \cup [[NN]] \cup \overline{[[NN]]} \cup [[NN]]^f$. The alphabets are defined as follows:

$$\begin{array}{ll}
\hat{N} = \{\hat{A} : A \in N\} & [NT]' = \{[Aa]' : A \in N, a \in T\} \\
\hat{T} = \{\hat{a} : a \in N\} & [NN]' = \{[AB]' : A, B \in N\} \\
\bar{N} = \{\bar{A} : A \in N\} & [TN]' = \{[aA]' : a \in T, A \in N\} \\
\bar{T} = \{\bar{a} : a \in T\} & [TT]' = \{[ab]' : a, b \in T\} \\
[NT] = \{[Aa] : A \in N, a \in T\} & [[NN]] = \{[[AB]] : A, B \in N\} \\
[NN] = \{[AB] : A, B \in N\} & \overline{[[NN]]} = \{\overline{[[AB]]} : A, B \in N\} \\
[TN] = \{[aA] : a \in T, A \in N\} & [[NN]]^f = \{[[AB]]^f : A, B \in N\} \\
[TT] = \{[ab] : a, b \in T\} &
\end{array}$$

The processors in Π are defined as follows

1. $N_1 = (M_1, \{S\}, V, FI_1, PO_1, \emptyset, m)$

   $M_1 = \{A \to [[BC]] : (A \to BC) \in P\} \cup \{A \to a : (A \to a) \in P\}$
   $FI_1 = \bar{N} \cup \bar{T} \cup \hat{N} \cup \hat{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT] \cup [NN]' \cup \overline{[[NN]]} \cup [[NN]]^f$
   $PO_1 = [[NN]]$
2. $N_2 = (M_2, \emptyset, PI_2, \emptyset, V, FO_2, m)$

   $$\begin{aligned} M_2 = \ & \{A \to [BA] : A, B \in (N \cup T)\} \cup \{A \to [BA]' : A, B \in (N \cup T)\} \\ & \cup \{A \to \bar{A} : A \in (N \cup T)\} \cup \{[[AB]] \to \overline{[[AB]]} : A, B \in N\} \\ & \cup \{[[AB]] \to [[AB]]^f : A, B \in N\} \end{aligned}$$
   $PI_2 = [[NN]]$
   $FO_2 = [[NN]] \cup N \cup T$
3. $N_3 = (\emptyset, (\hat{N} \cup \hat{T}), PI_3, FI_3, PO_3, \emptyset, c)$

   $PI_3 = \{[AB]' : A, B \in (N \cup T)\} \cup \{[[AB]]^f : A, B \in N\}$
   $$\begin{aligned} FI_3 = \ & N \cup T \cup \hat{N} \cup \hat{T} \cup \{[AB][CD] : B \neq C\} \\ & \cup \{[AB][CD]' : B \neq C\} \cup \{\overline{[[AB]]}[CD] : B \neq C\} \\ & \cup \{\overline{[[AB]]}[CD]' : B \neq C\} \cup \{[AB]\overline{[[CD]]} : A, B \in (N \cup T) \wedge C, D \in N\} \\ & \cup \{[AB]'C : A, B \in (N \cup T) \wedge C \in V\} \cup \{\bar{A}[BC]' : A, B, C \in (N \cup T)\} \\ & \cup \{\bar{A}[BC] : A, B, C \in (N \cup T)\} \cup \{[[AB]]^f C : A, B \in N \wedge C \in V\} \\ & \cup \{[AB][[CD]]^f : A, B \in (N \cup T) \wedge C, D \in N\} \end{aligned}$$
   $PO_3 = \{AB : A, B \in V\} \cup (V - (\hat{N} \cup \hat{T}))$

4. $N_4 = (M_4, \emptyset, PI_4, \emptyset, V, FO_4, m)$
   $M_4 = \{[AB] \to A : A, B \in (N \cup T)\} \cup \{[AB]' \to A : A, B \in (N \cup T)\}$
   $\cup\{\overline{[[AB]]} \to A : A, B \in N\} \cup \{[[AB]]^f \to A : A, B \in N\}$
   $\cup\{\hat{A} \to A : A \in (N \cup T)\} \cup \{\bar{A} \to A : A \in (N \cup T)\}$
   $PI_4 = \{[[AB]]^f \hat{B} : A, B \in N\} \cup \{[AB]'\hat{B} : A, B \in (N \cup T)\}$
   $FO_4 = \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT] \cup \overline{[[NN]]} \cup [NT]'$
   $\cup[NN]' \cup [TN]' \cup [TT]' \cup [[NN]]^f$

The processor N_1 collects the output strings and, in the network, the following transformations are carried out: Processor N_1 applies the grammar rules in the Chomsky Normal Form. The rule $A \to a$ is directly applied, while the rule $A \to BC$ needs a sequence of transformations in processors N_2, N_3 and N_4. First, A is transformed into $[[BC]]$ in processor N_1. Then, a string in the form $x[[BC]]y$, with $x, y \in (N \cup T)^*$, enters into processor N_2. In processor N_2 the string $x[[BC]]y_1y_2 \dots y_n$, with $y_j \in N \cup T$ is transformed into the string $\bar{x}[[BC]][Cy_1][y_1y_2] \dots [y_{n-2}y_{n-1}][y_{n-1}y_n]'$ and $[[BC]]$ is transformed into $\overline{[[BC]]}$ or $[[BC]]^f$. The symbol $[[BC]]^f$ is used for the case that y is the empty string and the symbol $[[BC]]$ is at the end of the string. Hence, only the strings in the form $\bar{x}[[BC]][Cy_1][y_1y_2] \dots [y_{n-2}y_{n-1}][y_{n-1}y_n]'$ or $\bar{x}[[BC]]^f$ can enter into the processor N_3. The rest of transformed strings leave the processor N_2 and cannot enter into a new processor, so they are lost. In processor N_3, crossover is carried out, and the strings $\bar{x}[[BC]][Cy_1][y_1y_2] \dots [y_{n-1}y_n]'\hat{y}_n$ or $\bar{x}[[BC]]^f\hat{C}$ can be obtained. In one communication step, all the strings obtained by crossover leave the processor N_3. Only the strings in the form $\bar{x}[[BC]][Cy_1] \dots [y_{n-1}y_n]'\hat{y}_n$ or $\bar{x}[[BC]]^f\hat{C}$ can enter into processor N_4. In the processor N_4, all the symbols that have been inserted or transformed in processors N_2 and N_3 are changed to the symbols of the grammar G. So, the string $\bar{x}[[BC]][Cy_1][y_1y_2] \dots [y_{n-1}y_n]'\hat{y}_n$ is transformed to $xBCy_1y_2 \dots y_{n-2}y_{n-1}y_n$ and the string $\bar{x}[[BC]]^f\hat{C}$ is transformed to xBC. The transformed strings leave the processor N_3 and they can enter only in processor N_1 where a new sequence of transformation could start again.

Formally, we can prove the following statement

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \text{ iff } (\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4) \text{ with } \alpha \in L_1 \cap (N \cup T)^*$$

We carry out an induction process over the number of derivation steps in the grammar G to obtain α. So, we prove that if $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$ then a configuration (L_1, L_2, L_3, L_4) exists with $\alpha \in L_1 \cap (N \cup T)^*$

Induction base

In this case, only one derivation step in the grammar is applied as $S \underset{G}{\Rightarrow} \alpha$. If one derivation step is carried out, the production $S \to \alpha \in P$. If $\alpha = a \in T$ then $S \to a \in M_1$, and in the first genetic step S mutates to a and, in the following communication step, a does not leave the processor N_1. So, $(\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4)$, $a \in L_1$ and the statement is true.

If $\alpha = AB$ then $S \to [[AB]] \in M_1$. Then, $[[AB]]$ leaves N_1 and enters into N_2 where it mutates to $[[AB]]^f$. The string $[[AB]]^f$ leaves N_2 and it enters into N_3. In the processor N_3, the strings $[[AB]]^f$ and $\hat{B}$ are recombined by crossover to obtain the string $[[AB]]^f\hat{B}$ that leaves the processor N_3 and it enters into the processor N_4. Finally, in

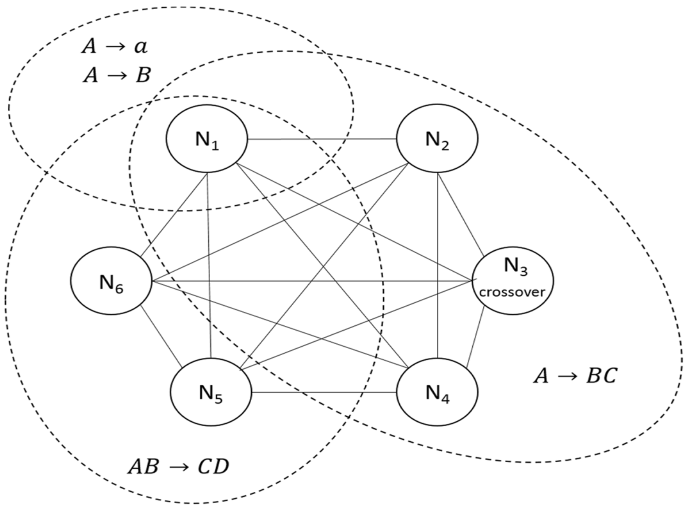

Fig. 3 GNGP structure for context-sensitive grammars

the processor N_4, the string $[[AB]]^f\hat{B}$ is transformed into AB and communicated to the processor N_1 where it enters. So, $(\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4)$, $AB \in L_1$ and the statement is true.

Induction hypothesis

Let us suppose that for every integer $p \geq 1$ such that $S \overset{p}{\underset{G}{\Rightarrow}} \alpha$, it also holds that $(\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4)$ with $\alpha \in L_1 \cap (N \cup T)^*$

Induction step

Let us suppose that $S \overset{p}{\underset{G}{\Rightarrow}} \beta \underset{G}{\Rightarrow} \alpha$, with $p \geq 1$. Here, $\beta = \beta_1 A \beta_2$ with $\beta_1, \beta_2 \in (N \cup T)^*$ and $A \in N$ and, by our induction hypothesis, $(\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4)$ with $\beta_1 A \beta_2 \in L_1 \cap (N \cup T)^*$. Now, we consider two cases to obtain α from $\beta_1 A \beta_2$: First, $A \to b \in P$ and $\alpha = \beta_1 b \beta_2$. In this case, the string $\beta_1 A \beta_2$ mutates to $\beta_1 b \beta_2$ in N_1 given that $A \to b \in M_1$. In the following communication step, $\beta_1 b \beta_2$ does not leave the processors N_1 and the statement is true.

The second case is when $A \to BC \in P$ and $\alpha = \beta_1 BC \beta_2$. Here, the string $\beta_1 A \beta_2$ is in processor N_1 as it is established in our induction hypothesis, and it is transformed into the string $\beta_1 BC \beta_2$ through a sequence of operations in the processors N_2, N_3 and N_4 as we have described before. The string $\beta_1 BC \beta_2$ enters into the processor N_1, and the statement holds.

The second part of the statement can be established as follows:

$$\text{if } (\{S\}, A_2, \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3) \text{ with } \alpha \in L_1 \cap (N \cup T)^* \text{ then } S \overset{*}{\underset{G}{\Rightarrow}} \alpha.$$

Here, we can carry out the induction proof in a way similar to the first part of the statement.

Given that every string $w \in L(G)$ follows from $S \overset{*}{\underset{G}{\Rightarrow}} w$, we have proved that there exists a configuration (L_1, L_2, L_3, L_4) such that $w \in L_1$ that can be obtained during the network computation and, consequently, $w \in L(\Pi)$. Hence, the theorem is true. □

The following class in the Chomsky hierarchy is the class of context-sensitive languages. In this case, we use part of the constructions that we have shown before. In Fig. 3, we show the network structure that we propose in the following result.

Theorem 3 *Every context-sensitive language can be generated by a GNGP with* 6 *processors.*

Proof Let $G = (N, T, P, S)$ be an arbitrary grammar in Kuroda's normal form. We propose the network $\Pi = (V, N_1, N_2, N_3, N_4, N_5, N_6, K_6, id, N_1)$ with

$$V = N \cup T \cup \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT] \cup [NT]' \cup [NN]' \cup [TN]' \cup [TT]' \cup [[NN]] \cup [[NN]]^f \cup \overline{[[NN]]} \cup [NNN]^l \cup [NNN]^r,$$

where the alphabets are defined as in the proof of Theorem 2, and the new alphabets are defined as $[NNN]^l = \{[ABC]^l : A, B, C \in N\}$ and $[NNN]^r = \{[ABC]^r : A, B, C \in N\}$.

The processors are defined as follows:

1. $N_1 = (M_1, \{S\}, V, FI_1, PO_1, \emptyset, m)$

$$\begin{aligned} M_1 &= \{A \to [[BC]] : (A \to BC) \in P\} \cup \{A \to a : (A \to a) \in P\} \\ &\quad \cup \{A \to B : (A \to B) \in P\} \cup \{A \to [ACD]^l : \exists B \in N \wedge (AB \to CD) \in P\} \\ FI_1 &= \bar{N} \cup \bar{T} \cup \hat{N} \cup \hat{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT] \cup [NN]' \cup [[NN]]^f \cup \overline{[[NN]]} \\ PO_1 &= [[NN]] \cup \{[ACD]^l : \exists B \in N \wedge (AB \to CD) \in P\} \end{aligned}$$

2. $N_2 = (M_2, \emptyset, PI_2, \emptyset, V, FO_2, m)$

$$\begin{aligned} M_2 &= \{A \to [BA] : A, B \in (N \cup T)\} \cup \{A \to [BA]' : A, B \in (N \cup T)\} \\ &\quad \cup \{A \to \bar{A} : A \in (N \cup T)\} \cup \{[[AB]] \to \overline{[[AB]]} : A, B \in N\} \\ &\quad \cup \{[[AB]] \to [[AB]]^f : A, B \in N\} \\ PI_2 &= [[NN]] \\ FO_2 &= [[NN]] \cup N \cup T \end{aligned}$$

3. $N_3 = (\emptyset, (\hat{N} \cup \hat{T}), PI_3, FI_3, PO_3, \emptyset, c)$

$$\begin{aligned} PI_3 &= [NN]' \cup [NT]' \cup [TN]' \cup [TT]' \cup [[NN]]^f \\ FI_3 &= \{[AB][CD] : B \neq C\} \cup \{[AB][CD]' : B \neq C\} \cup \{\overline{[[AB]]}[CD] : B \neq C\} \\ &\quad \cup \{\overline{[[AB]]}[CD]' : B \neq C\} \cup \{[AB]\overline{[[CD]]} : A, B \in (N \cup T) \wedge C, D \in N\} \\ &\quad \cup \{[AB]'C : A, B \in (N \cup T) \wedge C \in V\} \cup \{\bar{A}[BC]' : A, B, C \in (N \cup T)\} \\ &\quad \cup \{\bar{A}[BC] : A, B, C \in (N \cup T)\} \cup \{[[AB]]^f C : A, B \in N \wedge C \in V\} \\ &\quad \cup \{[AB][[CD]]^f : A, B \in (N \cup T) \wedge C, D \in N\} \\ &\quad \cup N \cup T \cup \hat{N} \cup \hat{T} \\ PO_3 &= \{AB : A, B \in V\} \cup (V - (\hat{N} \cup \hat{T})) \end{aligned}$$

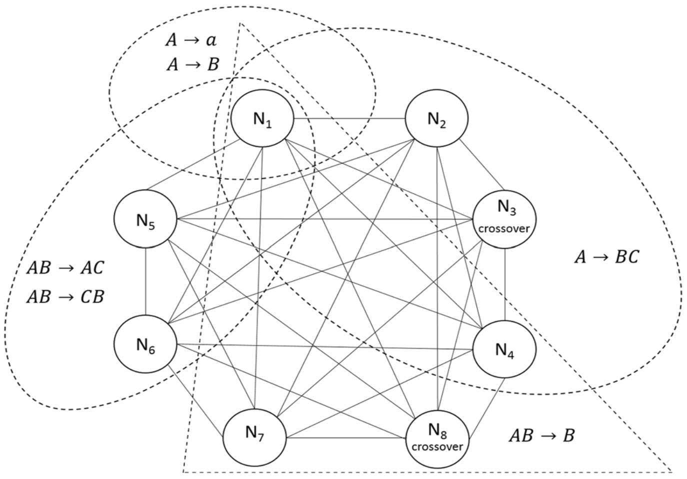

Fig. 4 GNGP structure for phrase structure (non-restricted) grammars

4. $N_4 = (M_4, \emptyset, PI_4, \emptyset, V, FO_4, m)$

$$\begin{aligned} M_4 &= \{[AB] \rightarrow A : A,B \in (N \cup T)\} \cup \{[AB]' \rightarrow A : A,B \in (N \cup T)\} \\ &\quad \cup \{\overline{[[AB]]} \rightarrow A : A,B \in N\} \cup \{[[AB]]^f \rightarrow A : A,B \in N\} \\ &\quad \cup \{\hat{A} \rightarrow A : A \in (N \cup T)\} \cup \{\bar{A} \rightarrow A : A \in (N \cup T)\} \\ PI_4 &= \{[[AB]]^f \hat{B} : A,B \in N\} \cup \{[AB]'\hat{B} : A,B \in (N \cup T)\} \\ FO_4 &= \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup [NN] \cup [NT] \cup [TN] \cup [TT] \cup \overline{[[NN]]} \\ &\quad \cup [NN]' \cup [NT]' \cup [TN]' \cup [TT]' \cup [[NN]]^f \end{aligned}$$

5. $N_5 = (M_5, \emptyset, PI_5, \emptyset, V, \emptyset, m)$

$$\begin{aligned} M_5 &= \{B \rightarrow [BCD]^r : \exists A \in N \wedge (AB \rightarrow CD) \in P\} \\ PI_5 &= \{[ACD]^l B : (AB \rightarrow CD) \in P\} \end{aligned}$$

6. $N_6 = (M_5, \emptyset, PI_6, \emptyset, V, FO_6, m)$

$$\begin{aligned} M_6 &= \{[ACD]^l \rightarrow C : \exists B \in N \text{ with } (AB \rightarrow CD) \in P\} \\ &\quad \cup \{[BCD]^r \rightarrow D : \exists A \in N \text{ with } (AB \rightarrow CD) \in P\} \\ PI_6 &= \{[ACD]^l [BCD]^r : (AB \rightarrow CD) \in P\} \\ FO_6 &= \{[ACD]^l : A,C,D \in N\} \cup \{[BCD]^r : B,C,D \in N\} \end{aligned}$$

As in the proof of Theorem 2, the processor N_1 collects the output strings. The network Π simulates the derivation process in the grammar G. Observe that G is in Kuroda Normal Form. Hence, the productions of the grammar can only be of the following forms:

1. $A \rightarrow a$, with $A \in N$ and $a \in T$
2. $A \rightarrow B$, with $A, B \in N$
3. $A \rightarrow BC$ with $A, B, C \in N$
4. $AB \rightarrow CD$ with $A, B, C, D \in N$

For the case of productions in the form (1) and (3), the network carries out a set of operations that simulate the rule application as we have described in the proof of Theorem 2. In the case of productions of type (3), the processors N_1, N_2, N_3 and N_4 carry out the sequence of transformations previously described. For the case of productions of type (2), the networks directly applies this production at processor N_1 in a way similar to productions of type (1).

Now, we focus on the productions of type (4). The first transformation is carried out at processor N_1, where, for any production in the form $AB \to CD$, the symbol A mutates to the symbol $[ACD]^l$. Then the string is sent out of processor N_1 and it enters into processor N_5 provided that the segment $[ACD]^l B$ appears in the string. Then, the symbol B mutates to $[BCD]^r$, the string is sent out of the processor N_5 and it enters into processor N_6 where $[ACD]^l$ mutates to C and $[BCD]^r$ mutates to D. So, the application of rule $AB \to CD$ is completed and the transformed string returns to processor N_1.

We must prove the following statement

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha \text{ iff } (\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset, \emptyset, \emptyset) \overset{*}{\underset{\Pi}{\mapsto}} (L_1, L_2, L_3, L_4, L_5, L_6)$$
$$\text{with } \alpha \in L_1 \cap (N \cup T)^*$$

The proof is similar as in Theorem 2, with the new rule applications that we have explained before. □

Finally, we define a GNGP network for the last class in the Chomsky hierarchy, that is the phrase structure (non-restricted) grammars. In this case, we propose the network structure that is shown in the Fig. 4. Observe that we take advantage of the previously proposed topologies, and we add new processors in order to deal with grammar productions in the form $AB \to B$. We enunciate the following theorem that can be considered as an universality result for the Generating Networks of Genetic Processors.

Theorem 4 *Every recursively enumerable language can be generated by a GNGP with* 8 *processors.*

Proof Let $G = (N, T, P, S)$ be an arbitrary phrase structure (non-restricted) grammar with the productions in the form established at Sect. 2. We omit the case for the production $S \to \varepsilon$.

We propose the network $\Pi = (V, N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, K_8, id, N_1)$, with where

$$V = N \cup T \cup \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup \widetilde{N} \cup \widetilde{T} \cup [NT] \cup [NN] \cup [TN] \cup [TT]'$$
$$\cup [NT]' \cup [NN]' \cup [TN]' \cup [TT]' \cup [[NN]] \cup [[NN]]^f$$
$$\cup \overline{[[NN]]} \cup [NNN]^l \cup [NNN]^r \cup \langle NN \rangle \cup \langle NT \rangle \cup \langle TN \rangle \cup \langle TT \rangle$$
$$\cup \langle NX \rangle \cup \langle TX \rangle \cup \overline{\langle NN \rangle} \cup \overline{\langle NT \rangle} \cup \overline{\langle TN \rangle} \cup \overline{\langle TT \rangle} \cup \overline{\langle\langle NN \rangle\rangle} \cup \overline{\overline{\langle\langle NN \rangle\rangle}}$$

$X \notin (T \cup N)$. In this case, the new alphabets are defined in a way similar to the previous results.

The processors are defined as follows:

1. $N_1 = (M_1, \{S\}, V, FI_1, PO_1, \emptyset, m)$

$$\begin{aligned}
M_1 &= \{A \rightarrow [[BC]] : (A \rightarrow BC) \in P\} \cup \{A \rightarrow a : (A \rightarrow a) \in P\} \\
&\quad \cup \{A \rightarrow B : (A \rightarrow B) \in P\} \cup \{A \rightarrow \langle\langle AB \rangle\rangle : (AB \rightarrow B) \in P\} \\
&\quad \cup \{A \rightarrow [ACD]^l : \exists B \in N \text{ with } (AB \rightarrow CD) \in P\} \\
FI_1 &= \bar{N} \cup \bar{T} \cup \hat{N} \cup \hat{T} \cup \widetilde{N} \cup \widetilde{T} \cup [NN] \cup [NT] \cup [TN] \cup [TT] \cup [NN] \\
&\quad \cup [[NN]]^f \cup \overline{[[NN]]} \cup \langle\langle NN \rangle\rangle \cup \langle NN \rangle \cup \langle NT \rangle \cup \langle TN \rangle \\
&\quad \cup \langle TT \rangle \cup \langle NX \rangle \cup \langle TX \rangle \cup \overline{\langle NN \rangle} \cup \overline{\langle NT \rangle} \cup \overline{\langle TN \rangle} \cup \overline{\langle TT \rangle} \\
PO_1 &= \{[ACD]^l : \exists B \in N \text{ with } (AB \rightarrow CD) \in P\} \cup [[NN]] \cup \langle\langle NN \rangle\rangle
\end{aligned}$$

2. $N_2 = (M_2, \emptyset, PI_2, \emptyset, V, FO_2, m)$

$$\begin{aligned}
M_2 &= \{A \rightarrow [BA] : A, B \in (N \cup T)\} \cup \{A \rightarrow [BA]' : A, B \in (N \cup T)\} \\
&\quad \cup \{A \rightarrow \bar{A} : A \in (N \cup T)\} \cup \{[[AB]] \rightarrow \overline{[[AB]]} : A, B \in N\} \\
&\quad \cup \{[[AB]] \rightarrow [[AB]]^f : A, B \in N\} \\
PI_2 &= [[NN]] \\
FO_2 &= N \cup T \cup [[NN]]
\end{aligned}$$

3. $N_3 = (\emptyset, (\hat{N} \cup \hat{T}), PI_3, FI_3, PO_3, \emptyset, c)$

$$\begin{aligned}
PI_3 &= [NN]' \cup [NT]' \cup [TN]' \cup [TT]' \cup [[NN]]^f \\
FI_3 &= N \cup T \cup \hat{N} \cup \hat{T} \cup \{[AB][CD] : B \neq C\} \cup \{[AB][CD]' : B \neq C\} \\
&\quad \cup \{\overline{[[AB]]}[CD] : B \neq C\} \cup \{\overline{[[AB]]}[CD]' : B \neq C\} \\
&\quad \cup \{[AB]\overline{[[CD]]} : A, B \in (N \cup T) \wedge C, D \in N\} \\
&\quad \cup \{[AB]'C : A, B \in (N \cup T) \wedge C \in V\} \\
&\quad \cup \{\bar{A}[BC]' : A, B, C \in (N \cup T)\} \\
&\quad \cup \{\bar{A}[BC] : A, B, C \in (N \cup T)\} \\
&\quad \cup \{[[AB]]^f C : A, B \in N \wedge C \in V\} \\
&\quad \cup \{[AB][[CD]]^f : A, B \in (N \cup T) \wedge C, D \in N\} \\
PO_3 &= \{AB : A, B \in V\} \cup (V - (\hat{N} \cup \hat{T}))
\end{aligned}$$

4. $N_4 = (M_4, \emptyset, PI_4, FI_4, V, FO_4, m)$

$$\begin{aligned}
M_4 &= \{[AB] \rightarrow A : A, B \in (N \cup T)\} \cup \{[AB]' \rightarrow A : A, B \in (N \cup T)\} \\
&\quad \cup \{\overline{[[AB]]} \rightarrow A : A, B \in N\} \cup \{[[AB]]^f \rightarrow A : A, B \in N\} \\
&\quad \cup \{\hat{A} \rightarrow A : A \in (N \cup T)\} \cup \{\bar{A} \rightarrow A : A \in (N \cup T)\} \\
&\quad \cup \{\widetilde{A} \rightarrow A : A \in (N \cup T)\} \cup \{\overline{\langle\langle AB \rangle\rangle} \rightarrow B : A, B \in N\} \\
&\quad \cup \{\overline{\overline{\langle\langle AB \rangle\rangle}} \rightarrow B : A, B \in N\} \cup \{\overline{\langle AB \rangle} \rightarrow B : A, B \in (N \cup T)\} \\
&\quad \cup \{\langle AB \rangle \rightarrow B : A, B \in (N \cup T)\} \\
PI_4 &= \{[[AB]]^f \hat{B} : A, B \in N\} \cup \{[AB]'\hat{B} : A, B \in (N \cup T)\} \cup \overline{\langle\langle NN \rangle\rangle} \\
&\quad \cup \overline{\overline{\langle\langle NN \rangle\rangle}} \\
FI_4 &= \{\overline{\langle AB \rangle}C : A, B \in N \wedge C \in V\} \cup \{\overline{\overline{\langle\langle AB \rangle\rangle}}C : A, B \in N \wedge C \in V\} \\
FO_4 &= \hat{N} \cup \hat{T} \cup \bar{N} \cup \bar{T} \cup \widetilde{N} \cup \widetilde{T} \cup [NN] \cup [NT] \cup [TN] \cup [TT] \cup \overline{[[NN]]} \\
&\quad \cup [NN]' \cup [NT] \cup [TN]' \cup [TT]' \cup [[NN]]^f \cup \langle NN \rangle \cup \langle NT \rangle \cup \langle TN \rangle \\
&\quad \cup \langle TT \rangle \cup \overline{\langle NN \rangle} \cup \overline{\langle NT \rangle} \cup \overline{\langle TN \rangle} \cup \overline{\langle TT \rangle} \cup \overline{\langle\langle NN \rangle\rangle} \cup \overline{\overline{\langle\langle NN \rangle\rangle}}
\end{aligned}$$

5. $N_5 = (M_5, \emptyset, PI_5, \emptyset, V, \emptyset, m)$

$$M_5 = \{B \to [BCD]^r : \exists A \in N \text{ with } (AB \to CD) \in P\}$$
$$PI_5 = \{[ACD]^l B : (AB \to CD) \in P\}$$

6. $N_6 = (M_5, \emptyset, PI_6, \emptyset, V, FO_6, m)$

$$M_6 = \{[ACD]^l \to C : \exists B \in N \text{ with } (AB \to CD) \in P\}$$
$$\cup\{[BCD]^r \to D : \exists A \in N \text{ with } (AB \to CD) \in P\}$$
$$PI_6 = \{[ACD]^l[BCD]^r : (AB \to CD) \in P\}$$
$$FO_6 = [NNN]^l \cup [NNN]^r$$

7. $N_7 = (M_7, \emptyset, PI_7, \emptyset, V, FO_7, m)$

$$M_7 = \{A \to \widetilde{A} : A \in (N \cup T)\} \cup \{A \to \langle AB\rangle : A, B \in (N \cup T)\}$$
$$\cup\{A \to \overline{\langle AB\rangle} : A, B \in (N \cup T)\} \cup \{A \to \langle AX\rangle : A \in (N \cup T)\}$$
$$\cup\{\langle\langle AB\rangle\rangle \to \overline{\langle\langle AB\rangle\rangle} : A, B \in N\} \cup \{\langle\langle AB\rangle\rangle \to \overline{\overline{\langle\langle AB\rangle\rangle}} : A, B \in N\}$$
$$PI_7 = \{\langle\langle AB\rangle\rangle B : (AB \to B) \in P\}$$
$$FO_7 = N \cup T \cup \langle\langle NN\rangle\rangle$$

8. $N_8 = (\emptyset, \emptyset, PI_8, FI_8, V, \emptyset, c)$

$$PI_8 = \overline{\langle\langle NN\rangle\rangle} \cup \overline{\overline{\langle\langle NN\rangle\rangle}}$$
$$FI_8 = \{\overline{\langle\langle AB\rangle\rangle}\widetilde{C} : A, B \in N \wedge \widetilde{C} \in (\widetilde{N} \cup \widetilde{T})\}$$
$$\cup\{\langle AX\rangle B : A \in (N \cup T) \wedge B \in V\}$$
$$\cup\{\langle AB\rangle\widetilde{C} : A, B, C \in (N \cup T)\}$$
$$\cup\{\widetilde{C}\langle AB\rangle : A, C \in (N \cup T) \wedge B \in (N \cup T \cup \{X\})\}$$
$$\cup\{\overline{\langle AB\rangle}\widetilde{C} : A, B, C \in (N \cup T)\}$$
$$\cup\{\widetilde{C}\overline{\langle AB\rangle} : A, B, C \in (N \cup T)\}$$
$$\cup\{\overline{\langle AB\rangle}\langle CD\rangle : A, B, C, D \in (N \cup T)\}$$
$$\cup\{\overline{\langle AB\rangle\langle CD\rangle} : A, B, C, D \in (N \cup T)\}$$
$$\cup\{\overline{\langle AB\rangle\langle\langle CD\rangle\rangle} : A, B \in (N \cup T) \wedge C, D \in N\}$$
$$\cup\{\overline{\langle\langle AB\rangle\rangle}\langle CD\rangle : B \neq C\} \cup \{\overline{\langle\langle AB\rangle\rangle\langle CD\rangle} : B \neq C\}$$
$$\cup\{\overline{\langle AB\rangle}\langle CX\rangle : B \neq C\} \cup \{\langle AB\rangle\langle CD\rangle : B \neq C\}$$
$$\cup\{\langle AB\rangle\overline{\langle CD\rangle} : B \neq C\}$$
$$\cup\{\overline{\langle\langle AB\rangle\rangle}\langle CX\rangle : A, B \in N \wedge C \in (N \cup T)\}$$
$$\cup\{\overline{\overline{\langle\langle AB\rangle\rangle}}CD : A, B \in N \wedge C, D \in V\}$$
$$\cup\{\overline{\overline{\langle\langle AB\rangle\rangle}}\widetilde{C} : A, B \in N \wedge \widetilde{C} \in (\widetilde{N} \cup \widetilde{T})\}$$
$$\cup\{\overline{\overline{\langle\langle AB\rangle\rangle}}\langle CD\rangle : A, B \in N \wedge C, D \in (N \cup T)\}$$
$$\cup\{\langle CD\rangle\overline{\overline{\langle\langle AB\rangle\rangle}} : A, B \in N \wedge C, D \in (N \cup T)\}$$
$$\cup\{\overline{\overline{\langle\langle AB\rangle\rangle}}\overline{\langle CD\rangle} : A, B \in N \wedge C, D \in (N \cup T)\}$$
$$\cup\{\overline{\langle CD\rangle}\overline{\overline{\langle\langle AB\rangle\rangle}} : A, B \in N \wedge C, D \in (N \cup T)\}$$

In the proposed network we use the structure that we have established at Theorem 3, and we introduce new processors, filters and mutation rules in order to apply the grammar rules in the form $AB \to B$. We can summarize the rules application as follows: The rules in the form $A \to a$ or $A \to B$ are directly applied in processor N_1 through mutation rules. The rules in the form $A \to BC$ are simulated by the processors N_1, N_2, N_3 and N_4 in a way similar as in Theorem 2 and Theorem 3. The rules in the form $AB \to AC$ and $AB \to CB$ are simulated by the processors N_1, N_5 and N_6 in a way similar as in Theorem 3 (observe that these rules are a restricted case of the rules in the form $AB \to CD$). Finally, the rules in the form $AB \to C$ are simulated as follows: First, at processor N_1, the symbol A mutates to the symbol $\langle\langle AB\rangle\rangle$, and the mutated string is sent out of the processor N_1. The string enters into the processor N_7 provided that it contains the segment $\langle\langle AB\rangle\rangle B$. At processor N_7 the string $\alpha\langle\langle AB\rangle\rangle BA_1A_2\dots A_n$ mutates to $\widetilde{\alpha}\langle\langle AB\rangle\rangle\langle BA_1\rangle\langle A_1A_2\rangle\langle A_2A_3\rangle\dots\langle A_{n-1}A_n\rangle\langle A_nX\rangle$. Observe that if B is the symbol at the rightmost position then the mutated string should be $\widetilde{\alpha}\langle\langle AB\rangle\rangle\langle BX\rangle$. The string is sent out of the processor N_7 and it enters into the processor N_8 where only crossover is applied in order to eliminate the last symbol of the string. Observe that in processor N_8 only self-crossover is carried out, given that the only string at processor is the mutated one. Finally, after the last symbol elimination, the string enters into processor N_4 where all the marked symbols are restored to the symbols of the grammar and the string is sent out to the processor N_1.

As in the previous theorems we can enunciate and prove the following statement:

$$S \underset{G}{\overset{*}{\Rightarrow}} \alpha \text{ iff } (\{S\}, \emptyset, (\hat{N} \cup \hat{T}), \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \underset{\Pi}{\overset{*}{\mapsto}} (L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8)$$
$$\text{with } \alpha \in L_1 \cap (N \cup T)^*$$

□

5 Further remarks and conclusions

First, we remark some achievements of this work: we have proposed the Networks of Genetic Processors as a generative model of computation. In addition, we have justified its definition as parallel genetic algorithms that act with certain restrictions when applying genetic operations, fitness selection and migration procedures between populations. This allows us to see parallel genetic algorithms as computational mechanisms for information generation. We have carried out a study about the generative capacity of the model based on the number of processors that act simultaneously. In our case, the research framework has been that of formal language theory, which is a valid operating framework because it covers all computational processes from the point of view of theoretical computer science. We have been able to prove that three, four, six and eight processors are sufficient to generate the classes of languages according to Chomsky hierarchy which is a generally accepted framework for the study of computatbility theory.

Regarding some improvements that we can address on our proposal, provided that the unitary productions $A \to B$ with $A, B \in N$ can be eliminated in the grammar, then the mutation rules $A \rightarrow B$ can be removed from the processors. Observe that the formulation of new normal forms for the grammars in the Chomsky hierarchy, can lead to new network structures. We think that the proposed networks are optimal with respect to the number of the processors for every language class, and we will analyze the question in future works. Nevertheless, this is not the case for the alphabet of the network and the filters used to apply the grammar productions. We think that these parameters should be deeply studied in order to produce optimal solutions for the descriptive complexity of the proposed model.

Finally, we would like to remark the roles of the operations in the network: The crossover operation is used only to add or remove new symbols in the strings, while the mutation rules together with the input and output filters are the main core of the processors to apply the grammar rules. Hence, the set of strings of a predefined length obtained in the grammar could be generated by applying only mutation rules. This opens a new aspect of Networks of Bio-inspired Processors in order to propose a complexity measure based in a (semi)uniform approach.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. P. Alarcón, F. Arroyo, V. Mitrana, Networks of polarized evolutionary processors. Inf. Sci. **265**, 189–197 (2014)
2. E. Alba, M. Tomassini, Parallelism and evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(2), 443–462 (2002)
3. E. Alba, J. Troya, A survey of parallel distributed genetic algorithms. Complexity **4**(4), 31–52 (1999)
4. F. Arroyo, J. Castellanos, V. Mitrana, E. Santos, J. Sempere, Networks of bio-inspired processors. Triangle **7**, 3–22 (2012)
5. F. Arroyo, S. Gómez Canaval, V. Mitrana, S. Popescu, On the computational power of networks of polarized evolutionary processor. Inf. Comput. **253**, 371–380 (2017)
6. M. Campos, J. Sempere, Accepting networks of genetic processors are computationally complete. Theor. Comput. Sci. **456**, 18–29 (2012)
7. M. Campos, J. Sempere, Solving combinatorial problems with networks of genetic processors. Int. J. Inf. Technol. Knowl. **7**(1), 65–71 (2013)

8. E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms* (Kluwer Academic Publishers, New York, 2001)
9. J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors. In: Proceedings of the 6th International Work-Conference on Artificial Intelligence, IWANN 2001 *LNCS 2084*, Springer, pp. 621–628 (2001)
10. J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors. Acta Inform. **39**, 517–529 (2003)
11. L. Kari, G. Rozenberg, The many facets of natural computing. Commun. ACM **51**(10), 72–83 (2008)
12. F. Manea, V. Mitrana, All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. Inf. Process. Lett. **103**, 112–118 (2007)
13. F. Manea, C. Martín-Vide, V. Mitrana, Accepting networks of splicing processors. In: Proceedings of the First Conference on Computability in Europe, CiE 2005 *LNCS 3526*, Springer, pp. 300–309 (2005)
14. F. Manea, C. Martín-Vide, V. Mitrana, Accepting networks of splicing processors: complexity results. Theor. Comput. Sci. **371**, 72–87 (2007)
15. A. Mateescu, A. Salomaa, *Aspects of Classical Language Theory. In Handbook of Formal Languages*, vol. I (Springer, Berlin, 1997)
16. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs* (Springer, Berlin, 1992)
17. G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing* (Springer, New Computing Paradigms, Berlin, 1998)
18. G. Révész, *Introduction to Formal Languages* (McGraw-Hill Book Co., New York, 1983)