GENERATING NETWORKS OF SPLICING PROCESSORS

Jürgen Dassow[1], Florin Manea[2,3] and Bianca Truthe[1]

Abstract. In this paper, we introduce Generating Networks of Splicing Processors (GNSP for short), a formal languages generating model related to networks of evolutionary processors and to accepting networks of splicing processors. We show that all recursively enumerable languages can be generated by GNSPs with only nine processors. We also show, by direct simulation, that two other variants of this computing model, where the communication between processors is conducted in different ways, have the same computational power.

1991 Mathematics Subject Classification. 68Q05, 68Q42, 68Q45.

Networks of language processors have been introduced in [9] by E. Csuhaj-Varjú and A. Salomaa. Such a network can be considered as a graph where the nodes contain sets of word rewriting rules and, at any moment of time, a language is associated with a node. In a derivation step, any node applies the rules to the words of the associated languages to obtain its new language. In a communication step, any node sends those words that satisfy an output condition given as a regular language (called output filter) to the neighbouring nodes and any node takes words sent by the other nodes, if the words satisfy an input condition also given by a regular language (called input filter). The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, related to Darwinian evolution, by J. Castellanos, C. Martin-Víde, V. Mitrana, and J. M. Sempere in [6], a special type of networks of language processors was introduced which are called networks with evolutionary processors. In such networks, the rewriting rules that were used had

Keywords and phrases: Splicing, Networks of Splicing Processors, Networks of Splicing Processors with Filtered Connections, Computational Completeness.

[1] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, PSF 4120, D-39016 Magdeburg, Germany; e-mail: {dassow,truthe}@iws.cs.uni-magdeburg.de

[2] Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany; e-mail: flm@informatik.uni-kiel.de

[3] Faculty of Mathematics and Computer Science, University of Bucharest, Academiei 14, RO-010014 Bucharest, Romania

a very simple form: they modelled the point mutation known from biology. The sets of productions had to be substitutions of one letter by another letter, insertions of letters or deletions of letters; the nodes were then called substitution node or insertion node or deletion node, respectively. The computation in such a network was conducted in a very similar fashion to the computation in networks of language processors. Results on networks of evolutionary processors can be found, e. g., in [6], [7], [18]. For instance, in [7], it was shown that networks of evolutionary processors are complete in that sense that they can generate any recursively enumerable language; other investigations concerned the finding of the minimum number such that the class of networks with that number of nodes generates the class of recursively enumerable languages [3, 4].

In the paper [17], an accepting version of such systems was introduced by M. Margenstern, V. Mitrana, and M. Pérez-Jiménez; such networks are called Accepting Networks of Evolutionary Processors (ANEPs, for short). A word w is accepted by such a network, if a special input node contains initially $\{w\}$, all other nodes are initially empty, and after some derivation and communication steps a word arrives in a special output node. Again, this type of networks is computationally complete. A summary on accepting networks of evolutionary processors can be found in [16]. Besides the results on the computational power of this model, other results worth mentioning regarded the computational complexity of ANEPs-based computations, their descriptional complexity, and the way ANEPs can be used as problem solvers; the aforementioned survey presents also some results on the way ANEPs can be used to accept picture languages.

The point mutations modelled by substitutions, insertions, and deletions belong to the basic operations which occur in evolution. Another basic operation is splicing. Intuitively, two DNA strands are cut at special positions given by recognition sites (i. e., special DNA sequences) under the effect of some enzymes and then the obtained parts are recombined. It is now natural to consider networks where splicing is used instead of point mutations.

The accepting variant of networks with splicing processors was introduced in [14] by F. Manea, C. Martin-Víde, and V. Mitrana. Again, the computational completeness was shown. Results on the complexity of such networks and the possibility of using them as problem solvers were developed in [14, 15]. They were followed by a series of results regarding the descriptional complexity of the model [12, 13].

In all the types of networks mentioned above, the filters are associated with the nodes. Each node has an input filter and an output filter which determine the words which can enter and leave the node in a communication step. Obviously, one can also introduce variants where the filters are connected with edges and a word can only pass along an edge if it satisfies the filter conditions. For accepting networks with evolutionary or splicing processors, such variants with filters associated with edges are introduced in [1, 2] and [5], respectively.

Surprisingly, a generating variant of networks with splicing processors, in the fashion of the initial model of networks of language processors, has not been considered hitherto. The aim of this paper is to close this gap. We start with a definition of a generating network of splicing processors where we follow as accurately as possible the definition of a network with evolutionary processors. We introduce two types of networks; in the first one, the filters are connected with the nodes; in the second type, the filters are associated with the directed edges. The filters are given by sets of letters which have to be present or absent in the current sentential form.

We prove that each generating network with splicing processors of one of these types (filters associated with nodes or edges) can be transformed into an equivalent generating network with splicing processors of the other type. In addition, we show that these two models are equivalent to a more restricted variant of networks with filters on edges, but where the edges are undirected. It is worth mentioning that the proofs showing that all these models are equivalent are made by direct simulations without using any intermediate model. Moreover, we show the computational completeness of generating networks with splicing processors by showing how one can construct a network that simulates the derivations of a phrase structure grammar.

Finally, we mention that generating networks of splicing processors are closely related to test tube systems (see e. g. [8] and [10]). A test tube behaves as a node of the network and the communication is done via filters which are given by the presence/absence of letters or length conditions. However, a derivation step does not consist of an application of a splicing rule as in the case of networks; in test tube systems the words produced in a derivation step are all the words which can be obtained by iterated applications of splicing rules. Therefore, one has no time bound in which all words are produced since the number of iteration steps can be arbitrarily large. Thus, it seems to us that it is more natural to have a time bound given by one application of a splicing rule. Hence, generating networks of splicing processors can be seen as similar to test tube systems with a special bound on the duration of a splicing step.

1. Basic Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [20]). We here only recall some notations used in the paper.

By V^* we denote the set of all words over an alphabet V (including the empty word λ). The length of a word w is denoted by $|w|$. The number of occurrences of a letter a or of letters from a set A is denoted by $|w|_a$ and $|w|_A$, respectively. For the number of elements of a set A, we write $|A|$. The minimal alphabet of a word w and a language L is denoted by $\mathrm{alph}(w)$ and $\mathrm{alph}(L)$, respectively.

In the proofs, we shall often add new letters of an alphabet U to a given alphabet V. In all these situations, we assume that $V \cap U = \emptyset$.

A phrase structure grammar is a quadruple

$$G = (N, T, P, S)$$

where N is a finite set of non-terminals, T is a finite set of terminals, P is a finite set of productions which are written as $\alpha \to \beta$ with $\alpha \in (N \cup T)^+ \setminus T^*$ and $\beta \in (N \cup T)^*$, and $S \in N$ is the axiom.

A splicing rule over a finite alphabet V is a tuple of the form $[(u_1, u_2), (v_1, v_2)]$ where u_1, u_2, v_1, and v_2 are in V^*. For a splicing rule

$$r = [(u_1, u_2), (v_1, v_2)]$$

and words $x, y, w, z \in V^*$, we say that the rule r produces the pair (z, w) from the pair (x, y) – denoted by

$$(x, y) \vdash_r (z, w)$$

if there exist words $x_1, x_2, y_1, y_2 \in V^*$ such that

$$\begin{aligned} x &= x_1 u_1 u_2 x_2, \\ y &= y_1 v_1 v_2 y_2, \\ z &= x_1 u_1 v_2 y_2, \text{ and} \\ w &= y_1 v_1 u_2 x_2. \end{aligned}$$

For a language L over V and a set of splicing rules R we define

$$\begin{aligned} \sigma_R(L) = {} & \{z, w \in V^* \mid (\exists u, v \in L, r \in R)[(u, v) \vdash_r (z, w)]\} \\ & \cup \{u \in L \mid \forall v \in L, \forall r \in R, r \text{ is not applicable on } (u, v) \text{ or } (v, u)\}. \end{aligned}$$

For two disjoint subsets P and F of an alphabet V and a word w over V, we define the predicates

$$\begin{aligned} \varphi^{(\mathsf{s})}(w; P, F) &\equiv P \subseteq \mathrm{alph}(w) \wedge F \cap \mathrm{alph}(w) = \emptyset, \\ \varphi^{(\mathsf{w})}(w; P, F) &\equiv (P = \emptyset \vee \mathrm{alph}(w) \cap P \neq \emptyset) \wedge F \cap \mathrm{alph}(w) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P of *permitting contexts/symbols* and F of *forbidding contexts/symbols*. Informally, the first condition requires that all permitting symbols are present in w and no forbidding symbol is present in w, while the second one is a weaker variant of the first, requiring that – for a non-empty set P – at least one permitting symbol appears in w and no forbidding symbol is present in w.

For every language $L \subseteq V^*$ and $\beta \in \{(\mathsf{s}), (\mathsf{w})\}$, we define:

$$\varphi^{\beta}(L, P, F) = \{w \in L \mid \varphi^{\beta}(w; P, F)\}.$$

We now introduce the basic concept of this paper, the generating networks of splicing processors (GNSP, for short). See [14, 15] for a closely related accepting model.

Definition 1.1.

(1) A generating network of splicing processors (of size n) is a tuple

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

where

- V is a finite alphabet,
- for $1 \le i \le n$, $N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i, \beta_i)$ where
  - M_i is a set of splicing rules, the rules of node N_i,
  - A_i is a finite subset of V^*, called the set of axioms of node N_i,
  - PI_i, FI_i, PO_i, FO_i are finite subsets of V, and PI_i and FI_i are the input filters of node N_i, while PO_i and FO_i are the output filters of node N_i,
  - $\beta_i \in \{(\mathsf{s}), (\mathsf{w})\}$ indicates the way the input and output filters of the node are used, as follows

$$\begin{aligned} \text{input filter:} \quad & \rho_i(\cdot) = \varphi^{\beta_i}(\cdot; PI_i, FI_i), \\ \text{output filter:} \quad & \tau_i(\cdot) = \varphi^{\beta_i}(\cdot; PO_i, FO_i). \end{aligned}$$

- E is a subset of $\{N_1, N_2, \ldots, N_n\} \times \{N_1, N_2, \ldots, N_n\}$, defining the directed edges of the network, and
- N_o is the output node of the network ($1 \le o \le n$).

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

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

- splicing step (written as $C \Longrightarrow C'$) if, for $1 \le i \le n$,

$$C'(i) = \sigma_{M_i}(C(i)),$$

- communication step (written as $C \vdash C'$) if, for $1 \le i \le n$,

$$C'(i) = (C(i) \setminus \tau_i(C(i))) \ \cup \bigcup_{(N_j, N_i) \in E} \rho_i(\tau_j(C(j))).$$

The computation of $\mathcal{N}$ is a sequence of configurations

$$C_t = (C_t(1), C_t(2), \ldots, C_t(n)), \quad t \ge 0,$$

such that

- $C_0 = (A_1, A_2, \ldots, A_n)$,
- C_{2t} derives C_{2t+1} in a splicing step: $C_{2t} \Longrightarrow C_{2t+1}$ ($t \ge 0$),
- C_{2t+1} derives C_{2t+2} in a communication step: $C_{2t+1} \vdash C_{2t+2}$ ($t \ge 0$).

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

$$L(\mathcal{N}) = \bigcup_{t \ge 0} C_t(o)$$

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

The following example shows how GNSPs can be used to generate a language.

Example 1.2. Let $V = \{a, b\}$ and let $w \in V^*$ be a word over V with $|w| \geq 2$. Let $L = \{x \mid x \in V^*, w \text{ is not a factor of } x\}$.

In order to generate L, we construct the network

$$\mathcal{N} = (U, N_1, N_2, N_3, N_4, \{(N_1, N_2), (N_2, N_3), (N_3, N_1), (N_3, N_4)\}, N_4)$$

depicted in Figure 1, where $U = \{a, b, \$, \#, \perp\}$.

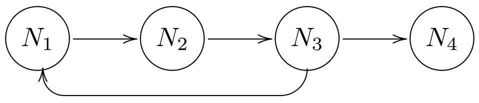

FIGURE 1. Graph of the GNSP $\mathcal{N}$ used in Example 1.2

The processors of $\mathcal{N}$ are defined as follows:

- $N_1 = (M_1, A_1, PI_1, FI_1, PO_1, FO_1, (\mathsf{w}))$, where:

$$\begin{aligned} M_1 &= \{[(\perp s', \$), (\#, s)], [(\perp s, \$), (\perp s, \$)] \mid s, s' \in V\}, \\ A_1 &= \{\#a, \#b, \perp a\$, \perp b\$, \#\$\}, \\ PI_1 &= \{\#\},\ FI_1 = \{\perp, \$\},\ PO_1 = \{\perp\},\ FO_1 = \{\$\}, \end{aligned}$$

- $N_2 = (M_2, A_2, PI_2, FI_2, PO_2, FO_2, (\mathsf{w}))$, where:

$$\begin{aligned} M_2 &= \{[(\perp, w), (\$, \#)], [(\$, \#), (\$, \#)]\}, \\ A_2 &= \{\$\#, \perp\#\}, \\ PI_2 &= \{\perp\},\ FI_2 = \emptyset,\ PO_2 = U,\ FO_2 = \{\#\}, \end{aligned}$$

- $N_3 = (M_3, A_3, PI_3, FI_3, PO_3, FO_3, (\mathsf{w}))$, where:

$$\begin{aligned} M_3 &= \{[(\perp, s), (\#, \$)], [(\perp, s), (\lambda, \#\$)], [(\#, \$), (\#, \$)] \mid s \in V\}, \\ A_3 &= \{\#\$, \perp\#\$, \perp\$\}, \\ PI_3 &= \{\perp\},\ FI_3 = \{\$\},\ PO_3 = U,\ FO_3 = \{\$\}, \end{aligned}$$

- $N_4 = (M_4, A_4, PI_4, FI_4, PO_4, FO_4, (\mathsf{w}))$, where:

$$\begin{aligned} M_4 &= \emptyset, \\ A_4 &= \{a, b\}, \\ PI_4 &= U, FI_4 = U \setminus \{a, b\},\ PO_4 = \emptyset,\ FO_4 = U. \end{aligned}$$

We show that $\mathcal{N}$ generates the language L by induction. We show that, for $k \geq 1$, before the execution of the $3(k-1)+1^{th}$ splicing step of the computation (thus, after $6(k-1)$ computation steps) the following statements hold: N_1 contains all the strings $\#x$, where x is an arbitrary string over V of length k that does not contain w, as well as the strings $\bot a\$$, $\bot b\$$, and $\#\$$; N_2 contains the strings $\$\#$ and $\bot\#$ and no other strings; N_3 contains the strings $\#\$$, $\bot\$$, and $\bot\#\$$; N_4 contains all the strings over V of length less or equal to k that do not contain the factor w. Clearly, this holds for $k=1$, at the beginning of the computation. Assume now that it is also true for some $k \geq 1$. After the execution of the $3(k-1)+1^{th}$ splicing step of the computation, the configurations of the nodes change as follows. The node N_1 contains the strings $\bot s'x$, for all $s' \in V$ and the words $x \in V^k$ that do not contain w, and the string $\#\$$, obtained by splicing $\#x$ with $\bot s'\$$, as well as the strings $\bot s\$$, with $s \in V$, obtained by splicing $\bot s\$$ with a copy of itself. The configurations of the other nodes remain unchanged. Now the $3(k-1)+1^{th}$ communication step is executed. The configurations of the nodes become: the node N_1 contains $\#\$$ and $\bot s\$$, with $s \in V$; the node N_2 contains the strings $\$\#$ and $\bot\#$ and the words $\bot s'x$, for all $s' \in V$ and the words $x \in V^k$ that do not contain w, that came from N_1; the configurations of N_3 and N_4 remain unchanged. A new splicing step is executed now and we obtain the following configurations. The node N_2 contains $\$\#$ and $\bot\#$ and the words $\bot y$, for all $y \in V^{k+1}$ that do not contain w, and $\$y$, for all $y \in V^{k+1}$ that start with w and do not contain the factor w on other positions. All the other configurations of the nodes remain unchanged. After the following communication step, the configurations of N_1 and N_4 remain unchanged. The node N_2 contains the strings $\$\#$ and $\bot\#$ and N_3 contains the words $\bot y$, for all $y \in V^{k+1}$ that do not contain w, that came from N_2 as well as the words $\#\$$, $\bot\$$, and $\bot\#\$$. It is worth noting that in this step the strings $\$y$, for all $y \in V^{k+1}$ that start with w and do not contain the factor w on other positions, left the node N_2 but were not accepted in N_3 so they were lost. In the following splicing step, the configurations of N_1, N_2 and N_4 remain unchanged. In N_3 we obtain the strings the words $\#y$ and y, for all $y \in V^{k+1}$ that do not contain w, and the words $\#\$$, $\bot\$$, and $\bot\#\$$; note that $\#y$ is obtained by splicing $\bot y$ with $\#\$$ using the rule $[(\bot, s), (\#, \$)]$, where s is the first symbol of y, while y was obtained from $\bot y$ and $\#\$$ using the rule $[(\bot, s), (\lambda, \#\$)]$, where s is the first symbol of y. In the next communication step, the configuration of N_2 remains unchanged. The node N_1 contains all the strings $\#x$, where x is an arbitrary string over V of length $k+1$ that does not contain w, which came from N_3, as well as the strings $\bot a\$, \bot b\$$, and $\#\$$; the node N_3 will contain only the words $\#\$$, $\bot\$$, and $\bot\#\$$; the node N_4 contains all the strings over V of length less or equal to $k+1$ that do not contain the factor w, considering that all such strings of length $k+1$ came in this step from N_3. The next step to be executed is the $3k+1^{th}$ splicing step of the computation, and we note that the configuration of the four nodes are exactly as we wanted to prove. Therefore, we have shown that the claim we made at the beginning of this discussion is true. Further, it follows that after $6(k-1)$ computation steps, where $k \geq 1$, the output node N_4 contains all the strings from V^k that do not contain w. It is immediate now that the language generated by $\mathcal{N}$ is L. $\Diamond$

We also consider a variant of networks of splicing processors where the communication is conducted in a different manner, namely generating networks of splicing processors with filtered connections (GNSPFC, for short). See [5] for a closely related accepting model.

Definition 1.3.

(1) A generating network of splicing processors with filtered connections (of size n) is a tuple

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

where

- V is a finite alphabet,
- for $1 \le i \le n$, $N_i = (M_i, A_i)$ where
  - M_i is a set of splicing rules, the rules of node N_i,
  - A_i is a finite subset of V^*, called the set of axioms of node N_i,
- E is a subset of $\{N_1, N_2, \ldots, N_n\} \times \{N_1, N_2, \ldots, N_n\}$, defining the edges of the network,
- $f : E \to 2^V \times 2^V \times \{(\mathsf{s}), (\mathsf{w})\}$ defines the filters on every edge; that is, for an edge (N_i, N_j) with $f((N_i, N_j)) = (P, F, (x))$ we have:

$$\text{edge filter: } \rho_{(N_i,N_j)}(\cdot) = \varphi^{(x)}(\cdot; P, F),$$

- N_o is the output node of the network ($1 \le o \le n$).

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

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

- splicing step (written as $C \Longrightarrow C'$) if, for $1 \le i \le n$,

$$C'(i) = \sigma_{M_i}(C(i)),$$

- communication step (written as $C \vdash C'$) if, for $1 \le i \le n$,

$$C'(i) = \Big(C(i) \setminus \bigcup_{(N_i,N_j)\in E} \rho_{(N_i,N_j)}(C(i))\Big) \cup \bigcup_{(N_j,N_i)\in E} \rho_{(N_j,N_i)}(C(j)).$$

The computation of $\mathcal{N}$ is a sequence of configurations

$$C_t = (C_t(1), C_t(2), \ldots, C_t(n)), \quad t \ge 0,$$

such that

- $C_0 = (A_1, A_2, \ldots, A_n)$,
- C_{2t} derives C_{2t+1} in a splicing step: $C_{2t} \Longrightarrow C_{2t+1}$ ($t \ge 0$),

- C_{2t+1} derives C_{2t+2} in a communication step: $C_{2t+1} \vdash C_{2t+2}$ $(t \geq 0)$.

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

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

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

We stress that both in the case of GNSPs and GNSPFCs the number of nodes of a network is said to be the size of the network.

It is worth noting that the filtering process that takes place during the communication steps is performed by the nodes in the case of GNSPs while in the case of GNSPFCs it is performed by the edges. Also, in the case of GNSPs, a word can be lost during a communication step (if it leaves a node but does not enter any other node), while in the case of GNSPFCs, no word is lost during such a step.

We briefly outline the way GNSPFCs can be used to generate a simple language in the following example.

Example 1.4. As in Example 1.2, let $V = \{a, b\}$ and let $w \in V^*$ be a word over V with $|w| \geq 2$. Once more, take $L = \{x \mid x \in V^*, w \text{ is not a factor of } x\}$. In this case, we construct the network with filtered connections

$$\mathcal{N} = (N_1, N_2, N_3, N_4, N_5, \{(N_1, N_2), (N_2, N_3), (N_2, N_5), (N_3, N_1), (N_3, N_4)\}, f, N_4)$$

depicted in Figure 2.

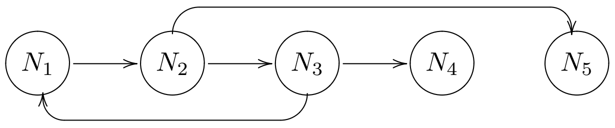

FIGURE 2. Graph of the GNSPFC $\mathcal{N}$ used in Example 1.4

The definitions of the splicing rules and of the axioms of nodes N_1, N_2, N_3, and N_4 is the same as in Example 1.2. However, these nodes do not have filters any more. Further, we define completely the rest of the network.

First, the set of rules and the set of axioms of the node N_5 are empty. Then, the filters placed on edges of the network (i.e., the values of the function f) are defined as follows:

- $f((N_1, N_2)) = (\{\perp\}, \{\$\}, (\mathsf{w}))$
- $f((N_2, N_3)) = (U, \{\#\}, (\mathsf{w}))$
- $f((N_2, N_5)) = (\{\$\}, \{\perp\}, (\mathsf{w}))$
- $f((N_3, N_1)) = (\{\perp\}, U \setminus \{\perp, a, b\}, (\mathsf{w}))$
- $f((N_3, N_4)) = (U, U \setminus V, (\mathsf{w}))$

The computation of this network goes on exactly as that of the GNSP designed in Example 1.2. There is only one difference: in the previous case, the words that left node N_3 and had the form $\$y$, such that y starts with w and the factor w did not occur on any other position of y, were lost during some of the communication steps of the network, and did not influence the computation any more. In this case, no string can be lost. Therefore, we simulate the disappearance of these strings from the computation by sending them to the node N_5 from the node N_2 and trapping them there for the rest of the computation. $\Diamond$

Before moving on to the main results of this paper, note that any GNSP or GNSPFC with exactly one node can be simulated by a network with two nodes. This allows us to assume in the sequel that any network has at least two nodes.

2. Simulation Results

First, we show that all the languages generated by GNSPFCs can be also generated by GNSPs. The result is obtained in a constructive manner: we start with a GNSPFC and give an effective method to construct a GNSP generating the same language.

Theorem 2.1. *For any GNSPFC, a GNSP generating the same language can be constructed.*

Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, f, N_o)$ be a GNSPFC. We may assume without loss of generality that $n \geq 2$ and that $E = \{N_1, \ldots, N_n\} \times \{N_1, \ldots, N_n\}$; indeed, if an edge (N_i, N_j) is not in E we may assume that it is in fact in E but $f((N_i, N_j)) = (V, V, (\mathsf{w}))$. In the following, we construct a GNSP $\mathcal{N}'$, over the same alphabet, that accepts the same language. More precisely, we construct for each node N_i of the network $\mathcal{N}$ a subnetwork $s(N_i)$, contained in the new network $\mathcal{N}'$, that simulates the computation of the processor found in the node N_i and the communication between this node and its neighbours. We denote by $nodes(s(N_i))$ the set of the nodes of the subnetwork $s(N_i)$ and by $edges(s(N_i))$ the set of the edges of the subnetwork.

Let us assume that $N_i = (M_i, A_i)$ is a node of $\mathcal{N}$ and recall that for all the nodes N_t, with $1 \leq t \leq n$, we have $(N_i, N_t) \in E$. Let $f((N_i, N_t)) = (P_t, F_t, \beta_t)$. The network that simulates the computation performed by the node N_i is defined in the following.

The nodes from $nodes(s(N_i))$ are:

- $N'_i = (M_i, A_i, V, \emptyset, V, \emptyset, (\mathsf{w}))$.
- For $k \in \{1, \ldots, n\}$, we have the nodes

$$N_{i,k,1} = (\emptyset, \emptyset, P_k, F_k, V, \emptyset, \beta_k) \text{ and}$$
$$N_{i,k,j} = (\emptyset, \emptyset, V, \emptyset, V, \emptyset, (\mathsf{w})) \text{ for } 2 \leq j \leq n.$$

- For $k \in \{1, \ldots, n\}$:

– If $f((N_i, N_k)) = (\{a_1, \ldots, a_t\}, F, (\mathsf{s}))$, we have $m_k = t + 1$ nodes

$$N'_{i,k,1} = (\emptyset, \emptyset, F, \emptyset, F, \emptyset, (\mathsf{w})) \text{ and}$$
$$N'_{i,k,j+1} = (\emptyset, \emptyset, V, \{a_j\}, V, \emptyset, (\mathsf{w})) \text{ for } j \in \{1, \ldots, t\}.$$

– If $f((N_i, N_k)) = (P, F, (\mathsf{w}))$, we have $m_k = 2$ nodes

$$N'_{i,k,1} = (\emptyset, \emptyset, F, \emptyset, F, \emptyset, (\mathsf{w})) \text{ and}$$
$$N'_{i,k,2} = (\emptyset, \emptyset, V, P, V, \emptyset, (\mathsf{w})).$$

The edges from $edges(s(N_i))$ are:

- for $k \in \{1, \ldots, n\}$, we have

$$(N'_i, N_{i,k,1}),\ (N_{i,k,j}, N_{i,k,j+1}) \text{ for } 1 \le j \le n-1, \text{ and } (N_{i,k,n}, N'_k),$$

- $(N'_i, N'_{i,1,t})$ for $1 \le t \le m_1$,
- for $1 \le k \le n-1$, we have $(N'_{i,k,\ell}, N'_{i,k+1,t})$ for $1 \le \ell \le m_k$, $1 \le t \le m_{k+1}$,
- $(N'_{i,n,t}, N'_i)$ for $1 \le t \le m_n$.

The new network of splicing processors $\mathcal{N}'$ has the nodes $\bigcup_{1 \le i \le n} nodes(s(N_i))$ and the edges $\bigcup_{1 \le i \le n} edges(s(N_i))$. The output node of the new network is N'_o.

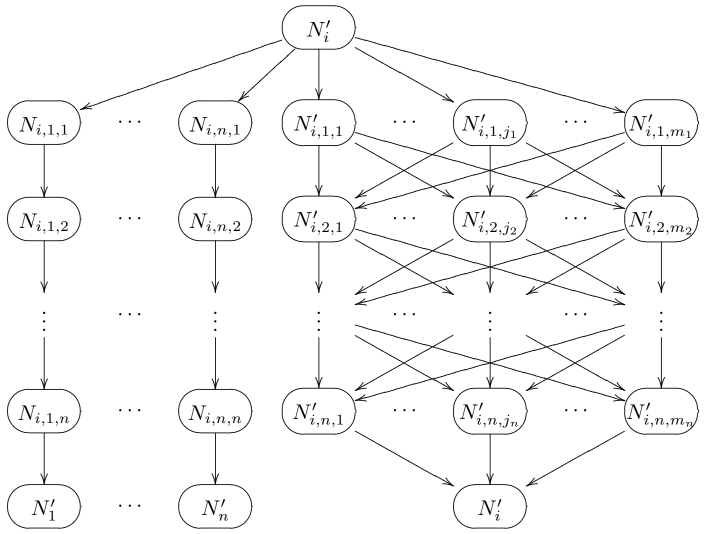

FIGURE 3. Graph of the subnetwork $s(N_i)$, defined in the proof of Theorem 2.1

Further, we describe how the network $\mathcal{N}'$ simulates the computation of the network $\mathcal{N}$. At the beginning of the computation of $\mathcal{N}$, the node N_i contains A_i and no other words for $i \in \{1, \ldots, n\}$. Similarly, the node N'_i contains the words of A_i and no other words in $\mathcal{N}'$ for $i \in \{1, \ldots, n\}$; all the other nodes of $\mathcal{N}'$ are empty.

We will show how a splicing and a communication step of $\mathcal{N}$ are simulated in exactly $n+1$ splicing and $n+1$ communication steps by the network $\mathcal{N}'$. For $k \in \mathbb{N}$ and for all $i \in \{1, \ldots, n\}$, we assume that after $k(n+1)$ splicing and $k(n+1)$ communication steps the nodes N'_i contain exactly the same words as N_i contains after k splicing and k communication steps and all the other nodes of $\mathcal{N}'$ do not contain any word. This assumption clearly holds at the beginning of the computation, that is for $k = 0$. Also, we will show that the words obtained in N'_i after $k(n+1)+1$ splicing steps of $\mathcal{N}'$ are exactly those obtained in N_i in $\mathcal{N}$ after $k+1$ splicing steps.

Let i be a number from $\{1, \ldots, n\}$. In the network $\mathcal{N}$, the splicing rules of M_i are applied on the words contained in the node N_i; this ends the application of the splicing step. Then, in a communication step, all the words of N_i are sent to the node N_j, for $1 \leq j \leq n$, if they can pass the filters on the edge (N_i, N_j); the words that cannot go to any of the neighbours of N_i remain inside this node. This behaviour is simulated in the network $\mathcal{N}'$ as follows. First the rules from M_i are applied to all the words from N'_i. Clearly, the words that are found now in N'_i are exactly those obtained in one splicing step in N_i. Now, all these words exit N'_i. Those that entered N_j in the network $\mathcal{N}$ enter now $N_{i,j,1}$ in $\mathcal{N}'$; now, in exactly n splicing and n communication steps, these words enter N'_j. Further, we must select the words that should return to N'_i from those that just left this node; but these are exactly those words that cannot enter any other node. To this end, from the words that exit N'_i, each word that cannot enter N_1 in $\mathcal{N}$ (because the word cannot pass the filters of (N_i, N_1)) enters in $\mathcal{N}'$ one of the nodes $N'_{i,1,t}$ where $1 \leq t \leq m_1$ and m_1 is defined as above. More precisely:

- If $f((N_i, N_1)) = (P, F, (\mathsf{w}))$ then $m_1 = 1$ and all the words that contain a symbol from F enter $N'_{i,1,1}$ and all the words that do not contain at least one symbol from P enter $N'_{i,1,2}$. The rest of the words are lost (they enter N_1 in $\mathcal{N}$, so they do not remain in N_i, and we should not return them to N'_i, in $\mathcal{N}'$).
- If $f((N_i, N_1)) = (\{a_1, \ldots, a_t\}, F, (\mathsf{s}))$ then $m_1 = t+1$ and all the words that contain a symbol from F enter $N'_{i,1,1}$ and all the words that do not contain a_j enter $N'_{i,1,j+1}$, where $1 \leq j \leq t$. The rest of the words are lost (by the same reason as above).

Now we have selected the words that cannot enter N_1. From them, using in a similar method the nodes $N'_{i,2,j}$ with $1 \leq j \leq m_2 + 1$, we select the words that cannot enter N_2. By iteration, we select the words that cannot enter any of the nodes $N_1, \ldots, N_n$ and return them to N'_i. More precisely, for $k \leq n-1$, the nodes $N'_{i,k,j}$ with $1 \leq j \leq m_k$ select the words that cannot enter N_k (and could not enter $N_1, \ldots, N_{k-1}$) and send them to the nodes $N'_{i,k+1,j}$ with $1 \leq j \leq m_{k+1}$; in the end, the nodes $N'_{i,n,j}$ with $1 \leq j \leq m_n$ select the words that cannot enter N_n (and could not enter $N_1, \ldots, N_{n-1}$) and return them to N'_i. The whole process described above takes n splicing and n communication steps.

As none of the nodes of the network $\mathcal{N}'$ has output filters that can retain words inside a node, it follows immediately from the above explanations that after $(k+1)(n+1)$ splicing and $(k+1)(n+1)$ communication steps of $\mathcal{N}'$ we will have, for all $i \in \{1,\ldots,n\}$, in the node N_i' exactly the words that were obtained after $k+1$ splicing and $k+1$ communication steps in N_i in $\mathcal{N}$; all the others nodes of $\mathcal{N}'$ are empty. Also, in the configuration obtained after exactly $k(n+1)+r$ splicing or communication steps of $\mathcal{N}'$ were executed, with $r < (n+1)$, the nodes N_i' are empty as well.

According to the above, w is present after some time in the node N_o of $\mathcal{N}$ if and only if w is present after some time in the node N_o' of $\mathcal{N}'$.

Therefore, $L(\mathcal{N}) = L(\mathcal{N}')$. □

The next result comes as a completion of the previous theorem and shows that GNSPFCs and GNSPs have actually the same computational power. We stress the fact that the proof of this result as well as the proof of the previous one are based on direct constructions that do not use any intermediate generative models; these proofs seem interesting to us, as they connect directly two related models and show some ideas and methods that may be useful in simulating one massively parallel model by another one.

Theorem 2.2. *For any GNSP, a GNSPFC generating the same language can be constructed.*

Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, N_o)$ be a GNSP. In the following, we construct a GNSPFC $\mathcal{N}'$ that generates the same language. More precisely, we construct for each node N_i of the network $\mathcal{N}$ a subnetwork $s(N_i)$ contained in the new network $\mathcal{N}'$ that simulates the computation of the processor found in the node N_i and the communication between this node and its neighbours. We denote by $nodes(s(N_i))$ the set of the nodes of the subnetwork $s(N_i)$ and by $edges(s(N_i))$ the set of the edges that connect the nodes of this subnetwork and the edges that connect the nodes of $s(N_i)$ with the nodes of other subnetworks.

The new network $\mathcal{N}'$ contains a special node $G = (\emptyset, \emptyset)$, which collects all the words that are produced during the computation of $\mathcal{N}$, in any of its nodes; the main reason why this node exists is to collect the words that were lost during the computation of $\mathcal{N}$ and to ensure that they cannot remain in any other nodes of $\mathcal{N}'$ (as no word can be lost during the computation of this network).

Let us now assume that $N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i, \beta_i)$ is a node of $\mathcal{N}$. The nodes of the subnetwork $s(N_i)$ that simulates the computation by a node N_i are defined as follows (the set of all those nodes is denoted by $nodes(s(N_i))$):

- We set
$$N_i' = (M_i, A_i) \text{ and } N_i'' = (\emptyset, \emptyset).$$
- If $\beta_i = (\mathsf{w})$, we have the nodes
$$N_{i,1}' = N_{i,2}' = (\emptyset, \emptyset).$$

- If $\beta_i = (\mathsf{s})$ and $PO_i = \{a_1, \ldots, a_t\}$, we have the $t+1$ nodes

$$N'_{i,j} = (\emptyset, \emptyset) \text{ for } 1 \leq j \leq t+1.$$

The edges from $edges(s(N_i))$ are the following:

- for $i \in \{1, \ldots, n\}$:

$$(N'_i, N''_i) \text{ with the filter } f((N'_i, N''_i)) = (PO_i, FO_i, \beta_i),$$

- for all i, k with $i, k \in \{1, \ldots, n\}$ and $(N_i, N_k) \in E$:

$$(N''_i, N'_k) \text{ with the filter } f((N''_i, N'_k)) = (PI_k, FI_k, \beta_k),$$

- (N''_i, G) with the filter $f((N''_i, G)) = (V, \emptyset, (\mathsf{w}))$,
- for all i with $i \in \{1, \ldots, n\}$, $\beta_i = (\mathsf{s})$, and $PO_i = \{a_1, \ldots, a_t\}$:
  - $(N'_i, N'_{i,1})$ with the filters $f((N'_i, N_{i,1})) = (FO_i, \emptyset, (\mathsf{w}))$,
  - for $1 \leq \ell \leq t$:

$$(N'_i, N'_{i,\ell+1}) \text{ with the filter } f((N'_i, N'_{i,\ell+1})) = (V, \{a_\ell\}, (\mathsf{w})),$$

  - for $1 \leq j \leq t+1$:

$$(N'_{i,j}, N'_i) \text{ with the filter } f((N'_{i,j}, N'_i)) = (V, \emptyset, (\mathsf{w})),$$

- for all i with $i \in \{1, \ldots, n\}$ and $\beta_i = (\mathsf{w})$:

$(N'_i, N'_{i,1})$ with the filter $f((N'_i, N'_{i,1})) = (FO_i, \emptyset, (\mathsf{w}))$,
$(N'_i, N'_{i,2})$ with the filter $f((N'_i, N'_{i,2})) = (V, PO_i, (\mathsf{w}))$,
$(N'_{i,1}, N'_i)$ and $(N'_{i,2}, N'_i)$
with the filters $f((N'_{i,1}, N'_i)) = f((N'_{i,2}, N'_i)) = (V, \emptyset, (\mathsf{w}))$.

The new network $\mathcal{N}'$ has as nodes the elements of the set

$$\{G\} \cup \bigcup_{1 \leq i \leq n} nodes(s(N_i))$$

and as edges the elements of the set

$$\bigcup_{1 \leq i \leq n} edges(s(N_i)).$$

The function f is defined as above and the output node of the network is N'_o.

We describe how the network $\mathcal{N}'$ simulates the computation of the network $\mathcal{N}$. At the beginning of the computation of $\mathcal{N}$, the node N_i contains A_i and no other words for $i \in \{1, \ldots, n\}$. Similarly, the node N'_i contains the words of A_i and no other words in $\mathcal{N}'$ for $i \in \{1, \ldots, n\}$; all the other nodes of $\mathcal{N}'$ are empty.

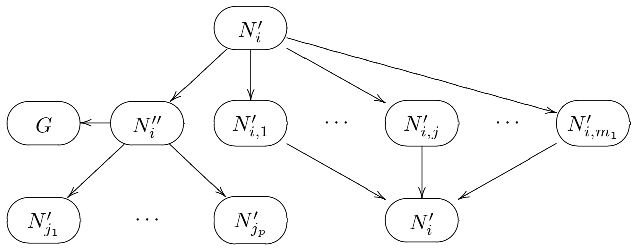

FIGURE 4. Graph of the subnetwork $s(N_i)$ where $\{N_{j_1}, \ldots, N_{j_p}\}$ is the set of neighbours of N_i, defined in the proof of Theorem 2.2

We will show how a splicing and a computation step of $\mathcal{N}$ are simulated in exactly 2 splicing and 2 communication steps by the network $\mathcal{N}'$. For $k \in \mathbb{N}$ and for all $i \in \{1, \ldots, n\}$, we assume that, after $2k$ splicing and $2k$ communication steps, the nodes N'_i of $\mathcal{N}'$ contain exactly the same words as the nodes N_i of $\mathcal{N}$ after k splicing and k communication steps, while all the other nodes of the network, except for G, do not contain any word. The node G contains all the words produced so far during the computation, but this has no effect on the rest of the computation, as there are no edges leaving this node. Also, we will show that the words obtained in N'_i after $2k+1$ splicing steps of $\mathcal{N}'$ are exactly those obtained in N_i in $\mathcal{N}$ after $k+1$ splicing steps. These assumptions clearly hold at the beginning of the computation, for $k = 0$.

Let i be a number from $\{1, \ldots, n\}$. In the network $\mathcal{N}$, the splicing rules of M_i are applied on the words contained in the node N_i; this ends the application of the splicing step. Then, in a communication step, all the words of N_i that can pass its output filters are sent to the nodes N_j such that $(N_i, N_j) \in E$ and, if they can pass the input filters of one of these nodes, they are accepted in it. The words that cannot pass the output filters of N_i remain inside this node. This behaviour is simulated in the network $\mathcal{N}'$ as follows. First the rules from M_i are applied to all the words of the node N'_i. Clearly, the words that are found now in N'_i are exactly those obtained in one splicing step in N_i. Now, the simulation of a communication step begins. All the words that can pass the output filters of N_i enter the node N''_i and from here they are sent to the nodes N'_j with $(N_i, N_j) \in E$. Clearly, only the words that can enter one of these nodes will be further processed by the network; all the others are trapped for the rest of the computation in the node G. The words that cannot pass the output filters of N_i are processed as follows:

- If $N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i, (\mathsf{w}))$ then all the words that contain a symbol from FO_i leave N'_i and enter $N'_{i,1}$ and all the words that contain no symbol of PO_i enter in $N'_{i,2}$.
- If $N_i = (M_i, A_i, PI_i, FI_i, \{a_1, \ldots, a_t\}, FO_i, (\mathsf{s}))$ then all the words that contain a symbol from FO_i leave N'_i and enter $N'_{i,1}$ and all the words that do not contain the symbol a_j enter in $N'_{i,j+1}$.

The words that are accepted in the nodes $N'_{i,j}$ must be returned to N'_i and this actually happens during the next communication step. The whole process described above takes two splicing and two communication steps.

As none of the nodes of the network $\mathcal{N}'$ has output filters that can retain words inside a node, it follows immediately from the above explanations that after $2(k+1)$ splicing and $2(k+1)$ communication steps of $\mathcal{N}'$ we will have in the node N'_i for $i \in \{1,\ldots,n\}$ exactly those words that were obtained after $k+1$ splicing and $k+1$ communication steps in N_i in $\mathcal{N}$; all the others nodes of $\mathcal{N}'$, except for the G node, are empty. Also, in the configuration obtained after exactly $2k+1$ splicing steps of $\mathcal{N}'$ the node N'_i contains the words obtained after $k+1$ splicing steps of $\mathcal{N}$ in N_i. Moreover, after $2k+1$ communication steps of $\mathcal{N}'$ the node N'_i contains no word.

Now it is clear that a word w appears at a given point of the computation of $\mathcal{N}$ in N_o if and only if w appears at a given point of the computation of $\mathcal{N}'$ in N'_o. Thus, $L(\mathcal{N}) = L(\mathcal{N}')$. $\square$

We now give several remarks on the efficiency of the simulations presented above. As far as the descriptional complexity is concerned, one may note that when constructing a GNSP that simulates a GNSPFC the new network has a number of nodes that is proportional to the maximum between the cube of the number of nodes of the initial network and the number of nodes in the initial network times the size of the working alphabet. Therefore, usually there is a big difference between the size of the initial GNSPFC and the size of the constructed GNSP. As Examples 1.2 and 1.4 show, there may be the case when a language is accepted by a GNSP and a GNSPFC whose size differs only by a small amount. This leads us to the open question of whether one can find a more efficient (from the size point of view) procedure of simulating a GNSPFC by a GNSP. When we simulate a GNSP by a GNSPFC, the size of the constructed network is proportional to the number of nodes of the initial network times the size of its alphabet. Although this construction seems more efficient, it is an interesting task to improve it as well.

One may also consider a computational complexity measure. That is, given a GNSP/GNSPFC $\mathcal{N}$ and a word $w \in L(\mathcal{N})$ we may define the measure $nr_{\mathcal{N}}(w)$ as the number of steps performed by the network before w appears for the first time in its output node. We have already shown that in the case of Theorem 2.1, when the GNSPFC $\mathcal{N}$ is simulated by the GNSP $\mathcal{N}'$, we obtain that for each word $w \in L(\mathcal{N}) = L(\mathcal{N}')$ we have $nr'_{\mathcal{N}}(w) \leq (n+1)(nr_{\mathcal{N}}(w))$, where n is the size of $\mathcal{N}$. One may also be interested in improving the computational efficiency of this simulation. This question seems motivated as we note that in the case of Theorem 2.2, when the GNSP $\mathcal{N}$ is simulated by the GNSPFC $\mathcal{N}'$, we obtain that for each word $w \in L(\mathcal{N}) = L(\mathcal{N}')$ we have $nr'_{\mathcal{N}}(w) \leq 2(nr_{\mathcal{N}}(w))$.

To end this section, we show by a somehow similar approach to that in Theorem 2.2 (but with more involved technicalities) a stronger result. A GNSPFC is called uniform if each time it contains an edge (N_i, N_j) then it also contains the edge (N_j, N_i) and the filters on these two edges coincide. Basically, in a uniform

GNSPFC the underlying graph is an undirected graph. If we consider that in fact any GNSPFC can be seen as a network with a complete underlying graph (as the missing edges can be seen as edges that do not allow any communication), we can think of uniform GNSPFCs as networks that have as underlying graph a complete undirected graph; therefore, such networks may be seen as networks that have a normal form.

We can show the following result regarding uniform GNSPFCs.

Theorem 2.3. *For any GNSP, a uniform GNSPFC generating the same language can be constructed.*

Proof. Let $\mathcal{N} = (V, N_1, N_2, \dots, N_n, E, N_o)$ be a GNSP. Let $U = \{\#, \perp, \perp', \$\} \cup V$ be a new alphabet.

First, we define three sets of splicing rules that will become useful in the sequel:

$$R_0 = \{[(\lambda, \#), (\$, \perp)], [(\$, \perp), (\$, \perp)]\},$$
$$R_1 = \{[(\$, \#), (\$, \#)], [(\$, \#), (\lambda, \perp)], [(\lambda, \#), (\perp', \perp')], [(\perp', \perp'), (\perp', \perp')]\},$$
$$R_2 = \{[(\perp', \perp'), (\lambda, \perp)], [(\perp', \perp'), (\perp', \perp')]\}.$$

The network $\mathcal{N}'$ contains a special node $G = (R_2, \{\perp'\perp'\})$.

Let us assume that $N_i = (M_i, A_i, PI_i, FI_i, PO_i, FO_i, \beta_i)$ is a node of $\mathcal{N}$.

The nodes of the network $s(N_i)$, that simulates the computation performed by the node N_i, are defined as follows.

- We set

$$N_i' = (M_i, \{w\# \mid w \in A_i\}),$$
$$N_i'' = (R_0, \{\$\perp\}),$$
$$N_i^{out} = (R_1, \{\perp'\perp', \$\#\}),$$

- if $\beta_i = (\mathsf{w})$, we have the three nodes

$$N_{i,j}' = (R_0, \{\$\perp\}) \text{ for } j \in \{1, 2\} \text{ and } N_i^r = (R_1, \{\perp'\perp', \$\#\}),$$

- if $\beta_i = (\mathsf{s})$ and $PO_i = \{a_1, \dots, a_t\}$, we have the nodes

$$N_{i,j}' = (R_0, \{\$\perp\}) \text{ for } 1 \leq j \leq t+1 \text{ and } N_i^r = (R_1, \{\perp'\perp', \$\#\}).$$

The edges connecting the nodes of these subnetworks are defined in the following. For the rest of the proof, we make the convention that, each time we define an edge (x, y), we also have the edge (y, x) with the same filters. The edges are:

- for $i \in \{1, \dots, n\}$:

$$(N_i', N_i'') \text{ with the filter } f((N_i', N_i'')) = (PO_i, FO_i \cup \{\$, \perp\}, \beta_i),$$
$$(N_i'', N_i^{out}) \text{ with the filter } f((N_i'', N_i^{out})) = (\{\perp\}, \{\$, \perp'\}, (\mathsf{s})),$$
$$(N_i^{out}, G) \text{ with the filters } f((N_i^{out}, G)) = (U, \{\perp', \$, \#\}, (\mathsf{w})).$$

- for all i, k with $(N_i, N_k) \in E$:

$$(N_i^{out}, N_k') \text{ with } f((N_i^{out}, N_k')) = (PI_k, FI_k \cup \{\$, \bot, \bot'\}, \beta_k),$$

- for all i with $i \in \{1, \ldots, n\}$, $\beta_i = (\mathsf{s})$, and $PO_i = \{a_1, \ldots, a_t\}$:
  - $(N_i', N_{i,j}')$ for $1 \leq j \leq t+1$ with the filters

$$f((N_i', N_{i,1}')) = (FO_i, \{\bot, \$\}, (\mathsf{w})) \text{ and}$$
$$f((N_i', N_{i,\ell+1}')) = (U, \{a_\ell, \bot, \$\}, (\mathsf{w})) \text{ for } 1 \leq \ell \leq t,$$

  - $(N_{i,j}', N_i^r)$ for $1 \leq j \leq t+1$ with the filter

$$f((N_{i,j}', N_i^r)) = (\{\bot\}, \{\$, \bot'\}, (\mathsf{s})),$$

  - (N_i^r, N_i') with the filter

$$f((N_i^r, N_i')) = (U, \{\$, \bot, \bot'\}, (\mathsf{w})),$$

- for all i with $i \in \{1, \ldots, n\}$ and $\beta_i = (\mathsf{w})$:
  - $(N_i', N_{i,1}')$ with the filter

$$f((N_i', N_{i,1}')) = (FO_i, \{\bot, \$\}, (\mathsf{w})),$$

  - $(N_i', N_{i,2}')$ with the filter

$$f((N_i', N_{i,2}')) = (U, PO_i \cup \{\bot, \$\}, (\mathsf{w})),$$

  - for $j \in \{1, 2\}$:

$$(N_{i,j}', N_i^r) \text{ with the filter } f((N_{i,j}', N_i^r)) = (\{\bot\}, \{\$, \bot'\}, (\mathsf{s})),$$

  - (N_i^r, N_i') with the filter

$$f((N_i^r, N_i')) = (U, \{\$, \bot, \bot'\}, (\mathsf{w})).$$

For the case of the output node, we add to $\mathcal{N}'$ three more nodes N_o^1, N_o^2, and N_o^3:

$$N_o^1 = (\{[(\lambda, \#), (\bot, \bot')], [(\bot, \bot'), (\bot, \bot')]\}, \{\bot\bot'\}),$$
$$N_o^2 = (\{[(\lambda, \bot), (\lambda, \bot)], [(\bot, \lambda), (\lambda, \bot')]\}, \{\bot\}),$$
$$N_o^3 = (\emptyset, A_o).$$

The edges that connect them to the other nodes are defined as follows:

- (N_i^{out}, N_o^1) for all i such that $(i, o) \in E$ with the filters

$$f((N_i^{out}, N_o^1) = (PI_o, FI_o \cup \{\$, \bot, \bot'\}, \beta_o),$$

- (N_o', N_o^1) with the filters

$$f((N_o', N_o^1) = (\{\#\}, \{\$, \perp, \perp'\}, (\mathsf{w})),$$

- (N_o^1, N_o^2) with the filters

$$f((N_o^1, N_o^2) = (\{\perp'\}, \{\$, \perp, \#\}, (\mathsf{w})),$$

- (N_o^2, N_o^3) with the filters

$$f((N_o^2, N_o^3) = (V, U \setminus V, (\mathsf{w})).$$

The network $\mathcal{N}'$ has as nodes the elements of the set

$$\bigcup_{1 \leq i \leq n} nodes(s(N_i))$$

and as edges the elements of the set

$$\bigcup_{1 \leq i \leq n} edges(s(N_i)).$$

The function f is defined as above and the output node of the network is N_o^3.

We describe how the network $\mathcal{N}'$ simulates the computation of the network $\mathcal{N}$. At the beginning of the computation of the network $\mathcal{N}$, every node N_i with $i \in \{1, \ldots, n\}$ contains the words of the set A_i and no other words. Similarly, every node N_i' contains the words of $\{w\# \mid w \in A_i\}$ and no other words in $\mathcal{N}'$ for $i \in \{1, \ldots, n\}$; all the other nodes of $\mathcal{N}'$ contain only words from $(U \setminus V)^*$, except for N_o^3 that contains all words of the set A_o.

The basic difference between this simulation and the simulation of GNSPs by GNSPFCs is that, in this case, the edges of $\mathcal{N}'$ are basically undirected: each time we have an edge (x, y) in $\mathcal{N}'$, we also have the edge (y, x) in the network, and each time a word goes through the edge in one direction it can also go in the other direction. Due to this aspect, we cannot apply directly the former simulation: there were words that remained unchanged in some splicing steps, thus, it may occur the possibility that they return to the nodes that already processed them. Unfortunately, in this case the network with filtered connections could generate more words than the original network. To overcome this drawback, we use some extra nodes (actually two extra nodes, N_i^{out} and N_i^r, for the simulation of each node N_i of the original network, and another three extra nodes associated with the output node N_o) and the symbols $\{\$, \#, \perp, \perp'\}$.

For simplicity, let us begin with a brief analysis of the way the axioms associated with the output node of the original network are processed in the new network in the nodes N_o^3 and N_o^2. These words are found in N_o^3 at the beginning of the computation. In the first splicing step, they remain unchanged and, then, they go to node N_o^2, where they cannot be changed either. Then they go back to N_o^3 once again and the process continues this way for the rest of the computation. Anyway, they are part of the generated language, as they were once present in N_o^3; this is correct, as A_o was part of the language generated by $\mathcal{N}$ as well.

A key remark in seeing how the rest of the words are processed is to note that if the pair of words (u, v) is obtained in one splicing step from (x, y), then $(u\#, v\#)$ is obtained in one splicing step from $(x\#, y\#)$; the reverse implication holds as well, in the case when the splicing rule contains no occurrence of the # symbol.

We will show how a splicing and a communication step of $\mathcal{N}$ are simulated in exactly three splicing and three communication steps by the network $\mathcal{N}'$.

For $k \in \mathbb{N}$ and for all $i \in \{1, \ldots, n\}$, we assume that after $3k$ simulation and $3k$ communication steps the nodes N_i' contain exactly the words $w\#$, where w is a word that is present in N_i exactly after k splicing and k communication steps. Also, assume that, at this point of the computation, N_i'' contains the word $\$\bot$ and if $k \geq 1$ it also contains $\$\#$; N_i^{out} contains $\bot'\bot'$ and $\$\#$ and may contain some words from $(V \cup \{\$\})^*\bot'$. Similarly, the nodes $N_{i,j}'$ contain the word $\$\bot$ and if $k \geq 1$ they also contain $\$\#$ and N_i^r contains $\bot'\bot'$ and $\$\#$ and may contain some words from $(V \cup \{\$\})^*\bot'$. The node N_o^1 contains $\bot\bot'$ and may contain $\bot\#$, the node N_o^2 contains $\bot$ and may contain some words from V^* (which cannot be spliced with anything) and $\bot\bot'$; the node N_o^3 contains some words from V^* which can be processed as described above. Finally, G contains $\bot'\bot'$ and some other words that end with $\bot'$ and cannot leave this node any longer.

This assumption clearly holds at the beginning of the computation, in the case when $k = 0$.

Let i be a number from $\{1, \ldots, n\}$. In the network $\mathcal{N}$, the splicing rules of M_i are applied to the words contained in the node N_i; this ends the application of the splicing step. Then, in a communication step, all the words of N_i that can pass its output filters are sent to the nodes N_j such that $(N_i, N_j) \in E$ and, if they can pass the input filters of one of these nodes, they are accepted in it; the words that cannot pass the output filters of N_i remain inside this node. This behaviour is simulated in the network $\mathcal{N}'$ as follows. First, the rules from M_i are applied to all the words of the node N_i'. After one splicing step (the $(3k + 1)^{th}$ splicing step of the computation of $\mathcal{N}'$) we obtain in N_i' the words $w\#$ where w is a word obtained after $k + 1$ splicing steps in N_i; the content of the rest of the nodes still fulfils the assumptions made above. Now, the simulation of a communication step begins. All the words $w\#$ leave the node N_i' and can enter the nodes N_j^{out} with $(N_j, N_i) \in E$, if they can pass the filters of that edge, the node N_i'', when w can pass the output filters of N_i, or go to the nodes $N_{i,j}'$, with $1 \leq j \leq m + 1$ where $m = 1$ if $\beta_i = (\mathsf{w})$ and $m = |PO_i|$ if $\beta_i = (\mathsf{s})$.

In the first case, they become $w\bot'$ and are blocked forever in N_j^{out}.

In the second case, when $w\#$ enters N_i'', it becomes $w\bot$, and is sent out by this node; in the same splicing step the words $\$\bot$ and $\$\#$ are obtained, but they will remain blocked in N_i''. The word $w\bot$ cannot go back to N_i' due to the $\bot$ symbol, so it can only go to N_i^{out}. Here we obtain from it the word $w\#$; in the same splicing step the words $\bot'\bot'$, $\$\bot$, $\$\bot'$, $\bot'\#$, and $\$\#$ are also obtained, but they will remain blocked in this node. All the other words contained in this node (words of the form $x\perp'$) are left unchanged. Now, only $w\#$ can exit the node and it can only go to a node N'_j with $(i,j) \in E$ if w can pass the filter defined by PI_j and FI_j; otherwise, it goes to the node G, where it becomes $w\perp$ and is blocked. The simulation above is restarted.

In the third case, $w\#$ enters $N'_{i,1}$ if it contains symbols from FO_i, or in one of the nodes $N'_{i,j}$ with $2 \leq j \leq m+1$ if it does not fulfil the conditions imposed by the permitting output filter PO_i. Then, the word is processed exactly as in the second case above and is finally returned to N'_i for the simulation of a new splicing step of $\mathcal{N}$.

From the explanations above and the form of the splicing rules, axioms, and filters of the networks, it is rather clear that after $3(k+1)$ splicing steps and $3(k+1)$ communication steps the assumptions made above hold once more. Moreover, for all positive integers k, the configuration of N'_i after $3k$ splicing and $3k$ communication steps of $\mathcal{N}'$ contains the word $w\#$ if and only if w is contained in the configuration of N_i after k splicing and k communication steps of $\mathcal{N}$; in the rest of the time, the configuration of N'_i is empty.

A special case occurs when the simulation of the output node is performed. In this case, each word $w\#$ that leaves N'_o or enters N'_o goes also to N^1_o. Here it becomes $w\perp'$ and can only go the N^2_o. Further, it is transformed into w and sent to N^3_o. It is clear now that w appears in N^3_o if and only if w is accepted by $\mathcal{N}$; no other words appear in N^3_o.

According to the above, $L(\mathcal{N}) = L(\mathcal{N}')$. □

3. Computational Completeness

Once we have shown that GNSPs, GNSPFCs, and uniform GNSPFCs have the same computational power, it seems interesting to us to characterize precisely this power. The following result answers this question, in an expected manner.

Theorem 3.1. *For any recursive enumerable language L, there exists a GNSP $\mathcal{N}$ with $L(\mathcal{N}) = L$.*

Proof. In the construction given below, we essentially follow the idea of the proof of Lemma 7.16 in [19], i. e., we use markers in front and behind the word itself (these markers ensure that the splicing rules can only be applied at the beginning or the end of the current word), rotate the word inside the markers where an additional letter B marks where the beginning of the non-rotated word is, and simulate the application of a rule replacing a subword by another one only at the end of the word. The difference is that we ensure the order of applications by the network whereas this is done in [19] by (an infinite set of) splicing rules with components of arbitrary length.

Let $G = (N, T, P, S)$ be a phrase structure grammar such that $L(G) = L$. Let X, Y, Z, B, C, D be additional letters and $N \cup T \cup \{B\} = \{x_1, x_2, \ldots, x_n\}$. We construct the GNSP

$$\mathcal{N} = (V, N_0, N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, E, N_8),$$

where E is given by the graph presented in Figure 5 and the other components

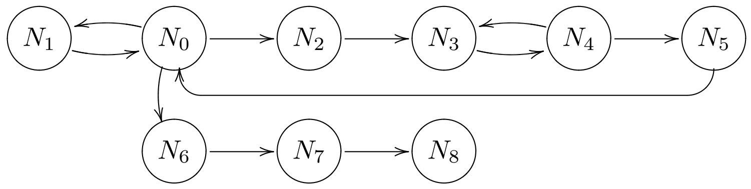
$$\begin{tikzcd}
N_1 \arrow[r, bend right] & N_0 \arrow[l, bend right] \arrow[r] \arrow[d] & N_2 \arrow[r] & N_3 \arrow[r, bend right] & N_4 \arrow[l, bend right] \arrow[r] & N_5 \arrow[llll, bend left] \\
 & N_6 \arrow[r] & N_7 \arrow[r] & N_8 & &
\end{tikzcd}$$

FIGURE 5. Graph of the Network $\mathcal{N}$ used in the proof of Theorem 3.1

are defined as follows:

$$V = \{X, Y, Z, B, C, D\} \cup N \cup T,$$
$$N_0 = (\emptyset, \{XBSY\}, N \cup T, \emptyset, \emptyset, \emptyset, (\mathsf{s}))$$

(N_0 performs no changes during a splicing step; it only distributes the words to N_1, where a simulation of a derivation step of G is done, or to N_2, where a rotation is started, or to N_6, where termination is started; moreover, it contains the word corresponding to the axiom of G; and at any moment, it only contains words of the form Xw_1Bw_2Y for some $w_1, w_2 \in (N \cup T)^*$),

$$N_1 = (M_1, A_1, \emptyset, \emptyset, \{X, Y\}, \emptyset, (\mathsf{s})) \text{ with}$$
$$M_1 = \{[(\lambda, uY)(Z, vY)] \mid u \to v \in P\} \cup \{[(Z, uY), (Z, vY)] \mid u \to v \in P\},$$
$$A_1 = \{ZuY \mid u \to v \in P\} \cup \{ZvY \mid u \to v \in P\}$$

(if XwY coming from N_0 has a suffix uY such that there is a rule $u \to v \in P$, then we can apply a rule $[(\lambda, uY)(Z, vY)]$ to it and a word $ZuY \in A_1$ which produces the words $Xw'vY$ with $w'u = w$, which is sent back to N_0 where it also arrives, and the word ZuY which remains in N_1 and is in A_1; if XwY does not have a suffix uY such that there is a rule $u \to v \in P$, then no rule can be applied to it and it is sent without change to N_0; moreover, by applications of $[(Z, uY), (Z, vY)]$ to words of A_1 we can reproduce the words in A_1, i.e., the words of A_1 are in N_1 at any moment),

$$N_2 = (M_2, \{ZC^iY \mid 1 \leq i \leq n\}, \emptyset, \emptyset, \{X, Y\}, \emptyset, (\mathsf{s})) \text{ with}$$
$$M_2 = \{[(\lambda, x_iY), (Z, C^iY)], [(Z, C^iY), (Z, C^iY)] \mid 1 \leq i \leq n\}$$

(by the second type of rules of N_2, the words $ZC^iY \in A_2$ are reproduced and cannot leave N_2, therefore they are in N_2 at any moment; if a word coming from N_0 has the form Xwx_iY, then we apply a splicing rule of the first type and get XwC^iY, which is sent to N_3, and Zx_iY, which remains in N_2; to the latter word we can apply the first splicing rule which leads to Zx_iY and ZC^iY, again, i.e., these words remain in N_2 for ever),

$$N_3 = (\{[(X,\lambda),(XD,Z)],[(XD,Z),(XD,Z)]\},\{XDZ\},\{C\},\emptyset,\emptyset,\{Z\},(\mathsf{s})),$$
$$N_4 = (\{[(\lambda,CY),(Z,Y)],[(Z,Y),(Z,Y)]\},\{ZY\},\{C\},\emptyset,\emptyset,\{Z\},(\mathsf{s}))$$

(by the second rules of N_3 and N_4, the words XDZ and ZY are reproduced and cannot leave N_3 and N_4, respectively, therefore they are in N_3 and N_4 at any moment; the word coming from N_2 into N_3 has the form XwC^iY, we apply the first splicing rule to XwC^iY and XDZ and get $XDwC^iY$, which is sent to N_4, and XZ, which stays in N_3 and produces XZ and XDZ, again, and thus for ever; if the word comes from N_4 into N_3, it has the form XD^pwC^qY and as above we get $XD^{p+1}wC^qY$, which is sent to N_4, and XZ; if the word comes from N_3 into N_4, it has the form XD^pwC^qY, $q \geq 1$ and we produce $XD^pwC^{q-1}Y$ and ZCY, which stays in N_4 and leads to ZCY and ZY only; if $q-1 \geq 1$, then the word $XD^pwC^{q-1}Y$ leaves N_4 and enters N_3, and if $q-1=0$, then it leaves N_4 and enters N_5; since any sequence N_3N_4 introduces a D and cancels C, the word sent to N_5 has the form XD^iwY if Xwx_iY was sent from N_0 to N_2),

$$N_5 = (M_5,\{Xx_iZ \mid 1 \leq i \leq n\},\emptyset,\{C\},\emptyset,\{D,Z\},(\mathsf{s})) \text{ with}$$
$$M_5 = \{[(XD^i,\lambda),(Xx_i,Z)],[(Xx_i,Z),(Xx_i,Z)] \mid 1 \leq i \leq n\}$$

(again, the initial words Xx_iZ are reproduced; to the word XD^iwY and Xx_jZ with $j \leq i$ we can apply the splicing rule $[(XD^j,\lambda),(Xx_j,Z)]$; if $i=j$, we obtain Xx_iwY, which is sent to N_0 and it enters N_0, and XD^iZ, which remains in N_5; thus we have performed a rotation since we started from Xwx_iY and finished with Xx_iwY in N_0; if $j<i$, we obtain $Xx_jD^{j-i}wY$ and XD^jZ, which both remain in N_5; no rule can be applied to the words $Xx_jD^{j-i}wY$ such that these words remain unchanged in N_5 for ever; from the words of the form XD^jZ only words of the form $Xx_kD^{j-k}Z$ and XD^kZ can be generated, which remain in N_5),

$$N_6 = (\{[(\lambda,ZY),(XB,\lambda)],[(Z,Y),(Z,Y)]\},\{ZY\},\emptyset,\emptyset,\{Y\},\{B\},(\mathsf{s}))$$

(by the second rule of N_6, the word ZY is reproduced and cannot leave N_6, therefore it is in N_6 at any moment; if a word coming from N_0 does not start with XB, then no splicing rule of N_6 can be applied to it, and by $FO_6 = \{B\}$, it remains for ever in N_1; if $XBwY$ comes from N_0, then we apply the first rule to it and ZY, which gives wY, which is sent to N_7, and $XBZY$, which remains in N_6, by the splicing rules it is reproduced and gives λ in addition; therefore $XBZY$ and λ stay in N_6 for ever),

$$N_7 = (\{[(\lambda,Y),(XZ,\lambda)],[(X,Z),(X,Z)]\},\{XZ\},\emptyset,\emptyset,\emptyset,\{X,Z\}\cup N,(\mathsf{s}))$$

(by the second rule of N_7, the word XZ is reproduced and cannot leave N_7, therefore it is in N_7 at any moment; the words coming from N_6 have the form wY and applying the first rule to it and XZ gives w, which is sent to N_8, and XZY, which is reproduced and gives the empty word in addition; words over $N \cup T$ containing at least one non-terminal stay in N_7 and cannot be changed by rules of N_7),

$$N_8 = (\emptyset, \emptyset, PI_8, N, \emptyset, \emptyset, (\mathsf{w}))$$

where PI_8 is the empty set, if the empty word belongs to L, and T otherwise (it is obvious that only words over T arrive in N_8).

First, we show that, for any sentential form z of G, there is a moment t such that N_0 contains at moment t the word $XBzY$. Let $z = y_1uy_2 \Longrightarrow y_1vy_2$ in G by a rule $u \to v \in P$. By the above explanations, we first rotate the word such that we get Xy_2By_1uY, then we send the word to N_1 where it is transformed into Xy_2By_1vY, and by further rotations we obtain XBy_1vy_2Y. Thus the statement holds.

Conversely, if Xz_1Bz_2Y is contained in N_0 at some moment, then z_2z_1 is a sentential form of G. This follows easily by induction because each cycle $N_0N_1N_0$ and $N_0N_2(N_3N_4)^tN_5N_0$, $t \geq 1$, starting with Xz_1Bz_2Y where z_2z_1 is a sentential form ends with $Xz_1'Bz_2'Y$ such that $z_2z_1 \Longrightarrow z_2'z_1'$ in G or $z_2z_1 = z_2'z_1'$.

Moreover, since only words $XBwY$ with $w \in T^*$ can be processed by N_6, N_7 in succession, we can get in N_8 only and all terminal sentential forms of G.

Thus, $L(G) = L(\mathcal{N})$. $\square$

From this completeness result together with the Theorems 2.2 and 2.3, we also obtain the following result.

Corollary 3.2. *For any recursively enumerable language L, there is a (uniform) network $\mathcal{N}$ of splicing processors with filtered connections such that $L(\mathcal{N}) = L$.*

Following the lines of the discussion started in the previous section about the computational efficiency of our simulations, one may note that the network constructed in the proof of Theorem 3.1 makes, in order to generate a word w, a number of steps that is proportional to the square of the number of steps made by the simulated grammar in order to generate w. Indeed, at each step, we first search in the sentential form (by the so-called rotations) the place where the rule should be applied; this takes a number of steps linear in the length of the sentential form, which is less than a constant number (depending on the productions of the grammar) times the number of steps made by the grammar to generate it. Then we just apply the rule. It seems interesting to us to investigate the existence of more efficient simulations.

4. Conclusions

In this paper, we considered a new formal languages generating model, the generating networks of splicing processors. We have shown that such networks with nine processors can be used to generate all the recursively enumerable languages. Moreover, we have shown, by direct simulations of GNSPs, that two variants of networks of splicing processors, namely networks with filtered connections and uniform networks with filtered connections, are also computationally complete.

Besides the open problems highlighted after Theorem 2.1 and at the end of the last section, several directions in which this work can be continued seem of interest to us. Similarly to the case of GNSPFCs, a GNSP is called uniform if the input and output filters of each node coincide. We think that it is worth investigating what the computational power of this model is and to find methods to simulate the non-uniform GNSPs by it. Moreover, our simulation methods can be used to obtain efficient simulations between different types of accepting networks of splicing processors (see [5]).

In most of our proofs, we used both strong and weak filters; for instance, the proof of Theorem 3.1 requires only one node working with weak filters and it is easy to show that it can be replaced by $t+1$ strong filters where t is the number of letters in the terminal alphabet. We are interested to see whether there is a possibility to get results similar to the ones we presented but with using only one type of filtering. To this end, it may also be interesting to investigate the relationship between the class of languages generated by networks of a certain type that only have strong filters and the class of languages generated by networks of the same type that only have weak filters.

Finally, we think that it may be an appealing subject to consider networks of splicing processors with filters from different subclasses of regular languages (not only filters based on random context conditions), as it was done in the case of networks of evolutionary processors [11], and to see what classes of languages can be generated by such networks.

5. Acknowledgements

The authors would like to thank the anonymous referees for their comments and suggestions that led to a better presentation of this paper.

Florin Manea's work is currently supported by the *DFG – Deutsche Forschungsgemeinschaft* grant 582014. Also, Florin Manea acknowledges the support of the *Alexander von Humboldt Foundation*, through a two-years Research Fellowship at the Otto-von-Guericke University Magdeburg, Germany, that ended in June 2011, and a subsequent Return Fellowship at the University of Bucharest, Romania, between July 2011 and October 2011.

References

[1] Cezara Drăgoi and Florin Manea. On the Descriptional Complexity of Accepting Networks of Evolutionary Processors with Filtered Connections. *Int. J. Found. Comput. Sci.*, 19(5):1113–1132, 2008.

[2] Cezara Drăgoi, Florin Manea, and Victor Mitrana. Accepting Networks of Evolutionary Processors with Filtered Connections. *J. UCS*, 13(11):1598–1614, 2007.

[3] Artiom Alhazov, Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, and Yurii Rogozhin. About Universal Hybrid Networks of Evolutionary Processors of Small Size. In *Proc. LATA 2008, LNCS 5196*, pages 28–39. Springer-Verlag, 2008.

[4] Artiom Alhazov, Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, and Yurii Rogozhin. On the size of computationally complete hybrid networks of evolutionary processors. *Theor. Comput. Sci.*, 410(35):3188–3197, 2009.

[5] Juan Castellanos, Florin Manea, Luis Fernando de Mingo López, and Victor Mitrana. Accepting Networks of Splicing Processors with Filtered Connections. In *Proc. MCU 2007, LNCS 4664*, pages 218–229. Springer-Verlag, 2007.

[6] Juan Castellanos, Carlos Martin-Vide, Victor Mitrana, and Jose M. Sempere. Solving NP-Complete Problems With Networks of Evolutionary Processors. In *Proc. IWANN 2001, LNCS 2084*, pages 621–628. Springer-Verlag, 2001.

[7] Juan Castellanos, Carlos Martín-Vide, Victor Mitrana, and José M. Sempere. Networks of Evolutionary Processors. *Acta Inf.*, 39(6-7):517–529, 2003.

[8] Erzsébet Csuhaj-Varjú, Lila Kari, and Gheorghe Paun. Test Tube Distributed Systems Based on Splicing. *Computers and Artificial Intelligence*, 15(2-3), 1996.

[9] Erzsébet Csuhaj-Varjú and Arto Salomaa. Networks of Parallel Language Processors. In *New Trends in Formal Languages – Control, Cooperation, and Combinatorics, LNCS 1218*, pages 299–318. Springer-Verlag, 1997.

[10] Erzsébet Csuhaj-Varjú and Sergey Verlan. On length-separating test tube systems. *Natural Computing*, 7(2):167–181, 2008.

[11] Jürgen Dassow, Florin Manea, and Bianca Truthe. Networks of evolutionary processors with subregular filters. In *Proc. LATA 2011, LNCS 6638*, pages 262–273. Springer-Verlag, 2011.

[12] Remco Loos. On accepting networks of splicing processors of size 3. In *Proc. CiE, LNCS 4497*, pages 497–506. Springer-Verlag, 2007.

[13] Remco Loos, Florin Manea, and Victor Mitrana. On Small, Reduced, and Fast Universal Accepting Networks of Splicing Processors. *Theor. Comput. Sci.*, 410(4-5):406–416, 2009.

[14] Florin Manea, Carlos Martín-Vide, and Victor Mitrana. Accepting Networks of Splicing Processors. In *Proc. CiE 2005, LNCS 3526*, pages 300–309. Springer-Verlag, 2005.

[15] Florin Manea, Carlos Martín-Vide, and Victor Mitrana. Accepting Networks of Splicing Processors: Complexity Results. *Theor. Comput. Sci.*, 371(1-2):72–82, 2007.

[16] Florin Manea, Carlos Martín-Vide, and Victor Mitrana. Accepting networks of evolutionary word and picture processors: A survey. In Carlos Martín-Vide, editor, *Scientific Applications of Language Methods*, volume 2 of *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, pages 525–560. World Scientific, 2010.

[17] Maurice Margenstern, Victor Mitrana, and Mario J. Pérez-Jiménez. Accepting Hybrid Networks of Evolutionary Processors. In *Proc. DNA 2004, LNCS 3384*, pages 235–246. Springer-Verlag, 2004.

[18] Carlos Martín-Vide and Victor Mitrana. Networks of evolutionary processors: Results and perspectives. In *Molecular Computational Models: Unconventional Approaches*, pages 78–114, 2005.

[19] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *DNA computing – new computing paradigms*. Texts in theoretical computer science. Springer, 1998.

[20] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.