Simulations between Three Types of Networks of Splicing Processors

José Ramón Sánchez Couso, José Angel Sanchez Martín, Victor Mitrana, and Mihaela Păun

Abstract: Networks of splicing processors (NSP for short) embody a subcategory among the new computational models inspired by natural phenomena with theoretical potential to handle unsolvable problems efficiently. Current literature considers three variants in the context of networks managed by random-context filters. Despite the divergences on system complexity and control degree over the filters, the three variants were proved to hold the same computational power through the simulations of two computationally complete systems: Turing machines and 2-tag systems. However, the conversion between the three models by means of a Turing machine is unattainable because of the huge computational costs incurred. This research paper addresses this issue with the proposal of direct and efficient simulations between the aforementioned paradigms. The information about the nodes and edges (i.e., splicing rules, random-context filters, and connections between nodes) composing any network of splicing processors belonging to one of the three categories is used to design equivalent networks working under the other two models. We demonstrate that these new networks are able to replicate any computational step performed by the original network in a constant number of computational steps and, consequently, we prove that any outcome achieved by the original architecture can be accomplished by the constructed architectures without worsening the time complexity.

Keywords: splicing processor; network of splicing processors; network of uniform splicing processors; network of splicing processors with filtered connections; theory of computation; computational models; Turing machine

1. Introduction

In recent years, the limitations encountered by our standard computational models have become more apparent, motivating the pursuit of knowledge regarding innovative paradigms capable of overcoming these barriers. Along this line of research, different computational models finding their inspiration on natural phenomena have been proposed in the scientific community. Further research showed their theoretical proficiency for solving intractable problems.

Networks of bio-inspired processors are one of such nature-inspired systems under the parallel and distributed computational framework, boasting a high degree of parallelism (theoretically, unlimited). This model shares numerous characteristics with other known computational paradigms of diverse origins: tissue-like P systems [1], which is a model inspired by the tissue structure introduced in the area of membrane computing [2], a model, called evolutionary systems, that is inspired by the evolution of various cell populations [3], networks of parallel language processors, which is a parallel model for generating languages [4], flow-based programming, which is a particular variant of the data-flow programming [5].
connection machine, which is a kind of a parallel non-conventional processing computer [6], distributed computing using mobile programs [7], etc.

A network of bio-inspired processors may be informally described as a set of processors that are placed in the vertices of a graph. Each processor acts over data organized in distinct structures: strings, pictures, graphs, multisets, etc. An early survey may be found in [8] and more recently in [9]. Thus far, the literature differentiates between two subcategories of networks of bio-inspired processors handling string data, namely evolutionary and splicing processors. The first one was proposed in [10] as a logical abstraction of the point mutations taking place in DNA sequences: insertion, deletion or substitution of a single base pair. The second category, introduced in [11] and analyzed for its theoretical application for problem solving in [12], mimics the splicing phenomena in DNA molecules overseen by the various restriction and ligase enzymes [13] from a computational point of view. A more general view of restriction enzymes as sources of inspiration for computing can be seen in [14], which opened a wide area of research.

Under this paradigm, the strings hosted in a splicing processor perform the role of DNA molecules, while the behavioral constraints for the splicing enzymes are abstracted to a set of splicing rules composed by quadruples of strings specifying the cutting positions of the strings in the node. Following a splicing operation, the pieces are recombined as long as they are yielded by the same splicing rule. This model shares common aspects with the test tube distributed splicing systems introduced in [15] and posteriorly examined in [16]. The divergences with this last splicing paradigm and time-varying distributed H systems [17] are discussed in [18].

The computation in the networks of splicing processors follows after a sequence of alternative steps, namely splicing and communication, that continues indefinitely until a predefined halting requirement is met. In the splicing step, the strings in each node are simultaneously cut and recombined into new ones according to the constraints set by all the applicable splicing rules of the processor hosted in that node. Thus, each string of the node has enough replications to allow for the synchronous completion of the relevant rules. Next in line, a communication step redistributes the string data in the network as follows:

(i) The string data in a splicing processor is sent to the connected nodes provided that it meets the established exiting conditions for that node.

(ii) The receiving nodes concurrently accept the incoming data satisfying its input requirements. Any string refused by all these nodes is consequently lost.

This data flow in the network is managed through the introduction of random-context filters deciding upon the entering and exiting requirements of the nodes through semantic or syntactical conditions. Thus, far, there are three variants of the latter category: (i) each node is assigned an input and output filter to manage the incoming and exiting string data, respectively, [19], (ii) these two filters are combined into one unique filter [20] and (iii) the filters of two connected nodes are replaced with ones assigned to the edge between them [21].

A simulation of a model by another one is said to preserve the time complexity if the numbers of computational steps performed by the two models on every input are of the same order. It is obvious that (ii) and (iii) are particular variants of (i). Consequently, they appear to offer less control over the computation; however, several works proved they share the same computational power through efficient simulations of known computationally complete models in [11,22,23].

Nevertheless, the indirect conversion between the previous variants through the intermission of a Turing machine is unattainable because of the enormous spike in the complexity. This paper aims to improve upon this line of research with the proposal of efficient direct simulations between the three discussed variants.

2. Basic Definitions

We describe here the basic definitions and nomenclature used for this research. For the remaining notations related to this work, the reader is encouraged to consult [24].
An alphabet refers to a finite and non-empty set of symbols. We write the cardinality of a given finite set \( S \) as \( \text{card}(S) \). A string over an alphabet \( V \) is a finite sequence of symbols belonging to that alphabet \( V \). The set of all strings over the alphabet \( V \) is denoted by \( V^* \), the empty string is expressed by \( \epsilon \), and the length of a string \( z \) is written \( |z| \). Moreover, \( \text{alph}(z) \) is the smallest set \( U \subseteq V \) such that \( z \in U^* \) holds.

A splicing rule over \( V \) is a 4-tuple of strings following the structure \( (u_1, u_2); (u_3, u_4) \), with \( u_1, u_2, u_3, u_4 \in V^* \). Given such a splicing rule \( r = (u_1, u_2); (u_3, u_4) \) and the strings \( \alpha, \beta \in V^* \), we define

\[
\sigma_r(\alpha, \beta) = \begin{cases} 
\beta_1 u_3 u_2 a_2 & |a = \alpha_1 u_1 u_2 a_2, \beta = \beta_1 u_3 u_4 a_2, \alpha_1, a_2, \beta_1, \beta_2 \in V^* \} 
\cup 
\{ \alpha_1 u_1 u_4 \beta_2 & |a = \alpha_1 u_1 u_2 a_2, \beta = \beta_1 u_3 u_4 a_2, \alpha_1, a_2, \beta_1, \beta_2 \in V^* \} 
\end{cases}.
\]

This definition may be extended as follows. Given a language \( L \) over \( V \) and a finite set of splicing rules \( R \), we define

\[
\sigma_R(L) = \bigcup_{r \in R} \left( \bigcup_{w_1, w_2 \in L} \sigma_r(w_1, w_2) \right).
\]

Given an alphabet \( V \), two disjoint subsets \( P, F \) of \( V \), and a string \( z \) over \( V \), we define the following predicates:

\[
\phi^{(s)}(z; P, F) \equiv P \subseteq \text{alph}(z) \land F \neq \emptyset \land \text{alph}(z) = \emptyset \\
\phi^{(w)}(z; P, F) \equiv \text{alph}(z) \cap P = \emptyset \land F \cap \text{alph}(z) = \emptyset.
\]

In these predicates, the set of permitting contexts \( P \) defines the symbols that are required to be present in the current string while the set of forbidding contexts \( F \) refers to those symbols that are banned. Both clauses require the nonexistence of symbols \( a \in F \) in the string \( z \) and differ on the flexibility of the conditions related to \( P \). The first one demands for all the symbols in \( P \) to exist in \( z \), while the latter statement is met as long as one or more of the symbols in \( P \) belong to \( z \).

Given \( \beta \in \{w, s\} \) and a language \( L \subseteq V^* \), we write

\[
\phi^\beta(L, P, F) = w \in L \mid \phi^\beta(w, P, F).
\]

A splicing processor with random-context filters (SP) over an alphabet \( V \) is a 6-tuple \((S, A, PI, FI, PO, FO)\), where:

- \( S \) is a finite set of splicing rules over \( V \).
- \( A \) is a finite set of strings over \( V \). Each string in \( A \) is called an auxiliary string.
- \( PI \) (permitting symbols) and \( FI \) (forbidding symbols) are two subsets of \( V \), which both define the input filter of the processor.
- \( PO, FO \subseteq V \) are similar subsets of \( V \) that define the output filter of the processor.

We say a splicing processor is uniform if the input and output filters are identical: \( PI = PO = P \) and \( FI = FO = F \). The set of all splicing processors over the alphabet \( U \) is denoted by \( \text{SP}_U \), while \( \text{USP}_U \) denotes the set of all uniform splicing processors over \( U \).

A network of splicing processors (NSP) is a 9-tuple \( \Gamma = (V, U, (\cdot, \cdot), G, N, \beta, \text{In}, \text{Halt}) \), where:

- \( V \subset U \) are the input and network alphabet, respectively. The working alphabet \( U \) contains two special symbols, namely, \((\cdot, \cdot)\) that do not belong to \( V \).
- \( G = (X_G, E_G) \) is an undirected graph defined by the set \( X_G \) of vertices and the set \( E_G \) of edges. As the graph does not contain loops, we define an edge by a binary set of vertices. \( G \) is called the underlying graph of the network. It is worth mentioning that many papers were dealing with NSP with a complete underlying graph, see, e.g., the survey [9].
• \( \mathcal{N} : X_G \rightarrow SP_U \) is a function that associates with each vertex \( x \in X_G \) the splicing processor \( \mathcal{N}(x) = (S_x, A_x, \Pi_x, F_l_x, F_o_x, P_O_x, F_O_x) \).

• \( \beta : X_G \rightarrow \{(s),(w)\} \) is a function that associates with each vertex the type of both its input and output filters. We now define two mappings on the set of all strings over \( U \setminus \{(\cdot)\} \):

  \[
  \text{input filter: } \rho_s(.) = \varphi^\beta(x)(.; \Pi_x, F_l_x).
  \]

  \[
  \text{output filter: } \tau_s(.) = \varphi^\beta(x)(.; P_O_x, F_O_x).
  \]

Informally, \( \rho_s(z) \) (resp. \( \tau_s(z) \)) decides whether or not the string \( z \) can pass the input (resp. output) filter of \( x \). If \( L \) is a language, we define \( \rho_s(L) \) by: \( \rho_s(L) = \{ w \in (U \setminus \{(\cdot)\})^* \mid \rho_s(w) \} \), that is the set of strings of \( L \) that can pass the input filter of \( x \). In an analogous way, we define \( \tau_s(L) \). Note that though we use the same notation for \( \rho_s(w) \) (\( \tau_s(w) \)) and \( \rho_s(L) \) (\( \tau_s(L) \)), there is no confusion because the arguments are different.

• In, Haltn ∈ X_G are the input and the halting node of \( \Gamma \), respectively.

Likewise, a network of splicing processors is considered to be uniform if it is composed by uniform splicing processors. A network of uniform splicing processors (NSP) is a 9-tuple \( \Gamma = (V, U, \langle \cdot \rangle, G, \mathcal{N}, \beta, \text{In, Haltn}) \), where:

• \( V, U, \langle \cdot \rangle, G, \text{In, Haltn} \) follow the same specifications as the parameters in NSP.

• \( \mathcal{N} : X_G \rightarrow USP_U \) is a function that returns the uniform splicing processor \( \mathcal{N}(x) = (S_x, A_x, \Pi_x, P_O_x, F_O_x) \) associated with the node \( x \in X_G \).

• \( \beta : X_G \rightarrow \{(s),(w)\} \) specifies the strength of the sets \( P \) and \( F \) associated to node filters.

A network of splicing processors with filtered connections (NSPFC) is a 9-tuple \( \Gamma = (V, U, \langle \cdot \rangle, G, \mathcal{N}, \beta, \text{In, Haltn}) \), where:

• \( V, U, \langle \cdot \rangle, \text{In, Haltn} \) follow the same specifications as the parameters in NSP.

• \( G = (X_G, E_G) \) is also an undirected graph such that each node \( x \in X_G \) is seen as a splicing processor without filters, with the set of splicing rules \( S_x \) and the set of axioms \( A_x \).

• \( \mathcal{N} : E_G \rightarrow 2^U \times 2^U \) associates with each edge \( e \in E_G \) a pair of sets \( P_e, F_e \) that define the filter on the edge \( e \).

• \( \beta : E_G \rightarrow \{(s),(w)\} \) defines the predicate variant assigned to the edge filters.

The size of a NSP \( \Gamma \) belonging to any of the variants above is defined as the number of nodes in the graph, i.e., \( \text{card}(X_G) \). A configuration of \( \Gamma \) is a mapping \( C : X_G \rightarrow 2^U \), which assigns a set of strings \( C(x) \) to each node \( x \in \Gamma \), that is the sets of strings that can be found in node \( x \) at a given moment. Although for each \( x \in X_G \), \( C(x) \) is actually a multiset of strings, each string appearing in an arbitrary number of copies, for the sake of simplicity, we work with the support of this multiset. For a string \( z \in V^* \), the initial configuration of \( \Gamma \) on the input string \( z \) is \( C_0(\text{In}) = \{(z)\} \) and \( C_0(x) = \emptyset \) for all \( x \in X_G \setminus \text{In} \).

A configuration can be altered through a splicing step or a communication step. In a splicing step, all the splicing rules belonging to the set \( S_x \) applicable to the strings in the set combination of \( C(x) \) and \( A_x \) are realized, causing the change of \( C \) to a new configuration \( C' \). Formally, a configuration \( C \) evolves to \( C' \) by a splicing step, denoted as \( C \Rightarrow C' \), if and only if the following proposition is true for all \( x \in X_G \):

\[
C'(x) = \sigma_{S_x}(C(x) \cup A_x).
\]

The communication step is different in NSP (NSPUS) and NSPFC. We first define how a communication step works in a NSP (the definition of a communication step in a NUSP is very similar, hence left to the reader). In every node \( x \), all the following tasks are accomplished simultaneously:
(i) The strings that satisfy the output filter condition of \( x \) are sent out;
(ii) Copies of the expelled strings from any node \( y \) connected to \( x \) enter \( x \), provided that they satisfy the input filter condition of \( x \).

We stress that those strings sent out of \( x \) that do not satisfy the input filter condition of any node are definitely lost. Formally, a configuration \( C' \) follows a configuration \( C \) by a communication step in a NSP (we write \( C' \models C \)) if for all \( x \in X_G \)

\[
C'(x) = (C(x) - \phi^b(x)(C(x), PO_x, FO_x)) \cup \\
\bigcup_{\{x,y\} \in E_G} (\phi^b(x)(C(y), PO_y, FO_y) \cap \phi^b(x)(C(y), PI_x, FI_y))
\]

holds.

We now describe how a communication step works in a NSPFC. For every pair of connected nodes \( x, y \), all the strings that satisfy the filter condition associated with the edge \( \{x, y\} \) are moved from one node to the other. Formally, \( C \vdash C' \), if

\[
C'(x) = (C(x) \setminus \bigcup_{\{x,y\} \in E_G} \phi^b(\{x,y\})(C(x), N(\{x,y\}))) \\
\cup \bigcup_{\{x,y\} \in E_G} \phi^b(\{x,y\})(C(y), N(\{x,y\}))
\]

for all \( x \in X_G \).

The computation of a NSP \( \Gamma \) on the input string \( w \in V^* \) is defined as a sequence of alternating steps (splicing, communication) that produces the configurations \( C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots, \) such that \( C_0^{(w)} \Rightarrow C_1^{(w)} \) and \( C_i^{(w)} \models C_{i+1}^{(w)} \), for all \( i \geq 0 \). The computation of NUSP and NSPFC are defined in the same way.

A computation as above halts, if there exists \( k \geq 1 \) such that the set of strings existing in the output node \( C(Halt) \) is non-empty. Then, we say that the string \( w \) is admitted by the splicing network in an accepting computation. The language defined/accepted by \( \Gamma \) is the set of all strings \( w \) over \( V \) such that the computation of \( \Gamma \) on \( w \) is an accepting one.

The time complexity of the halting computation \( C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots, C_m^{(x)} \) of \( \Gamma \) on \( x \in V^* \) is denoted by \( \text{Time}_{\Gamma}(x) \) and equals \( m \). The time complexity of \( \Gamma \) is the partial function from \( N \) to \( N \) formalized as follows:

\[
\text{Time}_{\Gamma}(n) = \max \{ \text{Time}_{\Gamma}(x) \mid x \in L(\Gamma), |x| = n \}.
\]

Furthermore, for a function \( f : N \to N \) and we write

\[
\text{Time}_{\text{NSP}}(f(n)) = \{ L \mid L \text{ is accepted by a NSP } \Gamma \text{ such that } \text{Time}_{\Gamma}(n) \in O(f(n)) \}.
\]

Analogously, one defines \( \text{Time}_{\text{NUSP}}(f(n)) \) as well as \( \text{Time}_{\text{NSPFC}}(f(n)) \).

3. Direct Simulations between NSPs and NUSPs

Obviously, since each NUSP can be immediately transformed into a NSP, we have:

**Proposition 1.** \( \text{Time}_{\text{NUSP}}(f(n)) \subseteq \text{Time}_{\text{NSP}}(f(n)) \) for any function \( f : N \to N \).

The converse is also true, namely:

**Proposition 2.** \( \text{Time}_{\text{NSP}}(f(n)) \subseteq \text{Time}_{\text{NUSP}}(f(n)) \) for any function \( f : N \to N \).

**Proof.** Let \( \Gamma = (V, U, \prec, >, G, \mathcal{N}, \beta, x_1, x_n) \) be a NSP with the underlying graph \( G = (X_G, E_G) \) and \( X_G = \{ x_1, x_2, \ldots, x_n \} \) for some \( n \geq 1 \); \( x_1 \equiv \text{Init} \) and \( x_n \equiv \text{Halt} \). Let further
\begin{equation}
\text{dom}(S_v) = \{a, b, u, v \in U^* \mid [(a, b); (u, v)] \in S_v\}. \end{equation}
We construct the NUSP \( \Gamma' = (V', \mathcal{U}', < \mathcal{U}', \mathcal{V}', \mathcal{X}', \mathcal{X}_v, \phi') \); \( \mathcal{X}_v \equiv \text{In} \) and \( \mathcal{X}_v \equiv \text{Hal} \), where

\begin{align}
\mathcal{U}' &= U \cup \mathcal{U}_v \cup T, \quad \mathcal{U}_v = \{a_v \mid a \in U\}, \\
T &= \{\# \psi \cup \{s_i \theta_i, \psi_i \mid 1 \leq i \leq n\}, \quad \mathcal{V}' = V, \\
u_v &= a_1 a_2 \ldots a_n \mid n = a_1 a_2 \ldots a_n \in U^*, \quad \psi_1 u_1 v_1 \varphi_2 \mid z = \beta_1 u v . \beta_2 \in U^*.
\end{align}

We now define the nodes of \( G' \) by Tables 1–3, while the edges of \( G' \) are both listed as well as graphically represented.

and \( \{x_i^0, x_i^1\}, \{x_i^2, x_i^3\} \in E_{G'} \).

Let \( x_i \), \( 1 \leq i \leq n - 1 \) be a splicing node. If \( \beta(x_i) = (w) \), then the nodes defined in Table 2 belong to \( X_{G'} \).

All the edges

\begin{itemize}
  \item \( \{x_i^0, x_i^1\} \), \( \{x_i^2, x_i^3\} \) for \( 1 \leq i \leq n - 1 \),
  \item \( \{x_i^1, x_i^2\}, \{x_i^1, x_i^2(\text{dom}(S_v))\} \) for \( 1 \leq i \leq n - 1 \),
  \item \( \{x_i^1, x_i^2\}, \{x_i^1, x_i^2\} \) for \( 1 \leq i \leq n - 1 \),
  \item \( \{x_i^2, x_i^3\} \) for \( 1 \leq i \leq n - 1 \),
  \item \( \{x_i^2(\text{dom}(S_v)), x_i^2(\text{dom}(S_v)))\} \) for all \( \{x_i, x_j\} \in E_G \), \( 1 \leq i \neq j \leq n - 1 \),
\end{itemize}

belong to \( E_{G'} \). As we have mentioned above, we present also a graphical representation in Figure 1.

If \( \beta(x_i) = (s) \), then \( x_i^2 \) is replaced by \( p \geq 1 \) nodes of the form \( x_i^2 \), \( 1 \leq k \leq p \), where \( \text{PO}_v = \{Z_1, Z_2, \ldots, Z_p\} \), \( p \geq 1 \). They are presented in Table 3. Furthermore, if \( \text{PO}_v = \emptyset \), then \( x_i^2 \) is removed. Analogously, if \( \text{FO}_v = \emptyset \), then \( x_i^2 \) is removed. Now, an edge between \( x_i^2 \) and \( x_i^2(\text{dom}(S_v)) \), \( [(a, b); (u, v)] \in \text{dom}(S_v) \), on the one hand, and each node \( x_i^2 \), on the other hand, is added to \( E_{G'} \).

The output node \( x_i^0 \) is defined as follows: \( S'(x_i^0) = S(x_i) \), \( P'(x_i^0) = PI(x_i) \) and \( P'(x_i^0) = PI(x_i) \), with \( A'(x_i^0) = A(x_i) \), \( \beta'(x_i^0) = \beta(x_i) \). Finally, we add all the edges \( \{x_i^0, x_i^1\} \) to \( E_{G'} \).

We now analyze a computation of \( \Gamma' \) on an input string \( \langle w \rangle \). In the input node \( x_i^0 \), \( \langle w \rangle \) is replaced with the sequence \( \psi \rangle \). Then, \( \langle w \psi \rangle \) is replaced with the symbols \( \beta_1 \psi \), and the resulting string is sent to the node \( x_i^0 \). Thus, the node \( x_i^1 \) contains the string \( z = \langle w \rangle \beta_1 \psi \), on the one hand, and each node \( x_i^2 \), on the other hand, is added to \( E_{G'} \).

We now analyze a computation of the first splicing step executed by \( \Gamma \).

Firstly, we analyze the procedure for the case of a splicing node \( x_i \in \Gamma \). In \( x_i^1 \), a rule \( [(a, b); (u, v)] \) is applied to \( z_1 z_2 \beta_1 \), producing \( z_1 u v \beta_2 \) and \( \beta_1 u v \beta_2 \), if a rule \( [(a, b); (u, v)] \) can be applied to \( z_1 z_2 \) in the node \( x_i \). Let \( y_1 z_2 \) be one string obtained after a splicing step from \( z_1 z_2 \beta_1 \) in \( x_i^1 \). Note that if the splicing rule cannot be applied, then \( z_1 z_2 \beta_1 \) may go out from \( x_i^1 \) and enter the following nodes:

- \( x_i^2 \), provided that \( z_1 z_2 \beta_1 \) satisfies the condition of the input filter of \( x_i \).
• $x_i^{\text{return}_1}$ and $x_i^{\text{return}_2}$ if $\beta(x_i) = (w)$ or each of the nodes $x_i^{\text{return}_1}$ if $\beta(x_i) = (s)$, provided that $z_1z_2$ does not satisfy the condition of the output filter of $x_i \in \mathcal{X}_G$.
• $x_i^\text{δ}(\text{dom}(S_{x_i}))$.

All cases are to be analyzed. If $z_1z_2\#_i$ leaves $x_i^1$ and enters $x_i^{\text{check-in}}$, the symbol $\#_i$ is replaced with $\theta_i$, which locks the string in that node. If $z_1z_2\#_i$ leaves $x_i^1$ and enters $x_i^{\text{return}_1}$, then ping-pong processes between these two nodes as well as between $x_i^1$ and $x_i^2(\text{dom}(S_{x_i}))$ start. We distinguish here the cases of weak and strong filters. If $\beta(x_i) = (w)$, the string $z_1z_2\#_i$ can also enter $x_i^{\text{return}_2}$, starting an identical relationship to the one between $x_i^{\text{return}_1}$ and the nodes $x_i^1, x_i^2(\text{dom}(S_{x_i}))$. If $\beta(x_i) = (s)$, the string enters the nodes $x_i^{\text{return}_2}$, provided that $z_1z_2\#_i$ does not contain the character $Z_{x_i}$, followed by the same ping-pong process between these nodes and $x_i^1, x_i^2(\text{dom}(S_{x_i}))$. In this last case, the structure simulates the situation where a string remains in the node $x_i$ because it only contains a proper subset of the characters in $\text{PO}(x_i)$.

If $z_1z_2\#_i$ leaves $x_i^1$ and enters $x_i^2(\text{dom}(S_{x_i}))$, then $\#_i$ is replaced by $\theta_i$; the new string $z_1z_2\theta_i$ is simultaneously sent to all nodes $x_i^{\text{check-out}}, x_i^{\text{return}_1}$ and $x_i^{\text{return}_2}$. If it enters $x_i^{\text{check-out}}$, then $\theta_i$ is replaced firstly by $\gamma_i$ (in $x_i^{\text{check-out}}$) and secondly by some $\gamma_j$ (in $x_i^{\text{continue}}$) provided that \{ $x_i, x_j$ \} $\in \mathcal{E}_G$. The newly obtained string $z_1z_2\gamma_j$ is sent to $x_i^{\text{check-in}}$. Note that this situation simulates exactly the situation when $z_1z_2$ is sent to $x_i$ after staying unchanged for one splicing step in $x_i$. The case when $z_1z_2\theta_i$ enters any of the nodes $x_i^{\text{return}_1}$ and $x_i^{\text{return}_2}$ is considered above.

The only case remaining to be analyzed is when $z_1z_2\#_i$ is transformed into $y_1y_2\#_i$ (either $y_1 = z_1$ or $y_2 = z_2$) after applying a splicing rule in $x_i^1$. Then, $y_1y_2\#_i$ leaves $x_i^1$. We follow the route of this string through the network: $x_i^1$, where $\#_i$ is replaced by $\theta_i$, then $x_i^j$, where $u_\uparrow$ and $v_\uparrow$ are replaced by $u$ and $v$, respectively. We analyze in detail the case for a string $y_1y_2\theta_i$ with $y_1 \neq z_1$. The application of the first splicing rule on a string $y = \beta_1u_\uparrow y_2\theta_i$ yields two strings, namely $y' = \beta_1u_\uparrow\#_i$ and $y'' = \pi uy_2\theta_i$. These two strings cannot exit the node, as they contain $\#_i$ and $\pi$, respectively. In the next splicing step, these two strings can only combine between themselves through the second splicing rule $\{(\pi, ub); (u, v_\uparrow\#_i)\}$, yielding two new strings: $y'^2 = \pi uy_\uparrow\#_i$ and $y''^2 = \beta_1uy_2\theta_i$. The first one cannot be used in the computation anymore, while the latter is the original string $y_1y_2\theta_i$ with $u_\uparrow$ replaced by $u$. The procedure for a string $y_1y_2\theta_i$ with $y_2 \neq z_2$ is analogous through the application of the other two splicing rules.

After leaving $x_i^1$, the new string, say $z'$, can enter either $x_i^{\text{check-out}}$ or at least one of $x_i^{\text{return}_1}$ and $x_i^{\text{return}_2}$. If it enters $x_i^{\text{check-out}}$ and consequently $x_i^{\text{continue}}$, then the following computational step in $\Gamma$ was simulated in $\Gamma'$ as follows: $z'$ was obtained from $z_1z_2$ by means of a splicing rule $\{(a, b); (u, v)\}$ in $x_i \in \mathcal{X}_G$, then $z'$ was sent to all neighbors of $x_i$. The situation when $z'$ enters one of the nodes $x_i^{\text{return}_1}$ and $x_i^{\text{return}_2}$ corresponds to the situation when $z'$ remains in $x_i$ for a new splicing step, as it cannot pass the output filter of $x_i$.

We now analyze the computational steps required for simulating a computation in $\Gamma$. The input node $x_i^1$ and $x_i^0$ require 1 splicing step (or 2 computational steps) each. In the worst case, a splicing step in one of the nodes \{ $x_i$ | $1 \leq i \leq n - 1$ \} can be simulated in $\Gamma'$ in 7 splicing steps (or 14 computational steps) distributed in the following way:

• 2 steps in $x_i^1$.
• 2 steps in $x_i^0$.
• 4 steps in $x_i^0$.
• 2 steps in $x_i^{\text{check-out}}$.
• 2 steps in $x_i^{\text{continue}}$.
• 2 steps in $x_i^{\text{check-in}}$ | $1 \leq j \leq n - 1$.

Note that the simulation only requires 12 computational steps if $x_i \equiv x_i^0$ since $x_i^0$ is not simulated by a subnetwork and the computation halts once a string enters that
node. By all the above considerations, we conclude that \( L(\Gamma) = L(\Gamma') \) and \( \text{Timer}(n) \in O(\text{Timer}(n)) \). □

Table 1. Description of the initial nodes.

| Node         | \( S \) | \( P \) | \( F \) | \( A \) | \( \beta' \) |
|--------------|---------|---------|---------|-------|------------|
| \( x_1^0 \)  | \{[(\#, \#), (#, \#)]\} | \{\#\} | \{\#, \#\} | \# \} | \#(\#) |
| \( x_1^0 \)  | \{[(\#, \#), (#, \#)]\} | \{\#\} | \{\#, \#\} | \# \} | \#(\#) |
| \( x_1^0 \)  | \{[(\#, \#), (#, \#)]\} | \{\#\} | \{\#, \#\} | \# \} | \#(\#) |

Table 2. Description of the intermediate nodes.

| Node         | \( S \) | \( P \) | \( F \) | \( A \) | \( \beta' \) |
|--------------|---------|---------|---------|-------|------------|
| \( x_1^{\text{check-in}} \) | \{[(\#, \#), (#, \#)]\} | \{\#\} | \{\#, \#\} | \# \} | \#(\#) |
| \( x_1^{\text{check-out}} \) | \{[(\#, \#), (#, \#)]\} | \{\#\} | \{\#, \#\} | \# \} | \#(\#) |

Figure 1. A graphical representation of the network.
Table 3. Description of the returning nodes.

| Node          | S             | P             | F              | A             | β' |
|---------------|---------------|---------------|----------------|---------------|----|
| \( x_{1,1}^{\text{return}} \) | \{[(\varepsilon, \theta_1); (\#, \#)]\} | \( \emptyset \) | \{Z_k\} \cup \{U_p \cup \{\#\} \cup \{\#_i\} \} \cup \{\#_j; \forall_j; 1 \leq j \leq n\} | (w) |    |

4. Direct Simulations between NSPs and NSPFCs

Proposition 3. \( \text{Time}_{\text{NSP}}(f(n)) \subseteq \text{Time}_{\text{NSPFC}}(f(n)) \) for any function \( f : N \rightarrow N \).

Proof. We take a NSP \( \Gamma = (V, U, <, >, G, N, \beta, x_1, x_n) \) with the underlying graph \( G = (X_G, E_G) \) with the set of nodes \( X_G = \{x_1, x_2, \ldots, x_n\} \) for some \( n \geq 1; x_1 \equiv I_n \) and \( x_n \equiv \text{Hall} \). We construct the NSPFC \( \Gamma' = (V', U', <, >, G', N', \beta', x_1, x_n'); x_1 \equiv I_n \) and \( x_n' \equiv \text{Hall} \), where

\[
V' = V, \quad U' = U \cup U_p \cup T, \\
U_p = \{a \mid a \in U\}, \quad u_v = a_1a_2 \cdots a_m | u \in U^v, \\
z_v(u, v) = \beta_1u \psi_2 | z = \beta_1u \psi_2 \in U', \\
T = \{S_i \mid i \in \{1, \ldots, n\} \} \cup \{\#, \pi, \theta, \sigma\}
\]

We now define the parameters of \( \Gamma' \). First, for each pair of nodes \( x_i, x_j \) where \( 1 \leq i \neq j \leq n, i \neq n \) of \( X_G \) with \( \{x_i, x_j\} \in E_G \), we have the following nodes in \( \Gamma' \):

- \( x_{1,i}^{U_p} \) \( x_{1,i}^{U_p} \)
- \( x_{2,i}^{U_p} \) \( x_{2,i}^{U_p} \)
- \( x_{3,i}^{U_p} \) \( x_{3,i}^{U_p} \)

and the following edges (the nodes \( x_i^\prime \) and \( x_i^\prime \) are to be defined below):

- \( \{x_i^1, x_i^1\} : P = PO(x_i), \quad F = FO(x_i) \cup \{\}$, \quad \beta' = \beta(x_i) \)
- \( \{x_i^2, x_i^2\} : P = \{\}$, \quad F = \{\psi\}, \quad \beta' = (w), \)
- \( \{x_i^3, x_i^3\} : P = PI(x_i), \quad F = FI(x_i) \cup \{\#, \pi, \theta\}, \quad \beta' = \beta(x_i) \)

The specifications for the input node \( x_i \) and the output node \( x_n' \) are as follows:

- \( x_i : S(x_i) = \{[(\varepsilon, >); (\#, > \psi)]\}, \quad A(x_i) = \{\# > \psi\}, \)
- \( x_n' : S(x_n') = S(x_n \in \Gamma), \quad A(x_n') = A(x_n \in \Gamma) \)

and the edge:

- \( \{x_i, x_n'\} : P = \{\psi\}, \quad F = \{\}$, \quad \beta' = (s) \)

For each node \( x_i \) where \( 1 \leq i \leq n - 1 \) in \( \Gamma \), we add a subnetwork (Figure 2) to \( \Gamma' \) according to the cases considered in the sequel.

Case 1. If \( x_i \) is a splicing node with \( \beta(x_i) = (w) \), the subnetwork is defined as follows:

- \( x_i^1 \)
  \( S = \{[(\varepsilon, \psi); (\#, S_i)]\} \cup \{[(\varepsilon, S_i); (\#, \theta)]\} \)
  \( A = \{\#_{S_i}, \#_{\theta}\} \)
- \( x_i^2 \)
  \( S = \{[(a, b); (u_v, v)]\} \cup \{[(a, b); (u, v)] \in S_{x_i}, \{a, b, u, v\} \in U^v\} \)
  \( A = \{\#_{u_v, v}^k \mid 1 \leq k \leq \text{card}(A_{x_i}), \#_{u_v}^k \in A_{x_i}\} \)
- \( x_i^3 \)
  \( S = \{[(\varepsilon, S_i); (\#, \theta)]\} \cup \{[(\varepsilon, \varepsilon); (\#, \#)]\} \cup \{[(\varepsilon, \theta); (\#, \#)]\} \)
  \( A = \{\#_{\theta}, \#_{\#}\} \)
• \( x^f_i \)

\[ S = \{ [(a, \nu \psi); (#, \nu \pi)] \mid a \in U^* \} \cup \{ [(avu, \pi); (#uv, \pi \tau)] \mid \{ a, v \} \in U^*, t \in \{ U^* \cup \{ \theta \} \} \} \cup \{ [(\nu \psi, \pi); (#uv, \pi \tau)] \mid \{ a, v \} \in U^*, \theta \in \{ U^* \cup \{ \theta \} \} \}

\[ A = \{ \pi u\#; \nu \pi \nu \pi \theta \}

\[ x^r_{ri} \]

\[ S = \{ [(\epsilon, \theta); (#, \sigma)] \}

\[ A = \{ \# \sigma \}

The edges between them are:

\[ \begin{align*}
\{ x^f_i, x^1_i \} : & \quad P = \{ S_i \}, \quad F = \{ \# \}, \quad \beta' = (w), \\
\{ x^r_i, x^1_i \} : & \quad P = \{ S_i \} \cup U^\psi, \quad F = \{ \theta, \# \}, \quad \beta' = (w), \\
\{ x^f_i, x^{rir}_i \} : & \quad P = \{ \theta \}, \quad F = \{ \# S_i, \pi \}, \quad \beta' = (w), \\
\{ x^f_i, x^{rir}_i \} : & \quad P = FO(x_i), \quad F = \{ \# S_i, \pi \}, \quad \beta' = (w), \\
\{ x^r_{ri}, x^1_i \} : & \quad P = \{ \sigma \}, \quad F = \{ \# S_i, \theta \} \cup U^\psi, \quad \beta' = (w), \\
\{ x^{rit}_i, x^1_i \} : & \quad P = \{ \sigma \}, \quad F = \{ \# S_i, \theta \} \cup U^\psi, \quad \beta' = (w).
\end{align*} \]

Figure 2. Subnetwork for splicing step.

Case 2. If \( x_i \) is a splicing node with strong filters, we replace the node \( x^{rit}_i \) with \( p \geq 1 \) nodes of the form \( x^{rit}_{1k} , 1 \leq k \leq p \), where \( PO_{x_i} = \{ Z_1, Z_2, \ldots, Z_p \} \), \( p \geq 1 \). These new nodes are defined as follows:

\[ x^{rit}_{1k} : S = \{ [(\epsilon, \theta); (#, \sigma)] \}, A = \{ \# \sigma \}, \]

and the edges between \( x^{rit}_i \) and the nodes \( x^f_i \) and \( x^r_i \) are replaced with \( 2p \) edges of the form

\[ \begin{align*}
\{ x^f_i, x^{rit}_{1k} \} : & \quad P = \emptyset, F = \{ Z_k, \#, \sigma, \pi \} \cup U^\psi, \beta' = (w), \\
\{ x^{rit}_{1k}, x^1_i \} : & \quad P = \{ \sigma \}, F = \{ \# S_i, \theta \} \cup U^\psi, \beta' = (w).
\end{align*} \]

In both cases, if \( PO_{x_i} = \emptyset \), then the nodes \( x^{rit}_i \) and \( x^{rit}_{1k} \) are removed. Analogously, if \( FO_{x_i} = \emptyset \), then \( x^{rit}_{ri} \) is removed.
We now analyze a computation of $\Gamma'$ on the input string $w = \langle z \rangle$. In the input node $x_1$, the symbol $\psi$ is attached to the end of the string. Next, it enters $x_1'$ and the simulation of a computation in $\Gamma$ starts. We assume that $w\psi$ lies in $x_1'$ while the string $w$ is found in $x_1$, the input node of $\Gamma$. Inductively, we may assume that a string $w$ is found in some $x_i$, a node of $\Gamma$, as well as $w\psi$ in $x_i'$ from $\Gamma'$.

Let $x_i$ be a splicing node, where a rule $[(a, b)]; (u, v)]$ is applied to $w$ producing $a_1av\beta_2$ and $b_1ub\beta_2$, if $w = a_1ab\beta_2$ or $w$ otherwise. In $\Gamma'$, the string $w\psi$ is processed as follows. First $w\psi$ becomes $w\psi_i$ in $x_i'$, then it can enter $x_i'$ only. Here, it may become $w' = a_1av\psi\beta_2\psi_i$ or $w' = b_1ub\psi\beta_2\psi_i$, if $w = a_1ab\beta_2$, or it is left unchanged.

Further, a string of the form $w'\psi_i$ produced in this node can go back to $x_i'$, where $\psi_i$ is changed to $\theta$, sealing that string in the node. In the other hand, the strings can also enter $x_i'^2$. In $x_i'^2$, the symbols $\psi_i$ and $\sigma$ are replaced by $\theta$, closing the route back to $x_i'^1$. Note that the other produced strings $\#\psi_i$, $\#\sigma$ and $\#\theta$ remain locked in the node.

Finally, the strings enter $x_i'^1$ where each symbol $c \in u_\psi$ or $c \in v_\psi$, if present, is rewritten into the associated symbol $c$, replacing $u_\psi$ with $u$ and $v_\psi$ with $v$. Thoroughly, the string $w = a_1av\psi\beta_2\theta$ is split into the strings $w_1 = a_1av\pi\theta$ and $w_2 = \#v\psi\beta_2\theta$. Both strings cannot leave the node because of the characters $\pi$ and $\#$. In the next splicing step, the only rule that can be applied to these strings is $[[av, \pi]; (#v_\psi, t)]$. At the same time, this rule can only be applied to these two new strings. The rule yields $w'_1 = \alpha_1av\pi\beta_2\theta$ and $w'_2 = \#v\psi\pi\theta$. The former contains the string generated by the original node $x_i$, $w_1 = a_1av\beta_2$, completing the simulation of the splicing step. On the other hand, the latter cannot exit the node or be modified by any rule, so it remains locked for the rest of the computation. The logic is analogous for the other string generated by the splicing rule. Thus, in node $x_i'^1$, we have obtained the strings $a_1av\psi\beta_2\theta$ and $b_1ub\psi\beta_2\theta$, if $w = a_1ab\beta_2$.

In conclusion, if $w \in U^*$ is a string in the nodes $x_i$ of $\Gamma$ and $w\psi$ in $x_i'$ of $\Gamma'$, then we can obtain $w' \in U^*$ in one splicing step of $\Gamma$ if and only if we can obtain the associated string $w'\theta$ in the node $x_i'^1$ of $\Gamma'$ in 4 splicing steps (or 8 computational steps). At this point, we note that $w'$ can leave $x_i$ and enter $x_j$ in $\Gamma$ if and only if the string can leave $x_i'$ and enter $x_j$ via the nodes $x_{ij}$ and $x_{ij}'$. If $w'$ can leave $x_i$, but cannot enter $x_j$ in $\Gamma$, then it is trapped in $x_{ij}'$ in $\Gamma'$. Furthermore, a string entering $x_j$ has the character $\psi$ replaced with $\psi$, and consequently, it cannot return back to $x_{ij}'$. Finally, if the string cannot leave the node $x_i$, then it is sent by $x_i'$ to $x_i'^1$ and $x_i'^2$ (in the case of weak filters) or to the nodes $x_i'^{t1}$ and $x_i'^{t2}$, for all $Z_k \in PO(x_i)$ (in the case of strong filters). In these nodes, the character $\theta$ is replaced with $\sigma$, and the resulting string is sent to $x_j'$. If the string is not split in the following splicing step, it returns to the nodes $x_i'^{t1}$ and $x_i'^{t2}$ in the case of weak filters or to the nodes $x_i'^{s1}$ and $x_i'^{s2}$ in the case of strong filters, starting a ping-pong process between them, which continues until the string yields new ones by means of another splicing step in $x_i'$. In this last case, the new strings can only enter $x_i'^2$ because it contains characters $a_\psi \in U_\psi$ and the process described above is repeated.

We now analyze the computational steps required for simulating a computation in $\Gamma$. The input node $x_1$ and the node $x_1'$ require 1 splicing step (or 2 computational steps) each. In the worst case, a splicing step in one of the nodes $\{x_i | 1 \leq i \leq n - 1\}$ can be simulated in $\Gamma'$ in 7 splicing steps (or 14 computational steps) distributed in the following way:

- 2 steps in $x_i'$.
- 2 steps in $x_i'^2$.
- 4 steps in $x_i'^1$.
- 2 steps in $x_{ij}'$.
- 2 steps in $x_{ij}'^2$.  
- 2 steps in $x_{ij}'^2$.  
- 2 steps in $x_i'$.  
- 2 steps in $x_i'^2$.  
- 4 steps in $x_i'^1$.
Note that the simulation only requires 12 computational steps if \( x^j_i \equiv x^k_n \) since the computation halts when a string enters \( x^j_i \). We conclude that \( L(\Gamma) = L(\Gamma') \) and \( Time_\Gamma(n) \in O(Time_{\Gamma'}(n)) \). □

The converse of the previous proposition holds.

**Proposition 4.** \( \text{Time}_{\text{NSPF}}(f(n)) \subseteq \text{Time}_{\text{NSPF}}(f(n)) \) for any function \( f : N \rightarrow N \).

**Proof.** Let \( \Gamma = (V, U, \langle, >, G, N, \beta, x_1, x_n) \) be a NSPFC with \( G = (X_G, E_G) \), \( X_G \) having \( n \) nodes \( x_1, x_2, \ldots, x_n \); \( x_1 \equiv \text{In} \) and \( x_n \equiv \text{Hal} \). We construct the NSP \( \Gamma' = (V', U', \langle, >, G', N', \beta', x_1, x_n) \); \( x_1 \equiv \text{In} \) and \( x_n \equiv \text{Hal} \), where

\[
V' = \{\phi, \psi, \Gamma\}, \quad U' = U \cup U_\phi \cup \{\phi, \psi\}, \quad Y = \{Y_i \mid 1 \leq i \leq n - 1\}, \quad u_\phi = a_1 a_2 \ldots a_i [u], \quad |u| = a_1 a_2 \ldots a_i \in U^*.
\]

We now define the parameters of \( \Gamma' \). First, we add a main subnetwork composed by the input node \( x_1 \), the nodes \( \{x_i^j \mid 1 \leq i \leq n\} \), the nodes \( \{x_i^j \mid 1 \leq i \leq n - 1\} \) and the nodes \( \{x_i, x_j \} \in E_G \).

- **node \( x_1 \):**
  \[
  S = \{((\phi, \psi), \#), (\#), X_1]\}, \quad A = \{\# > X_1\}, \quad -PI = V, \quad PO = X, \quad FO = \emptyset, \quad -\beta' = (\emptyset).
  \]

- **nodes \( x_i^j \), \( 1 \leq i \leq n - 1 \):**
  \[
  S = \{((a, b); (u, v, \psi)) \mid \{a, b\}; (u, v) \in S, \{a, b, u, v\} \in U^*\}, \quad A = \{a \in U\}, \quad -PI = \{X_i, \psi\}, \quad PI = X \cap \{X_i\} \cap \{\#\}, \quad PO = \{X_i, U_\psi\}, \quad FO = \emptyset, \quad -\beta' = (\emptyset).
  \]

- **node \( x_n^i \):**
  \[
  S = S(x_n), \quad A = A(x_n), \quad -PI = \{X_n, \psi\}, \quad PI = X \cap \{X_n\} \cap \{\#\}, \quad PO = \{X_n, U_\psi\}, \quad FO = \emptyset, \quad -\beta' = (\emptyset).
  \]

- **nodes \( x_i^j \), \( 1 \leq i \leq n - 1 \):**
  \[
  S = \{((\phi, X_i); (\#), X'_i)) \} \cup \{((\phi, \psi); (\#), X'_i))\}, \quad A = \{\#X'_i\}, \quad -PI = \{X_i, U_\psi\}, \quad PI = \{\#\}, \quad PO = \{X'_i\}, \quad FO = \emptyset, \quad -\beta' = (\emptyset).
  \]

- **nodes \( x_j^i \), \( 1 \leq i \leq n - 1 \):**
  \[
  S = \{((a, v, \psi) \# (\#), X'_i)) \} \cup ((a, v, \psi) \# (\#), X'_i))\}, \quad A = \{\#X'_i\}, \quad -PI = \{X_i, U_\psi\}, \quad PI = \{\#\}, \quad PO = \{X'_i\}, \quad FO = \{\#\}, \quad -\beta' = (\emptyset).
  \]

- **nodes \( x_i, x_j \), \( 1 \leq i \neq j \leq n, i \neq n, \{x_i, x_j\} \in E_G \):**
  \[
  S = \{((\phi, X'_i); (\#), X'_j))\}, \quad A = \{\#X'_i\}, \quad -PI = P_{\{x_i, x_j\}}, \quad PI = F_{\{x_i, x_j\}} \cup \{X_j\}, \quad PO = \{X_j\}, \quad FO = \{\#\}, \quad -\beta' = \beta((x_i, x_j))
  \]
The edges between them are:
\[
\begin{align*}
S &= \\
A &= 
\end{align*}
\]

Next, for each node \( x_i \mid 1 \leq i \leq n - 1 \) in \( \Gamma \), we add a subnetwork to \( \Gamma' \) according to the cases considered in the sequel.

Case 1. Let \( E_i \) be the number of nodes connected to the node \( x_i^2 \) of the form \( x_{ij} \), we add the corresponding nodes \( \{x^{r1}_{ij}(k), x^{r2}_{ij}(k) \mid 1 \leq k \leq E_i \} \) to the subnetwork. Note that \( E_i \) is also equal to the number of edges between the node \( x_i \) and the nodes \( x_j \mid 1 \leq i \neq j \leq n, i \neq n \) in the network \( \Gamma \). If the edges \( \{x_i, x_j \} \in \Gamma \) have \( \beta = (w) \), the subnetwork is defined as follows:

- nodes \( x^{r1}_{ij}(k), 1 \leq k \leq E_i, 1 \leq i \neq j \leq n, i \neq n, \{x_i, x_j \} \in E_G \):

- \( S = \)

- \( A = \)

- \( -PI = F_{(x_i, x_j)}, FI = \{#, \psi\} \cup \{Y \setminus Y_k\}, PO = \{\psi, Y_{k+1}\}, FO = \{#\}, \)

- \( -\beta' = (w). \)

The edges between them are:
\[
\begin{align*}
\{\{x^2_i, x^{r1}_{ij}(1)\}, \{x^2_i, x^{r2}_{ij}(1)\}\} \cup \\
\{\{x^{r1}_{ij}(k), x^{r1}_{ij}(k + 1)\}, \{x^{r1}_{ij}(k), x^{r1}_{ij}(k + 1)\} \mid 1 \leq k \leq E_i - 1\} \cup \\
\{\{x^{r1}_{ij}(k), x^{r1}_{ij}(k + 1)\}, \{x^{r1}_{ij}(k), x^{r1}_{ij}(k + 1)\} \mid 1 \leq k \leq E_i - 1\} \cup \\
\{\{x^{r2}_{ij}(E_i), x^r_i\}, \{x^{r2}_{ij}(E_i), x^r_i\}\}, 
\end{align*}
\]

Case 2. If the \( k \) edge \( \{x_i, x_j\} \) has \( \beta = (s) \), we replace each of the nodes \( x^{r2}_{ij}(k) \) for \( p \geq 1 \) nodes of the form \( x^{r2}_{ij}(k), 1 \leq t \leq p \), where \( p(x_i, x_j) = \{Z_1, Z_2, \ldots, Z_p\}, p \geq 1 \). The parameters \( S, A \) and \( \beta \) of these new nodes remain the same, while the input and output filters are defined as follows:

\[
x^{r2}_{ij}(k) : PI = \emptyset, FI = \{Z_t, #, \psi\} \cup \{Y \setminus Y_k\}, PO = \{\psi, Y_{k+1}\}, FO = \{#\}
\]
and the edges involving $x_{i,j}^{rt}(k)$ are replaced with $p$ edges involving each of the nodes $x_{i,j}^{rt}(k)$.

In both cases, if the $k$ edge has $P_{\{x_i,x_j\}} = \emptyset$, then the nodes $x_{i,j}^{rt}(k)$ and $x_{i,j}^{rt}(k)$ are removed. Analogously, if $F_{\{x_i,x_j\}} = \emptyset$, then $x_{i,j}^{rt}(k)$ is removed. Furthermore, if $P_{\{x_i,x_j\}} = \emptyset$ and $F_{\{x_i,x_j\}} = \emptyset$, the whole subnetwork given in Figure 3 is removed because a string is always able to leave the node $x_i \in \Gamma$ through the edge $k$.

![Figure 3](image)

Figure 3. Subnetwork to handle strings that cannot exit the node $x_i$.

Let us assume the string $w = < z >$ to be the input string in $\Gamma$. In the input node $x_i$ of $\Gamma'$, the symbol $X_i$ is attached to the string and then sent to the node $x_i^\prime$. Thus, the node $x_i \in \Gamma$ contains the string $w = < z >$ while $w = < z > X_i$ lies in the analogous node $x_i^\prime \in \Gamma'$. In a more general setting, we assume that a string $y' = yX_i, y \in U^*$, enters $x_i^\prime \in \Gamma'$ at a given step of the computation of $\Gamma'$ on $w$ if and only if the string $y$ enters $x_i \in \Gamma$ at a given step of the computation of $\Gamma$ on $w$.

Let $y$ be transformed into $z_1 = \alpha_1av\beta_2$ and $z_2 = \beta_1ub\alpha_2$ in node $x_i$ and $z_1$ and/or $z_2$ can pass the filter on the edge between $x_i$ and $x_j$. In $\Gamma'$, the string $y' = yX_i$ is transformed into $z_1' = \alpha_1av\beta_2X_i$ and $z_2' = \beta_1ub\alpha_2X_i$ in the node $x_i'$. If the string is of the form $y' = y\psi$, it is transformed into $z_1' = \alpha_1av\beta_2X_i$ and $z_2' = \beta_1ub\alpha_2\psi$. The strings enter $x_i'$ where the characters $X_i$ and $\psi$ are replaced with $X_i'$. Thus, we get the same strings with either $X_i$ or $\psi$. Then, it continues into $x_j'$, where the symbols in $u\psi$, if present, are replaced with the original counterpart $u$ and $v$. In detail, the string $z_i'$ is split into the strings $z_1(1) = \alpha_1av \pi X_i'$ and $z_1(2) = \#u\psi\beta_2X_i'$. Both strings cannot leave the node because of the characters $\pi$ and $\#$. In the next splicing step, the rule $\{(av, \pi); (\#v, \psi)\}$ is applied to the two strings, yielding $z_i(1)'' = \alpha_1av\psi\beta_2X_i'$ and $z_i(2)'' = \#uv\pi X_i'$. The former constitutes the string $z = z_iX_i'$ where $z_1$ is the string yielded by $x_i \in \Gamma$, completing the simulation of the application of a splicing rule of $\Gamma$. On the other hand, the latter cannot exit the node or be modified by any rule, so it remains locked for the rest of the computation. The logic is analogous for the other string generated by the splicing rule. The new string $z = z_iX_i'$ is sent to the nodes $x_{i,j}$ associated to the edges $\{x_i, x_j\}$ in the original network. Since the input filters of these nodes are identical to the edge filters in $\Gamma$, the string $z_iX_i'$ can enter a node $x_{i,j} \in \Gamma'$ if and only if the string $z_1$ can pass the filter between $x_i$ and $x_j$ in $\Gamma$. Note that the converse is also true. Subsequently, the symbol $X_i'$ is replaced with $X_i$ and the new string is sent to $x_j'$ in $\Gamma'$ where a similar procedure starts. Alternatively, if the string $y$ is not split in $x_j \in \Gamma$, the same event happens in $x_j' \in \Gamma'$. If the string is of the form $y' = y\psi$, it remains locked in that node until a splicing rule can be applied on it while a string of the form $y' = yX_i$ still enters the node $x_i'$ because of the symbol $X_i$ and the same computational process described above starts.

On the other hand, a copy of the string $z_iX_i'$ is also sent to the subnetwork illustrated in Figure 3, where the case of a string not passing the filters of any edge between the node $x_i$ and the connected nodes in the original network $\Gamma$ is handled. We distinguish here between the cases of strong and weak filters. For each edge $k$ with $\beta = (w), 1 \leq k \leq E_i$,
the node \( x_{ij}^s(k) \) checks if the string cannot enter the corresponding node \( x_{ij}(k) \) because of containing forbidden symbols, while the node \( x_{ij}^f(k) \) verifies if the string contains the characters required by the \( PI \) filter of the node \( x_{ij}(k) \). Thoroughly, the string exits \( x_i^s \) and enters the first pair of nodes \( x_{ij}^s(1) \) and \( x_{ij}^f(1) \) granted that the associated string \( z \) could not pass the filters of the first edge in the network \( \Gamma \). In that node, \( X'_j \) is replaced with \( Y_2 \) ensuring that the new string could only enter the next pair of nodes \( x_{ij}^s(2) \) and \( x_{ij}^f(2) \) associated to the second edge. Subsequently, \( Y_k \mid 2 \leq k \leq E_i - 1 \) is replaced by the next character \( Y_{k+1} \) in the remaining pair of nodes \( x_{ij}^s(k) \) and \( x_{ij}^f(k) \), forcing the string to continue through them in sequence. If the string can enter one of the connected nodes through an edge in \( \Gamma \), it will be lost at some point of this subnetwork as it will be refused by the input filters of the nodes corresponding to that edge. Otherwise, it will reach the last pair of nodes where the symbol \( Y_{E_i} \) will be changed to \( \psi \), yielding a string of the form \( z\psi \) which will be returned to \( x_i^s \). Because of the character \( \psi \), this string cannot leave the node \( x_i^s \) ensuring it remains there until it can be used in a new splicing step. Thus, the following computational step in \( \Gamma \) was simulated in \( \Gamma' \): \( z \) could not leave the node \( x_i \) and it remained in that node for new splicing steps. In the case of an edge \( k \) with strong filters, the computation follows the same procedure described above with the difference that the node \( x_{ij}^f(k) \) is replaced with \( p \geq 1 \) nodes of the form \( x_{ij}^s(k) \), \( 1 \leq t \leq p \), where \( P_{x_i,x_j} = \{Z_1, Z_2, \ldots, Z_p\}, p \geq 1 \). A node \( x_{ij}^f(k) \) refuses any string \( z \) containing the character \( Z_i \). Thus, the string can only continue through this network by means of a node \( x_{ij}^s(k) \) for some \( 1 \leq t \leq p \) if it only contains a proper subset of \( P_{x_i,x_j} \).

We analyze now the case where a string \( zX'_j \) enters \( x_{ij} \) from a node \( x_i^s \), for some \( 1 \leq i \leq n - 1 \). In that node, \( X'_j \) is replaced with \( X_j \) and the resulting string enters \( x_n^s \), ending the computation. Note that by the considerations above, a string enters \( x_{ij} \) if and only if a string from \( x_j \) was able to pass the filter on the edge between \( x_i \) and \( x_n \) in \( \Gamma \).

We now analyze the computational steps required for simulating a computation in \( \Gamma \). The input node \( x_I \) requires 1 splicing step (or 2 computational steps). In the worst case, a splicing step in one of the nodes \( \{x_i \mid 1 \leq i \leq n - 1\} \) can be simulated in the associated subnetwork in \( 4 + E_i \) splicing steps (or \( 8 + 2E_i \) computational steps) distributed in the following way:

- 2 steps in \( x_i^s \).
- 2 steps in \( x_i^f \).
- 4 steps in \( x_i^s \).
- 2 steps in \( x_{ij}^f \) if the string \( z \) yielded by the splicing step in the node \( x_i \) could pass any of the edge filters between \( x_i \) and \( x_j \) in \( \Gamma; 1 \leq i \neq j \leq n, i \neq n \).
- \( 2E_i \) steps if \( z \) could not pass any of the edge filters in \( \Gamma \).

Since \( E_i \) is bounded by \( n - 1 \), where \( n \) is the number of nodes in \( \Gamma \), a NSPFC can be simulated by NSP in \( \Theta(n) \) time. As a result of this analysis, we conclude that \( L(\Gamma) = L(\Gamma') \) and \( Time_{\Gamma}(n) \in \Theta(Time_{\Gamma'}(n)) \).

5. Direct Simulations between NUSPs and NSPFCs

Proposition 5. \( \text{Time}_{\text{NSUP}}(f(n)) \subseteq \text{Time}_{\text{NSPFC}}(f(n)) \) for any function \( f : \mathbb{N} \rightarrow \mathbb{N} \).

Proof. This proposition can be proved through a NSPFC simulation of NUSP identical to the proposed simulation for NSP in the previous section, with the only difference being the replacement of the filters \( PI, PO \) with \( P \) and \( FI, FO \) with \( F \) in the specifications.

Therefore, the input node \( x_I \) and the node \( x_I^s \) require 1 splicing step (or 2 computational steps) each. In the worst case, a splicing step in one of the nodes \( \{x_i \mid 1 \leq i \leq n - 1\} \) can be simulated in \( \Gamma' \) in 7 splicing steps (or 14 computational steps) distributed in the following way:
• 2 steps in $x^1_i$.
• 2 steps in $x^2_i$.
• 4 steps in $x_i^t$.
• 2 steps in $x^1_i$.
• 2 steps in $x^2_i$.
• 2 steps in $x^3_i$.

Similarly, we conclude that $L(\Gamma) = L(\Gamma')$ and $\text{Time}_\Gamma(n) = O(\text{Time}_{\Gamma'}(n))$. □

The converse of the previous proposition holds.

Proposition 6. $\text{Time}_\text{NSPFC}(f(n)) \subseteq \text{Time}_\text{NUSP}(f(n))$ for any function $f : \mathbb{N} \rightarrow \mathbb{N}$.

Proof. Let $\Gamma = (V, U, <, >, G, R, N, a, x_1, x_n)$ be a NSPFC with $G = (X_G, E_G)$, $X_G$ having $n$ nodes $x_1, x_2, \ldots, x_n$, $x_1 = \text{In}$ and $x_n = \text{Halt}$. We construct the NUSP $\Gamma' = (V, U', <, >, G', N', a', x_1, x_n)$; $x_1 = \text{In}$ and $x_n' = \text{Halt}$, where

$$V' = V,$$
$$U' = U \cup U' \cup X \cup Y \cup \{\#, \psi\},$$
$$X = \{X_i, X_i' \mid 1 \leq i \leq n\},$$
$$Y = \{Y_i \mid 1 \leq i \leq n - 1\},$$
$$U' = \{a' \mid a \in U\},$$
$$z(u, v) = \beta_1 u v \psi \beta_2, z = \beta_1 u v \psi \beta_2 \in U^*.$$

We now define the parameters of $\Gamma'$. First, we add a main subnetwork composed by the input node $x_i$, the nodes $\{x_i \mid 1 \leq i \leq n\}$, the nodes $\{x_i, x_i^2 \mid 1 \leq i \leq n - 1\}$ and the nodes $\{x_i, x'_{i} \mid 1 \leq i \neq j \leq n, i \neq n\}$, and the nodes $\{x_i, x_i \mid i \in E_G\}$.

• node $x_i$: $\overline{S} = \{(\varepsilon,>); (\#,> X_i')\}$,
  $\overline{A} = \{\# > X_i'\}$,
  $\overline{P} = V, F = \{\#, X_i\}$,
  $\overline{\beta'} = \{\psi\}$.

• node $x^1_i$: $\overline{S} = \{(\varepsilon, X_i') \cup \{\varepsilon, X_i\} \cup \{\#\},$ $\overline{A} = \{\# X_i \#,\}$,
  $\overline{P} = \{X_i, X_i'\}, F = \{\#\}$,
  $\overline{\beta'} = \{\psi\}$.

• nodes $x^2_i$: $1 \leq i \leq n - 1$: $\overline{S} = \{(a, b) \cup (u v, \psi v)\} \mid (a, b) \in S_{X_i}, (u, v) \in U^*\},$ $\overline{A} = \{z^k u v \psi v X_i \mid 1 \leq k \leq \text{card}(A_{x_k}) \}, z^k \in A_{x_k}\},$
  $\overline{P} = \{X_i, \psi, U'\}, F = X \setminus \{X_i\} \cup \{\#\}$,
  $\overline{\beta'} = \{\psi\}$.

• nodes $x^3_i$: $1 \leq i \leq n - 1$: $\overline{S} = \{(\varepsilon, X_i) \cup \{\varepsilon, X_i'\} \cup \{\varepsilon, X_i^2\} \cup \{\varepsilon, X_i'\} \cup \{\#\},$ $\overline{A} = \{\# X_i \#,\}$,
  $\overline{P} = \{X_i, X_i', U'\}, F = \{\#\}$,
  $\overline{\beta'} = \{\psi\}$.

• nodes $x^4_i$: $1 \leq i \leq n - 1$: $\overline{S} = \{(a, v) \cup (\#, v \pi)\} \mid a \in U^* \cup \{(a v, \pi) \cup (\# v \pi, t)\} \mid \{a, v\} \in U^* \cup \{U^* \cup \{X_i, X_i^2\}\} \cup \{\#\},$
  $\overline{P} = \{X_i, X_i^2, U'\}, F = \{t\}$,
  $\overline{\beta'} = \{\psi\}$.
\[ A = \{ \pi u\#, \pi v \pi X'_i \}, \]
\[-P = \{ X'_i \}, F = \{ \pi r\# \}, \]
\[-\beta' = (w). \]

- nodes \( x_{ij}, 1 \leq i \neq j \leq n, i \neq n, \{ x_i, x_j \} \in E_G \):
  \( S = \{ \{ (x, X'_i); (#, X'_j) \} \cup \{ (x, \psi); (#, \#) \} \}, \)
  \( A = \{ # X'_i, #\}, \)
  \(-P = P_{x_{ij}}, F = F_{x_{ij}}, \)
  \(-\beta' = \beta(x_{ij}). \)

The edges between them are:
\[
\{ \{ x_i, x_j \} \} \cup \\
\{ \{ x_i', x_j' \} \} \cup \\
\{ \{ x_i, x_j \} \mid 1 \leq i \leq n - 1 \} \cup \\
\{ \{ x_i', x_j' \} \mid 1 \leq i \leq n - 1 \} \cup \\
\{ \{ x_i, x_j \} \mid 1 \leq i \neq j \leq n, i \neq n, \text{ and } \{ x_i, x_j \} \in E_G \} \cup \\
\{ \{ x_{ij}, x_{ij}' \} \mid 1 \leq i \neq j \leq n, i \neq n, \text{ and } \{ x_i, x_j \} \in E_G \}. 
\]

Next, for each node \( x_i \mid 1 \leq i \leq n - 1 \) in \( \Gamma \), we add a subnetwork to \( \Gamma' \) according to the cases considered in the sequel.

Case 1. Let \( E_i \) be the number of nodes connected to the node \( x_i^2 \) of the form \( x_{ij} \), we add the corresponding nodes \( \{ x_{ij}^{r_1}(k), x_{ij}^{r_2}(k) \mid 1 \leq k \leq E_i \} \) to the subnetwork. Note that \( E_i \) is also equal to the number of edges between the node \( x_i \) and the nodes \( x_j \mid 1 \leq i \neq j \leq n \) in the network \( \Gamma \). If the edges \( \{ x_i, x_j \} \in \Gamma \) have \( \beta = (w) \), the subnetwork is defined as follows:

- nodes \( x_{ij}^{r_1}(k), 1 \leq k \leq E_i, 1 \leq i \neq j \leq n, i \neq n, \{ x_i, x_j \} \in E_G \):
  \[ S = \{ \{ (x, X'_i); (#, Y_{k+1}) \} \cup \{ (x, \psi); (#, \#) \}, \text{ if } k = 1 \land E_i > 1 \}
    \{ \{ (x, Y_k); (#, Y_{k+1}) \} \cup \{ (x, \psi); (#, \#) \}, \text{ if } 2 \leq k \leq E_i - 1 \}
    \{ \{ (x, Y_{E_i}); (#, \#) \}, \text{ if } k = E_i \land E_i > 1 \}
    \{ \{ (x, X'_i); (#, \#) \}, \text{ if } E_i = 1 \}
  \]
  \[-P = F_{x_{ij}}, F = \{ # X'_i \} \cup \{ Y \setminus \{ Y_k, Y_{k+1} \} \} \cup \{ U \}, \]
  \[-\beta' = (w). \]

- nodes \( x_{ij}^{r_2}(k), 1 \leq k \leq E_i, 1 \leq i \neq j \leq n, i \neq n, \{ x_i, x_j \} \in E_G \):
  \[ S = \{ \{ (x, X'_i); (#, Y_{k+1}) \} \cup \{ (x, \psi); (#, \#) \}, \text{ if } k = 1 \land E_i > 1 \}
    \{ \{ (x, Y_k); (#, Y_{k+1}) \} \cup \{ (x, \psi); (#, \#) \}, \text{ if } 2 \leq k \leq E_i - 1 \}
    \{ \{ (x, Y_{E_i}); (#, \#) \}, \text{ if } k = E_i \land E_i > 1 \}
    \{ \{ (x, X'_i); (#, \#) \}, \text{ if } E_i = 1 \}
  \]
  \[-P = \emptyset, F = P_{x_{ij}} \cup \{ # X'_i \} \cup \{ Y \setminus \{ Y_k, Y_{k+1} \} \} \cup \{ U \}, \]
  \[-\beta' = (w). \]

The edges between them are:
\[
\{ \{ x_i^2, x_{ij}^{r_1}(1) \}, \{ x_i^2, x_{ij}^{r_2}(1) \} \} \cup \\
\{ \{ x_{ij}^{r_1}(k), x_{ij}^{r_2}(k + 1) \}, \{ x_{ij}^{r_1}(k), x_{ij}^{r_2}(k + 1) \} \mid 1 \leq k \leq E_i - 1 \} \cup \\
\{ \{ x_{ij}^{r_1}(k), x_{ij}^{r_2}(k + 1) \}, \{ x_{ij}^{r_1}(k), x_{ij}^{r_2}(k + 1) \} \mid 1 \leq k \leq E_i - 1 \} \cup \\
\{ \{ x_{ij}^{r_1}(E_i), x_i^2 \}, \{ x_{ij}^{r_2}(E_i), x_i^2 \} \}. 
\]
Case 2. If the k edge \( \{x_i, x_j\} \) has \( \beta = (s) \), we replace each of the nodes \( x_{ij}^{k} \) for \( p \geq 1 \) nodes of the form \( x_{ij}^{k} (k) \), \( 1 \leq t \leq p \), where \( P_{\{x_i, x_j\}} = \{Z_1, Z_2, \ldots, Z_p\} \), \( p \geq 1 \). The parameters \( S, A \) and \( \beta \) of these new nodes remain the same, while the input and output filters are defined as follows:

\[
x_{ij}^{k} (k) : P = \emptyset, F = \{Z_t, \#, X_i\} \cup \{Y \setminus \{Y_{k}, Y_{k+1}\}\} \cup \{U_v\}
\]

and the edges involving \( x_{ij}^{k} (k) \) are replaced with \( p \) edges involving each of the nodes \( x_{ij}^{k} (k) \).

In both cases, if the k edge has \( P_{\{x_i, x_j\}} = \emptyset \), then the nodes \( x_{ij}^{k} (k) \) and \( x_{ij}^{k} (k) \) are removed. Analogously, if \( F_{\{x_i, x_j\}} = \emptyset \), then \( x_{ij}^{k} (k) \) is removed. Furthermore, if \( P_{\{x_i, x_j\}} = \emptyset \) and \( F_{\{x_i, x_j\}} = \emptyset \), the whole subnetwork given in in Figure 3 is removed because a string is always able to leave the node \( x_i \in \Gamma \) through the edge \( k \).

Let us assume the string \( w = < z > \) to be the input string in \( \Gamma \). In the input node \( x_1 \) of \( \Gamma \), the symbol \( X_1^i \) is attached to the string and then sent to the node \( x_1^j \), where it is replaced with \( X_1 \). Then, the new string is sent to \( x_1 \) and the simulation process starts. Thus, the node \( x_1 \in \Gamma \) contains the string \( w = < z > \) while \( w = < z > X_1 \) lies in the analogous node \( X_1^i \). In a more general setting, we assume that a string \( y' = yX_1, y \in U^* \), enters \( x_1 \) at a given step of the computation of \( \Gamma' \) on \( w \) if and only if the string \( y \) enters \( x_1 \in \Gamma \) at a given step of the computation of \( \Gamma \) on \( w \). Note that any string entering \( x_1 \) from \( x_1 \) will have the character \( X_1 \) replaced with \( \# \), which will result in that string being locked in that node. Therefore, the two nodes \( x_1 \) and \( x_1 \) do not interfere with the computation anymore.

Let \( y \) be transformed into \( z_1 = a_1 a v s \beta_2 \) and \( z_2 = \beta_1 u b a_2 \) in node \( x_i \) and \( z_1 \) and/or \( z_2 \) can pass the filter on the edge between \( x_i \) and \( x_j \). In \( \Gamma' \), the string \( y' = y_c \), with \( c \in \{X_i, \psi\} \), is transformed into \( z_1' = a_1 a v v_1 \beta_2 X_i \) and \( z_2' = \beta_1 u b a_2 c \). The strings enter \( x_1 \), where the characters \( X_i \) and \( \psi \) are replaced with \( X_1 \). Thus, we get the same strings with any \( c \in \{X_i, \psi\} \). Then, it continues into \( x_2 \), where the symbols in \( u_i \) and \( v_i \), if present, are replaced with the original counterpart \( u \) and \( v \). In detail, the string \( z_1 \) is split into the strings \( z_1(1) = a_1 a v v_1 \pi X_1^i \) and \( z_1(2) = \# v_1 \pi X_1^i \). Both strings cannot leave the node because of the characters \( \pi \) and \# in the string. In the next splicing step, the rule \( \{(a v, \pi); (\# v, t)\} \) is applied to the two strings, yielding \( z_1(1)' = a_1 a v s \beta_2 X_i \) and \( z_2(2)' = \# v_1 \pi X_1^i \). The former constitutes the string \( z = z_1 X_1^i \) where \( z_1 \) is the string yielded by \( x_i \in \Gamma \), completing the simulation of the splicing step. On the other hand, the latter cannot exit the node or be modified by any rule, so it remains locked for the rest of the computation. The logic is analogous for the other string generated by the splicing rule. The new string \( z = z_1 X_1^i \) is sent to the nodes \( x_{ij} \) associated to the edges \( \{x_i, x_j\} \) in the original network. Since the filters of these nodes are identical to the edge filters in \( \Gamma \), the string \( z_1 X_1^i \) can enter a node \( x_{ij} \in \Gamma \) if and only if the string \( z_1 \) can pass the filter between \( x_i \) and \( x_j \) in \( \Gamma \). Note that the converse is also true. Consequently, the symbol \( X_1^i \) is replaced with \( X_i \) and the new string is sent to \( x_1 \) where a similar procedure starts. Note that if the string returns to the previous node in this sequence, it is ultimately blocked or lost. In \( x_1 \), the strings entering from \( x_2 \) are split into two strings containing the forbidden symbol \# and consequently they remain locked in that node. The same situation happens if a string enters \( x_2 \) from a node \( x_{ij} \). On the other hand, if a string enters a node \( x_{ij} \) from \( x_2 \) for a given \( j \), the string is split into two strings containing the symbol \#. In the next communication step, these two strings are ejected from the network because they are not accepted by any node connected to \( x_{ij} \). Alternatively, if the string \( y \) remains unchanged in \( x_i \in \Gamma \), the same event happens in \( x_1 \) in \( \Gamma' \). If the string is of the form \( y' = y \psi \), it starts a ping-pong process with the nodes \( x_{ij}^{rt1} \) and \( x_{ij}^{rt2} \) (or the nodes \( x_{ij}^{rt1} (k) \) if the edge \( \{x_i, x_j\} \in \Gamma \) has strong filters) until a splicing rule can be applied on it. On the other hand, if the string is of the form \( y' = y X_i \), it still enters the node \( x_1 \) because of the symbol \( X_i \), starting the same computational process described above.
On the other hand, a copy of the string $zX'_i$ is also sent to the subnetwork illustrated in Figure 3, where the case of a string not passing the filters of any edge between the node $x_i$ and the connected nodes in the original network $\Gamma$ is handled. We distinguish here between the cases of strong and weak filters. For each edge $k$ with $\beta = (w), 1 \leq k \leq E_i$, the node $x'_{\beta j}^{E}(k)$ checks if the string cannot enter the corresponding node $x_{ij}(k)$ because of containing forbidden symbols, while the node $x^{rt}_{\beta j} (k)$ verifies if the string contains the characters required by the $P$ filter of the node $x_{ij}(k)$. Thoroughly, the string exits $x^s_j$ and enters the first pair of nodes $x^{rt}_{\beta j} (1)$ and $x^{rt}_{\beta j} (1)$ granted that the associated string $z$ could not pass the filters of the first edge in the network $\Gamma$. In that node, $X'_i$ is replaced with $Y_2$, ensuring that the new string could only enter the next pair of nodes $x^{rt}_{\beta j} (2)$ and $x^{rt}_{\beta j} (2)$ associated to the second edge. Subsequently, $Y_k | 2 \leq k \leq E_i - 1$ is replaced by the next character $Y_{k+1}$ in the remaining pair of nodes $x^{rt}_{\beta j} (k)$ and $x^{rt}_{\beta j} (k)$, forcing the string to continue through them in sequence. If the string can enter one of the connected nodes through an edge in $\Gamma$, it will be lost at some point of this subnetwork as it will be refused by the input filters of the nodes corresponding to that edge. Otherwise, it will reach the last pair of nodes where the symbol $Y_{E_i}$ will be changed to $\psi$ yielding a string of the form $z\psi$ which will be returned to $x^s_i$. Now, the string can either be used for another splicing step in which case the resulting strings will exit the node and enter $x^s_j$ as it will contain a character $a_{\psi} \in V_\psi$ or a ping-pong process will start between this node and the nodes $x^{rt}_{\beta j} (E_i)$ and $x^{rt}_{\beta j} (E_i)$ that may continue forever or until a splicing rule may be applied to the string. Thus, the following computational step in $\Gamma'$ will be simulated in $\Gamma$: $z$ could not leave the node $x_i$, and it remained in that node for new splicing steps. In the case of an edge $k$ with strong filters, the computation follows the same procedure described above with the difference that the node $x^{rt}_{\beta j} (k)$ is replaced with $p \geq 1$ nodes of the form $x^{rt}_{\beta j} (k), 1 \leq t \leq p,$ where $P_{\{x_i, x_j\}} = \{Z_1, Z_2, \ldots, Z_p\}, p \geq 1$. A node $x^{rt}_{\beta j} (k)$ refuses any string $z$ containing the character $Z_t$. Thus, the string can only continue through this network by means of a node $x^{rt}_{\beta j} (k)$ for some $1 \leq t \leq p$ if it only contains a proper subset of $P_{\{x_i, x_j\}}$. Note that a string $z\psi$ can also enter the previous nodes $x^{rt}_{\beta j} (E_i - 1)$ and $x^{rt}_{\beta j} (E_i - 1)$ granted that $E_i > 1$, but that string is split into two new strings with the forbidden symbol # through the application of the rule $\{(\epsilon, \psi); (\#, \#)\}$ and, consequently, they will remain locked in that node.

We analyze now the case where a string $zX'_i$ enters $x_{ij}$ from a node $x^s_i$, for some $1 \leq i \leq n - 1$. In that node, $X'_i$ is replaced with $X_n$ and the resulting string enters $x^s_n$, ending the computation. Note that by the considerations above, a string enters $x_{ij}$ if and only if a string from $x_i$ was able to pass the filter on the edge between $x_i$ and $x_n$ in $\Gamma$.

We analyze now the computational steps required for simulating a computation in $\Gamma$. The input node $x_i$ requires 1 splicing step (or 2 computational steps). In the worst case, a splicing step in one of the nodes $\{x_i | 1 \leq i \leq n - 1\}$ can be simulated in the associated subnetwork in $4 + E_i$ splicing steps (or $8 + 2E_i$ computational steps) distributed in the following way:

- 2 steps in $x^s_i$.
- 2 steps in $x^s_j$.
- 4 steps in $x^s_i$.
- 2 steps in $x_{ij}$ if the string $z$ yielded by the splicing step in the node $x_i$ could pass any of the edge filters between $x_i$ and $x_j$ in $\Gamma$; $1 \leq i \neq j \leq n, i \neq n$.
- $2E_i$ steps if $z$ could not pass any of the edge filters in $\Gamma$.

Since $E_i$ is bounded by $n - 1$ where $n$ is the number of nodes in $\Gamma$, a NSPFC can be simulated by NUSP in $O(n)$ time. As a result of this analysis, we conclude that $L(\Gamma) = L(\Gamma')$ and $Time_\Gamma(n) \in O(Time_\Gamma(n))$. □

In virtue of the results obtained in the proposed simulations, we are now able to state the main conclusion of this research:
Theorem 1.

\[ \text{Time}_{\text{NSP}}(f(n)) = \text{Time}_{\text{NUSP}}(f(n)) = \text{Time}_{\text{NSPFC}}(f(n)) \]

for any function \( f : \mathbb{N} \rightarrow \mathbb{N} \).

6. Conclusions and Further Work

As being a theoretical piece of work, the methodology used in this paper is the standard one: identifying the problem, discussing the theoretical implications and possible practical applications, stating the results, proving the statements, and discussing different aspects of the results.

We demonstrate here an efficient method to simulate different variants of NSP with random-context filters with each other. We describe a methodology to translate any network belonging to one of the three variants into an equivalent one of the other two models. The definitions of the former regarding splicing rules, auxiliary strings, random-context filters and connections between nodes are used to determine the corresponding ones for the new network. Later on, they are assigned to the nodes and edges according to an established procedure. Although the constructed networks may appear to be more complex and to have a significantly bigger size than the original network, the time complexity remains unchanged. This statement is proven true by the capability of these new networks to simulate any computational step in the original architecture in a constant number of computational steps. This method is theoretically important and attractive, as it allows direct and time efficient conversions of one variant of NSP into another one, avoiding intermediate computational models. Efficient simulations between different bio-inspired computational models are essential in the theoretical and practical studies in this field. It is common for these paradigms to be used as problem solvers, and simulations between two models would allow the scientific community to translate a solution proposed for one of the models to the other, avoiding the time and effort required otherwise. Furthermore, research about this topic could also have important practical applications in architecture design, such as the hypothetical establishment of systems, composed of a mix of different bio-inspired architectures, in an effort towards improving the efficiency of a given task.

As one can see, the underlying graph of the simulating network might be different than that of the original network. An attractive line for further research, which may be considered not only for the variants considered here, would be to impose some conditions on the topology of the underlying graph (to be the same as the original one, to have a predefined form, etc.).

Last but not least, we have started a study of possible numerical simulations and implementations of some of these models. A few steps have already been made by considering some possible ways of assigning probabilities to some similar models, see, e.g., [25–27].

Author Contributions: Conceptualization, V.M. and J.A.S.M.; formal analysis, V.M. and M.P.; investigation, J.R.S.C. and V.M.; writing—original draft preparation, J.A.S.M.; writing—review and editing, V.M.; supervision, V.M.; funding acquisition, M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by a grant of the Romanian Ministry of Education and Research, CCCDI-UEFISCDI, Project No. PN-III-P2-2.1-PED-2019-2391, within PNCDI III.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work was partially supported by the Comunidad de Madrid under Convenio Plurianual with the Universidad Politécnica de Madrid in the actuation line of Programa de Excelencia para el Profesorado Universitario.

Conflicts of Interest: The authors declare no conflict of interest.
References

1. Martín-Vide, C.; Pazos, J.; Păun, G.; Rodrixuguez-Patón, A. A new class of symbolic abstract neural nets: Tissue P systems. *Lect. Notes Comput. Sci.* 2002, 2387, 290–299.

2. Păun, G. *Membrane Computing. An Introduction*; Springer: Berlin/Heidelberg, Germany, 2002.

3. Cşuhaj-Varjú, E.; Mitrana, V. Evolutionary systems: A language generating device inspired by evolving communities of cells. *Acta Inform.* 2000, 36, 913–926. [CrossRef]

4. Cşuhaj-Varjú, E.; Salomaa, A. Networks of parallel language processors. *Lect. Notes Comput. Sci. (LNCS)* 1997, 1218, 299–318.

5. Morrison, J. *Flow-Based Programming: A New Approach to Application Development*; CreateSpace: Scotts Valley, CA, USA, 2010.

6. Hillis, D. *The Connection Machine*; MIT Press: Cambridge, MA, USA, 1986.

7. Gray, R.; Kotz, D.; Nog, S.; Rus, D.; Cybenko, G. Mobile agents: The next generation in distributed computing. In *Proceedings of the IEEE International Symposium on Parallel Algorithms Architecture Synthesis*, Aizu-Wakamatsu, Japan, 17–21 March 1997; pp. 8–24.

8. Manea, F.; Martín-Vide, C.; Mitrana, V. Accepting networks of evolutionary word and picture processors: A survey. *Sci. Appl. Lang. Methods* 2010, 2, 525–560.

9. Arroyo, F.; Castellanos, J.; Mitrana, V.; Santos, E.; Sempere, J.M. Networks of bio-inspired processors. *Triangle Lang. Lit. Comput.* 2012, 7, 4–22.

10. Castellanos, J.; Martín-Vide, C.; Mitrana, V.; Sempere, J.M. Networks of evolutionary processors. *Acta Inform.* 2003, 39, 517–529. [CrossRef]

11. Manea, F.; Martín-Vide, C.; Mitrana, V. Accepting networks of splicing processors. In *New Computational Paradigms, First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, 8–12 June 2005*; Barry Cooper, S., Löwe, B., Torenvliet, L., Eds.; Volume 3526 of Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; pp. 300–309.

12. Manea, F.; Martín-Vide, C.; Mitrana, V. Accepting hybrid networks of evolutionary processors of constant size. In *DNA Computing, 12th International Meeting on DNA Computing, DNA12, Seoul, Korea, 5–9 June 2006, Revised Selected Papers*; Chengde, M., Takashi, Y., Eds.; Volume 4287 of Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2006; pp. 47–57.

13. Head, T.; Păun, G.; Pixton, D. Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination. In *Handbook of Formal Languages, Volume 2. Linear Modeling: Background and Application*; Grzegorzyk, R., Arto, S., Eds.; Springer: Berlin/Heidelberg, Germany, 1997; pp. 295–360.

14. Head, T. Restriction Enzymes in Language Generation and Plasmid Computing. In *Biomolecular Information Processing: From Logic Systems to Smart Sensors and Actuators*; Wiley: Hoboken, NJ, USA, 2012; pp. 245–263.

15. Cşuhaj-Varjú, E.; Kari, L.; Păun, G. Test tube distributed systems based on splicing. *Comput. Artif. Intell.* 1996, 15, 211–232.

16. Păun, G. Distributed architectures in DNA computing based on splicing: Limiting the size of components. In Proceedings of the 1st International Conference on Unconventional Models of Computation, Auckland, New Zealand, 5–9 January 1998; pp. 323–335.

17. Păun, G. DNA computing; Distributed splicing systems. *Lect. Notes Comput. Sci.* 2005, 1261, 353–370.

18. Manea, F.; Martín-Vide, C.; Mitrana, V. Accepting networks of splicing processors: Complexity results. *Theor. Comput. Sci.* 2007, 371, 72–82. [CrossRef]

19. Margenstern, M.; Mitrana, V.; Pérez-Jiménez, M.J. Accepting hybrid networks of evolutionary processors. In *DNA Computing, 10th International Workshop on DNA Computing, DNA 10, Milan, Italy, 7–10 June 2004, Revised Selected Papers*; Claudio, F., Giancarlo, M., Claudio, Z., Eds.; Volume 3384 of Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2004; pp. 235–246.

20. Bottino, P.; Labella, A.; Manea, F.; Mitrana, V.; Petre, I.; Sempere, J.M. Complexity-preserving simulations among three variants of accepting networks of evolutionary processors. *Nat. Comput.* 2011, 10, 429–445. [CrossRef]

21. Dragoi, C.; Manea, F.; Mitrana, V. Accepting networks of evolutionary processors with filtered connections. *J. Univers. Comput. Sci.* 2007, 13, 1598–1614.

22. Castellanos, J.; Manea, F.; de Mingo López, L.F.; Mitrana, V. Accepting networks of splicing processors with filtered connections. In *Machines, Computations, and Universality; Jérôme, D.-L., Maurice, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2007*; pp. 218–229.

23. Gómez-Canaual, S.; Mitrana, V.; Păun, M.; Sánchez Martín, J.A.; Sánchez Couso, J.R. Networks of uniform splicing processors: Computational power and simulation. *Mathematics 2020*, 8, 1217. [CrossRef]

24. Rozenberg, G.; Salomaa, A. *Handbook of Formal Languages*; Springer: Berlin, Germany, 1997.

25. Arroyo, F.; Gómez-Canaual, S.; Mitrana, V.; Păun, M.; Sánchez Couso, J.R. Towards probabilistic networks of polarized evolutionary processors. In *Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS)*; Orléans, France, 16–20 July 2018; pp. 764–771.

26. Gómez-Canaual, S.; Mitrana, V.; Păun, M.; Vararuk, S. High performance and scalable simulations of a bio-inspired computational model. In *Proceedings of the 2019 International Conference on High Performance Computing & Simulation (HPCS)*; Dublin, Ireland, 15–19 July 2019; pp. 543–550.

27. Sánchez, J.A.; Arroyo, F. Simulating probabilistic networks of polarized evolutionary processors. *Procedia Comput. Sci.* 2019, 159, 1421–1430. [CrossRef]