INTERNATIONAL JOURNAL OF FOUNDATIONS OF COMPUTER SCIENCE

Volume 22, Number 1, January 2011

CONTENTS

This journal is covered in SciSearch® (also known as Science Citation Index - Expanded), ISI Alerting Services, CompuMath Citation Index®, Current Contents/Engineering, Computing & Technology, MathSciNet, Mathematical Reviews, DBLP Bibliography Server and Zentralblatt MATH.

International Journal of Foundations of Computer Science
Vol. 22, No. 1 (2011) 155–165

DOI: 10.1142/S0129054111007915

World Scientific
www.worldscientific.com

FILTER POSITION IN NETWORKS OF SUBSTITUTION PROCESSORS DOES NOT MATTER

FERNANDO ARROYO MONTORO[1], JUAN CASTELLANOS[2]
VICTOR MITRANA[3,4], EUGENIO SANTOS[3] and JOSE M. SEMPERE[5]

[1] *Depto. Lenguajes, Proyectos y Sistemas Informticos,*
Escuela Universitaria de Informtica
Universidad Politcnica de Madrid,
Ctra. de Valencia Km. 7, 28031 Madrid, Spain
farroyo@eui.upm.es

[2] *Department of Artificial Intelligence*
Polytechnical University of Madrid
28660 Boadilla del Monte, Madrid, Spain
jcastellanos@fi.upm.es

[3] *Depto. Organización y Estructura de la Información*
Escuela Universitaria de Informática
Universidad Politécnica de Madrid,
Crta. de Valencia km. 7 - 28031 Madrid, Spain
mitrana@fmi.unibuc.ro,esantos@eui.upm.es

[4] *Faculty of Mathematics and Computer Science*
University of Bucharest, Str. Academiei 14, 010014, Bucharest, Romania

[5] *Department of Information Systems and Computation*
Technical University of Valencia,
Camino de Vera s/n. 46022 Valencia, Spain
jsempere@dsic.upv.es

Received 10 June 2010
Accepted 5 September 2010
Communicated by Rudolf Freund

It is known ([4]) that moving the filters from the nodes to the edges in accepting hybrid networks of evolutionary processors does not decrease the computational power of the model which equals that of a Turing machine. A direct and time complexity preserving simulation is presented in [1]. All three types of processors (substitution, insertion, deletion) are essentially used in this simulation.
In this note we prove that such a direct simulation between networks containing substitution nodes only still exists.

Keywords: Substitution processor; accepting network of substitution processors; accepting network of substitution processors with filtered connections.

1991 Mathematics Subject Classification: 68Q45, 68Q50, 68Q52

1. Introduction

A basic architecture for parallel and distributed computing consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules. Local data are then sent through the network according to well-defined protocols. Only that data which are able to pass a filtering process can be communicated. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies. This general architecture may be met in several areas of Computer Science like Artificial Intelligence [7, 6], Symbolic Computation [5], Grammar Systems [10], Membrane Computing [11].

The origin of *accepting hybrid networks of evolutionary processors* (AHNEP for short) is such an architecture in connection with the work [2] (see also [3] that considers a computing model inspired by the evolution of cell populations), where a distributed computing device called network of language processors is proposed. Each processor placed in a node is called evolutionary processor, i.e. an abstract processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). More generally, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node is organized in the form of multisets of words (each word may appear in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place. Further, all the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies modeled as permitting and forbidding filters and filtering criteria, see [8]. The reader interested in a more detailed discussion about the accepting model is referred to [9].

It is clear that filters associated with each node of an AHNEP allow a strong control of the computation. Indeed, every node has an input and output filter; two nodes can exchange data if it passes the output filter of the sender *and* the input filter of the receiver. Moreover, if some data is sent out by some node and not able to enter any node, then it is lost. The AHNEP model considered in [8] is simplified in [4] by moving the filters from the nodes to the edges. Each edge is viewed as a two-way channel such that the input and output filters, respectively, of the two nodes connected by the edge coincide. Clearly, the possibility of controlling the computation in such networks seems to be diminished. For instance, there is no possibility to discard data during the communication steps. In spite of this fact, in the aforementioned work one proves that these new devices, called *accepting hybrid*

networks of evolutionary processors with filtered connections (AHNEPFC) are still computationally complete. This means that moving the filters from the nodes to the edges does not decrease the computational power of the model. Although the two variants are equivalent from the computational power point of view, no direct proof for this equivalence has been proposed until the work [1], where direct simulations between the two variants are presented. Moreover, both simulations are time efficient, namely each computational step in one model is simulated in a constant number of computational steps in the other. This is particularly useful when one wants to translate a solution from one model into the other. A translation via a Turing machine squares the time complexity of the new solution.

These simulations essentially use all types of processors (substitution, insertion, deletion). A natural problem regards the existence of such simulations when the two networks that simulate each other are formed by just two types of processors or even one type only. In this note we continue the investigation in this direction. More precisely, we consider simulations between networks containing substitution processors only. It turned out that even for these restricted variants a direct and computationally efficient simulation exists.

2. Accepting Networks of Substitution Processors

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. Any finite sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^*, the empty word is denoted by ε, while $alph(x)$ denotes the minimal alphabet with respect to inclusion $W \subseteq V$ such that $x \in W^*$.

We say that a rule $a \to b$, with $a, b \in V$ is a *substitution rule*. Given a rule σ as above and a word $w \in V^*$, we define the following *action* of σ on w:

$$\sigma(w) = \begin{cases} \{ubv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$

Note that a rule as above is applied to all occurrences of the letter a in different copies of the word w. An implicit assumption is that arbitrarily many copies of w are available.

For every rule σ and $L \subseteq V^*$, we define the *action of* σ *on* L by $\sigma(L) = \bigcup_{w \in L} \sigma(w)$. Given a finite set of substitution rules M, we define the *action of* M on the word w and the language L by $M(w) = \bigcup_{\sigma \in M} \sigma(w)$ and $M(L) = \bigcup_{w \in L} M(w)$, respectively. By convention, we state $\emptyset(w) = \{w\}$. For two disjoint subsets P and F of an alphabet V and a word z over V, we define the predicates:

$$\begin{aligned} \varphi^{(s)}(z; P, F) \equiv \quad & P \subseteq alph(z) && \wedge \quad F \cap alph(z) = \emptyset \\ \varphi^{(w)}(z; P, F) \equiv \quad & \text{if } P \neq \emptyset \text{ then } alph(z) \cap P \neq \emptyset && \wedge \quad F \cap alph(z) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding*

contexts/symbols). Informally, the first condition requires that all permitting symbols are present in z and no forbidding symbol is present in z, while the second one is a weaker variant of the first, requiring that at least one permitting symbol appears in z and no forbidding symbol is present in z. For every language $L \subseteq V^*$ and $\beta \in \{(s), (w)\}$, we define:

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

A *substitution processor over* V is a tuple (M, PI, FI, PO, FO), where:

- M is a set of substitution rules over the alphabet V.
- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor. Informally, the permitting input/output contexts are the set of symbols that should be present in a word, when it enters/leaves the processor, while the forbidding contexts are the set of symbols that should not be present in a word in order to enter/leave the processor.

We now define two variants of accepting networks of substitution processors as particular cases of AHNEPs. The reader interested in the definition of these networks in the general case of where all types of processors are allowed is referred to [8] (for AHNEP) and [4] (for AHNEPFC). Note that the word *hybrid* appearing in the name of AHNEPs does not make sense anymore as substitution is applied only at any position and not in either one end or any position of the word as it is the case for insertion and deletion in AHNEPs.

An *accepting network of substitution processors* (ANSP for short) is a 7-tuple $\Gamma = (V, U, G, \mathcal{N}, \beta, x_I, x_O)$, where:

• V and U are the input and network alphabets, respectively, $V \subseteq U$.

• $G = (X_G, E_G)$ is an undirected graph without loops, with the set of nodes X_G and the set of edges E_G. Each edge is given in the form of a binary set. G is called the *underlying graph* of the network.

• $\mathcal{N}$ is a mapping which associates with each node $x \in X_G$ the substitution processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.

• $\beta : X_G \longrightarrow \{(s), (w)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

$$\text{input filter: } \rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x),$$
$$\text{output filter: } \tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x).$$

That is, $\rho_x(z)$ (resp. τ_x) indicates whether or not the word z can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.

• In and $Out \in X_G$ are the *input node*, and the *output node*, respectively, of the ANSP.

An *accepting network of substitution processors with filtered connections* (ANSPFC for short) is a 8-tuple $\Gamma = (V, U, G, \mathcal{R}, \mathcal{N}, \beta, In, Out)$, where:

◇ V, U, G, In, and Out have the same meaning as for ANSPs.

$\diamond$ $\mathcal{R}$ is a mapping which associates with each node the set of substitution rules that can be applied in that node.
$\diamond$ $\mathcal{N} : E_G \longrightarrow 2^U \times 2^U$ is a mapping which associates with each edge $e \in E_G$ the disjoint sets $\mathcal{N}(e) = (P_e, F_e)$, $P_e, F_e \subset U$.
$\diamond$ $\beta : E_G \longrightarrow \{(s), (w)\}$ defines the *filter* type of an edge.

For the two variants we say that $card(X_G)$ is the size of Γ. When we want to refer to any of the two variants we use the notation ANSP[FC].

A *configuration* of an ANSP[FC] Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node at a given moment. A configuration can change either by a *substitution step* or by a *communication step*.

A substitution step is common to all models. When changing by a substitution step each component $C(x)$ of the configuration C is changed in accordance with the set of substitution rules M_x or $\mathcal{R}(x)$ associated with the node x. Formally, we say that the configuration C' is obtained in *one substitution step* from the configuration C, written as $C \Longrightarrow C'$, if and only if

$$C'(x) = M_x(C(x)) \qquad \text{or} \qquad C'(x) = \mathcal{R}(x)(C(x)), \text{ for all } x \in X_G.$$

When changing by a communication step, each node processor $x \in X_G$ of an ANSP sends one copy of each word it has (without keeping any copy of it), which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x provided that they can pass its input filter. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, if and only if

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

for all $x \in X_G$. Note that words which leave a node are eliminated from that node. If they cannot pass the input filter of any node, they are lost.

When changing by a communication step, each node processor $x \in X_G$ of an ANSPFC sends one copy of each word it has to every node processor y connected to x, provided they can pass the filter of the edge between x and y. It keeps no copy of these words but receives all the words sent by any node processor z connected with x providing that they can pass the filter of the edge between x and z.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) \setminus (\bigcup_{\{x,y\} \in E_G} \varphi^{\beta(\{x,y\})}(C(x), P_{\{x,y\}}, F_{\{x,y\}})))$$
$$\cup (\bigcup_{\{x,y\} \in E_G} \varphi^{\beta(\{x,y\})}(C(y), P_{\{x,y\}}, F_{\{x,y\}}))$$

for all $x \in X_G$. Note that a copy of a word remains in the sending node x only if it not able to pass the filter of any edge connected to x.

Let Γ be an ANSP[FC], the computation of Γ on the input word $z \in V^*$ is a sequence of configurations $C_0^{(z)}, C_1^{(z)}, C_2^{(z)}, \ldots$, where $C_0^{(z)}$ is the initial configuration of Γ defined by $C_0^{(z)}(In) = \{z\}$ and $C_0^{(z)}(x) = \emptyset$ for all $x \in X_G$, $x \neq In$, $C_{2i}^{(z)} \Longrightarrow C_{2i+1}^{(z)}$ and $C_{2i+1}^{(z)} \vdash C_{2i+2}^{(z)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(z)}$ is uniquely determined by the configuration $C_{i-1}^{(z)}$. A computation as above is said to be an *accepting computation* if there exists a configuration in which the set of words existing in the output node *Out* is non-empty. The *language accepted* by Γ is

$$L(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is an accepting one}\}.$$

We denote by $\mathcal{L}(ANSP)$ and $\mathcal{L}(ANSPFC)$ the class of languages accepted by ANSPs and ANSPFCs, respectively.

3. Equivalence Between the Two Variants

In this section we prove the main result of the paper, namely $\mathcal{L}(ANSP)$ and $\mathcal{L}(ANSPFC)$ coincide. The proof consists of direct simulations between the two variants. Furthermore, we show that both simulations are time complexity preserving.

Proposition 1. $\mathcal{L}(ANSP) \subseteq \mathcal{L}(ANSPFC)$.

Proof. Let $\Gamma = (V, U, G, \mathcal{N}, \beta, In, Out)$ be an ANSP with $X_G = \{x_1, x_2, \ldots, x_n\}$, for some $n \geq 2$ such that $In = x_1$ and $Out = x_n$. We construct the ANSPFC $\Gamma' = (V, W, G', \mathcal{R}, \mathcal{N}', \beta', In', Out)$, where

$$W = U \cup \underbrace{\{a' \mid a \in U\}}_{U'} \cup \underbrace{\{a'' \mid a \in U\}}_{U''} \cup \underbrace{\{\overline{a} \mid a \in U\}}_{\overline{U}},$$
$$G' = (X_{G'}, E_{G'}),$$

and the nodes in $X_{G'}$ and their set of substitution rules are defined as follows. For each $x_i \in X_G$, $i \neq n$, we consider the following nodes in $X_{G'}$:

$$\begin{aligned}
x_i' &: \{a \to b' \mid a \to b \in M_{x_i}\},\\
x_i^a(prepare_check_out), a \in U &: \{a' \to a\},\\
x_i(check_out) &: \{a \to a'' \mid a \in U\},\\
x_i(prepare_check_in) &: \{a'' \to a \mid a \in U\},\\
x_i^1(prepare_back) &: \{a \to \overline{a} \mid a \in U\},\\
x_i(back) &: \{\overline{a} \to a \mid a \in U\}.
\end{aligned}$$

Some more nodes are in $X_{G'}$ depending on the filter type of x_i, namely $x_i^2(prepare\_back)$, if $\beta(x_i) = (w)$, or all nodes $x_i^Z(prepare\_back)$, $Z \in PO_{x_i}$, if $\beta(x_i) = (s)$. The set of substitution rules in all these nodes is the same: $\{a \to \overline{a} \mid a \in U\}$.

Table 1.

Edge	P	F	β'
$\{x'_i, x_i^a(prepare\_check\_out)\}, a \in U$	$\{a'\}$	$\emptyset$	(w)
$\{x_i^a(prepare\_check\_out), x_i(check\_out)\}$, $a \in U$	PO_{x_i}	$FO_{x_i} \cup U' \cup U''$	$\beta(x_i)$
$\{x_i(check\_out), x_i(prepare\_check\_in)\}$	U''	$\emptyset$	(w)
$\{x_i(prepare\_check\_in), x'_j\}$, $\{x_i, x_j\} \in E_G$	PI_{x_j}	$FI_{x_j} \cup U' \cup U''$	$\beta(x_j)$
$\{x_i^a(prepare\_check\_out), x_i^1(prepare\_back)\}$, $a \in U$	FO_{x_i}	$U' \cup \overline{U}$	(w)
$\{x_i^a(prepare\_check\_out), x_i^2(prepare\_back)\}$, $a \in U$, if $\beta(x_i) = (w)$	W	$PO_{x_i} \cup U' \cup \overline{U}$	(w)
$\{x_i^a(prepare\_check\_out), x_i^Z(prepare\_back)\}$, $a \in U, Z \in PO_{x_i}$, if $\beta(x_i) = (s)$	W	$\{Z\} \cup U' \cup \overline{U}$	(w)
$\{x_i^k(prepare\_back), x_i(back)\}$, $k \in \{1, 2\}$, if $\beta(x_i) = (w)$	$\overline{U}$	$\emptyset$	(w)
$\{x_i^Z(prepare\_back), x_i(back)\}$, $Z \in PO_{x_i}$, if $\beta(x_i) = (s)$	$\overline{U}$	$\emptyset$	(w)
$\{x_i(back), x'_i\}$	W	$U' \cup \overline{U}$	(w)

We now define the edges of $E_{G'}$ and their filters.

We now show how Γ' simulates any computation of Γ on some input word. To this aim, let us consider that a word, say $z \in U^*$, is in the node x_i of Γ before a substitution step. We analyze all possible cases. We first assume that z is transformed by one substitution rule, say $a \to b$, into z_1 which can pass the output filter of x_i. Further z_1 enters x_j and a copy originated from z_1 will eventually enter Out.

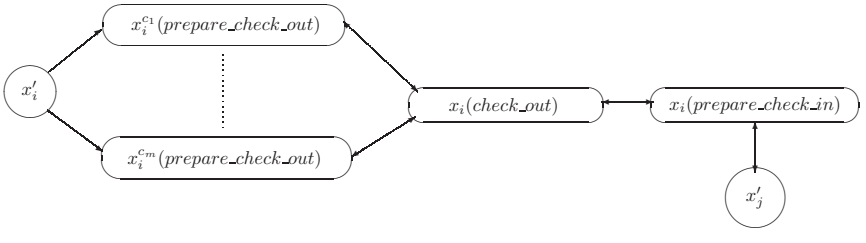

Diagram 1.

This computation is simulated in Γ' as follows. The rule $a \to b'$ is applied to z in x'_i, all the obtained words go out from x'_i and enter $x_i^b(prepare\_check\_out)$ where b' is restored to the original b. Note that no word can return to x'_i. Among them is also z_1. As z_1 is able to pass the output filter of x_i in Γ, the same word arrives in $x_i(check\_out)$ of Γ'. Two possible cases may appear now: (i) a symbol of z_1 is replaced by its double primed copy, or (ii) no substitution is applied to z_1 (this is possible when z_1 does not contain all symbols from U). In the first case, all words go to $x_i(prepare\_check\_in)$. Again two situations may appear: (i.1) the

modified symbol is restored, or (i.2) no substitution is applied. In case (i.1), z_1 which has just been restored enters x'_j and the process of simulation is resumed for the new node. In case (i.2), an infinite "ping-pong" process between $x_i(check\_out)$ and $x_i(prepare\_check\_in)$ may occur and/or z_1 is obtained in $x_i(prepare\_check\_in)$. Note that the possible infinite "ping-pong" process neither prevents z_1 to enter x'_j nor produces a copy that could parasitically lead to acceptance. We return now to the second case (ii) which may simply initiate a similar infinite "ping-pong" process between $x_i^b(prepare\_check\_out)$ and $x_i(check\_out)$. This discussion is schematically illustrated by Diagram 1, where an arrow indicates how words can move between nodes. Note that these arrows do not mean the edges in the underlying graph which is undirected. Furthermore, we assume that $U = \{c_1, c_2, \ldots, c_m\}$, for some $m \geq 1$.

Another possible situation when z lies in x_i of Γ is to get another word, say z_1, after several substitutions in succession (the word could not pass the output filter of x_i after any intermediate substitution) which is now able to go out from x_i. As above, we further assume that z_1 enters x_j and a copy originated from z_1 will eventually enter Out. This situation is captured in Γ' as shown in Diagram 2 for the case $\beta(x_i) = (w)$.

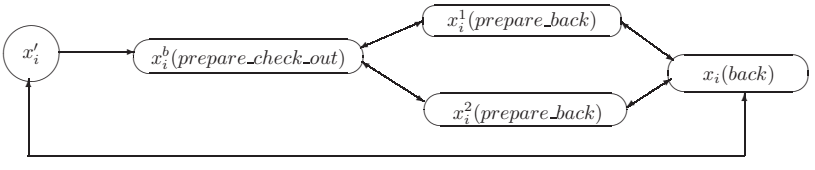

Diagram 2.

More precisely, every word obtained in x_i after an intermediate substitution step will return to x'_i via the following itinerary: $x_i^b(prepare\_check\_out)$, for some $b \in U$, then $x_i^\alpha(prepare\_back)$, where α is either in $\{1, 2\}$, provided that $\beta(x_i) = (w)$, or in PO_{x_i}, provided that $\beta(x_i) = (s)$, then $x_i(back)$ and finally x'_i. Again, this process does not lead to the acceptance of illegal words.

The last situation that may occur in x_i is that no word originating from z can either leave x_i or enter some node x_j. The same happens when z lies in x'_i of Γ'. □

From the above considerations, it follows that Γ' simulates in at most 3 processing steps and 4 communication steps a processing step of Γ, while every communication step of Γ is simulated in a communication step in Γ'. Therefore, the simulation of Γ by Γ' is done efficiently.

Proposition 2. $\mathcal{L}(ANSPFC) \subseteq \mathcal{L}(ANSP)$.

Proof. Let $\Gamma = (V, U, G, \mathcal{R}, \mathcal{N}, \beta, In', Out)$ be an ANSPFC with $X_G = \{x_1, x_2, \ldots, x_n\}$, for some $n \geq 2$ such that $In = x_1$ and $Out = x_n$. We construct

the ANSP $\Gamma' = (V, W, G', \mathcal{N}', \beta', In', Out)$, where

$$W = U \cup \underbrace{\{a' \mid a \in U\}}_{U'} \cup \underbrace{\{\overline{a} \mid a \in U\}}_{\overline{U}},$$
$$G' = (X_{G'}, E_{G'}).$$

The nodes in $X_{G'}$ and the substitution processors associated with them are defined as follows:

Node	M	PI	FI	PO	FO	β'
y_i, $1 \leq i \leq n$	$\{a \to a' \mid a \in U\}$	U	U'	U'	$\emptyset$	(w)
y'_i, $1 \leq i \leq n$	$\{a' \to a \mid a \in U\}$	U'	$\emptyset$	U	U'	(w)
$y_{(i,j)}$, $1 \leq i \neq j \leq n$, $\{x_i, x_j\} \in E_G$	$\mathcal{R}(x_i)$	U	$U' \cup \overline{U}$	$P_{\{x_i,x_j\}}$	$F_{\{x_i,x_j\}} \cup U' \cup \overline{U}$	$\beta(\{x_i, x_j\})$
z_j, $1 \leq j \leq n$	$\{a \to \overline{a} \mid a \in U\}$	U	$\emptyset$	$\overline{U}$	U'	(w)
$\overline{z_j}$, $1 \leq j \leq n$	$\{\overline{a} \to a \mid a \in U\}$	$\overline{U}$	U'	U	$\overline{U}$	(w)

Furthermore,

$$E_{G'} = \{\{y_i, y'_i\} \mid 1 \leq i \leq n\} \cup \{\{z_i, \overline{z_i}\} \mid 1 \leq i \leq n\} \cup \{\{\overline{z_i}, y_i\} \mid 1 \leq i \leq n\} \cup$$
$$\{\{y'_i, y_{(i,j)}\} \mid 1 \leq i \neq j \leq n, \{x_i, x_j\} \in E_G\} \cup$$
$$\{\{y_{(i,j)}, z_j\} \mid 1 \leq i \neq j \leq n, \{x_i, x_j\} \in E_G\}.$$

Note that for each edge $\{x_i, x_j\} \in E_G$, both nodes $y_{(i,j)}$ and $y_{(j,i)}$ belong to $X_{G'}$. They differ each other by the set of substitutions: $\mathcal{R}(x_i)$ in $y_{(i,j)}$ and $\mathcal{R}(x_j)$ in $y_{(j,i)}$.

The simulation proposed by this construction can be followed more easily than the previous one. Let $u \in U^*$ be a word in the node x_i of Γ before a substitution step. We assume that u is transformed by one or more substitution rules into u_1 which can pass the filter on the edge $\{x_i, x_j\}$ and a copy originated from u_1 will eventually enter Out.

We now follow the itinerary of u through Γ' which leads to acceptance. From y_i, all words obtained from u, in which an arbitrary occurrence of some symbol was replaced by its primed copy, go to y'_i, where u is recovered. From y'_i, u enters $y_{(i,j)}$, where u_1 is obtained and send out to z_j. From now on, it enters $\overline{z_j}$ and then y_j and the simulation process is resumed with this node. Note the role played by the intermediate nodes y'_i and $\overline{z_j}$, namely to prevent a word going out from $y_{(i,j)}$ to re-enter $y_{(i,j)}$ in the next communication step. A possible direct exchange between the nodes x_i and x_j in Γ is simulated in Γ' by an itinerary following (not in immediate succession) $y_{(i,j)}$, $y_{(j,i)}$, $y_{(i,j)}$, and so on. □

It is easy to note that each substitution/communication step in Γ is simulated by Γ' with a constant number of substitution/communication steps. Therefore, we can state the main result of the paper.

Theorem 1. $\mathcal{L}(ANSPFC) = \mathcal{L}(ANSP)$. *Moreover, each simulation is time complexity preserving.*

4. Further Work

In almost all works devoted to AHNEPs and AHNEPFCs, the underlying graph is a complete graph. Simulations preserving the type of the underlying graph of the simulated network (together with its computational complexity) represent, in our view, a matter of interest. Starting from the observation that every ANSPFC can be immediately transformed into an equivalent ANSPFC with a complete underlying graph (the edges that are to be added are associated with filters which make them useless), we may immediately state that Proposition 1 holds for complete ANSPs and ANSPFCs as well. A direct simulation of ANSPFCs by complete ANSPs seems to be possible but the complexity is increased by a factor equal to the length of the input word. A complexity preserving simulation remains open.

Furthermore, simulations preserving complexity as well as the shape (ring, star, grid, etc.) of the underlying graph remain to be further investigated. Last but not least, the investigation started here for substitution may be continued for the other two operations: insertion and deletion.

References

[1] P. Bottoni, A. Labella, F. Manea, V. Mitrana and J. Sempere, Filter position in networks of evolutionary processors does not matter: A direct proof, in *Proc. 15th International Conf. on DNA Computing and Molecular Programming*, (LNCS 5877, Springer, 2009), pp. 1–11.

[2] E. Csuhaj-Varjú and A. Salomaa, Networks of parallel language processors, in *New Trends in Formal Languages* (LNCS 1218, Springer, 1997), pp. 299-318.

[3] E. Csuhaj-Varjú and V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, *Acta Informatica* **36** (2000) 913-926.

[4] C. Drăgoi, F. Manea and V. Mitrana, Accepting networks of evolutionary processors with filtered connections, *Journal of Universal Computer Science* **13** (2007) 1598-1614.

[5] L. Errico and C. Jesshope, Towards a new architecture for symbolic processing, in *Artificial Intelligence and Information-Control Systems of Robots 94*, (World Scientific, 1994), pp. 31-40.

[6] S.E. Fahlman, G.E. Hinton and T.J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann Machines, in *Proc. of the 3rd. National Conference on Artificial Intelligence* (AAAI Press, Washington DC, 1983), pp. 109-113.

[7] W. Hillis, *The Connection Machine* (MIT Press, Cambridge, 1985).

[8] F. Manea, M. Margenstern, V. Mitrana and M.J. Perez-Jimenez, A new characterization of NP, P, and PSPACE with accepting hybrid networks of evolutionary processors, *Theory of Computing Systems* **46** (2010) 174-192.

[9] F. Manea, C. Martin-Vide and V. Mitrana, Accepting networks of evolutionary word and picture processors: a survey, in *Scientific Applications of Language Methods*, (World Scientific, 2010), pp. 523–560.

[10] G. Păun and L. Sântean, Parallel communicating grammar systems: the regular case, *Annals of University of Bucharest, Ser. Matematica-Informatica* **38** (1989) 55–63.
[11] G. Păun, (2000). Computing with Membranes, *Journal of Computer and System Sciences* **61** (2000) 108-143.