Nat Comput (2016) 15:51–68
DOI 10.1007/s11047-015-9534-1

Computational completeness of complete, star-like, and linear hybrid networks of evolutionary processors with a small number of processors

Artiom Alhazov[1] · Rudolf Freund[2] · Vladimir Rogozhin[3] · Yurii Rogozhin[1]

Published online: 27 January 2016

Abstract A hybrid network of evolutionary processors (HNEP) is a graph where each node is associated with a special rewriting system called an evolutionary processor, an input filter, and an output filter. Each evolutionary processor is given a finite set of one type of point mutations (insertion, deletion or a substitution of a symbol) which can be applied to certain positions in a string. An HNEP rewrites the strings in the nodes and then re-distributes them according to a filter-based communication protocol; the filters are defined by certain variants of random-context conditions. HNEPs can be considered both as languages generating devices (GHNEPs) and language accepting devices (AHNEPs); most previous approaches treated the accepting and generating cases separately. For both cases, in this paper we show that five nodes are sufficient to accept (AHNEPs) or generate (GHNEPs) any recursively enumerable language by showing the more general result that any partial recursive relation can be computed by an HNEP with (at most) five nodes with the underlying graph structure for the communication between the evolutionary processors being the complete or the linear graph with five nodes, whereas with a star-like communication graph we need six nodes. If the final results are defined by only taking the terminal strings out of the designated output node, then for these extended HNEPs we can prove that only four nodes are needed in all cases—for computing any partial recursive relation as well as for generating and accepting any recursively enumerable language—and the underlying communication structure can be a complete or a linear graph, but now even a star-like graph, too.

Keywords Circular Post machines · Communication graph · Computational completeness · Hybrid networks of evolutionary processors

1 Introduction

Networks of evolutionary processors (NEPs) were introduced in Castellanos et al. (2001) as a model of string processing devices distributed over a graph. The nodes of the graph contain the processors that carry out operations of insertion, deletion, and substitution, which reflect basic biological processes known as point mutations. Models based on these operations are of particular interest in formal language theory due to the simplicity of these operations. In NEPs, an evolutionary processor is located at every node of a graph and processes objects, for example (finite sets of) strings. The system functions by rewriting the collections of objects present in the nodes and then redistributing the resulting objects according to a communication protocol defined by the underlying communication

Yurii Rogozhin: deceased on March 10th 2014.

✉ Rudolf Freund
rudi@emcc.at

Artiom Alhazov
artiom@math.md

Vladimir Rogozhin
vladimir.rogojin@abo.fi

[1] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Academiei 5, 2028 Chişinău, Moldova

[2] Faculty of Informatics, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria

[3] Department of Information Technologies, Åbo Akademi University, Joukahainengatan 3-5, 20520 Turku, Finland

structure and a specific filtering mechanism. The language determined by the network is defined as the set of objects which appear in some distinguished node in the course of the computation.

NEPs are models inspired by cell biology, since each processor represents a cell performing point mutations of DNA and controlling its passage out of and into the cell through a specific filtering mechanism. An evolutionary processor corresponds to a cell, the generated string to a DNA strand, and the operations insertion, deletion, and substitution of a symbol correspond to the point mutations. By using an appropriate filtering mechanism, NEPs with a very small number of nodes are very powerful computational devices: already with two nodes, they are as powerful as Turing machines, see Alhazov et al. (2006, 2007, 2009b).

Special variants of these devices are the so-called *hybrid networks of evolutionary processors* (HNEPs), where each language processor performs only one of these operations on a certain position of the strings in that node. Furthermore, the filters are defined by some variants of random-context conditions, i.e., they check the presence and the absence of certain symbols in the strings. Hybrid networks of evolutionary processors can be considered both as language generating and accepting devices: the notion of an HNEP as a language generating device (GHNEP) was introduced in Martín-Vide et al. (2003), and the concept of an accepting HNEP (AHNEP) was defined in Margenstern et al. (2005).

Csuhaj-Varjú et al. (2005) it was shown that GHNEPs with $27 + 3 \cdot card(V)$ nodes are computationally complete, with V being the underlying alphabet. For specific variants of AHNEPs, in Manea et al. (2007) it was shown that 31 nodes are sufficient for recognizing any recursively enumerable language (irrespectively of the size of the alphabet); the result was improved considerably in Manea and Mitrana (2007) where the number of necessary nodes was reduced to 24. In the following, the results were further improved significantly: AHNEPs and GHNEPs of the specific types as defined above were shown to be computationally complete already with 10 nodes in Alhazov et al. (2008a) and only 7 nodes in Alhazov et al. (2008b, 2009a). Then, in Loos et al. (2010) it was already claimed that acceptance can be done with (at most) 6 nodes, as there the authors showed that AHNEP of size 6 are universal by giving a construction of an AHNEP simulating a 2-tag system.

Variants of the underlying graph structure—star networks, ring networks, and grid networks—were considered (Dassow and Manea 2010), with 13 nodes needed for the star-like variant. Truthe (2013), AHNEPs in the form of a chain or a ring (with each processor having at most two neighbours) and ANEPs in the form of a wheel with 56 processors were shown to be computationally complete; these numbers were improved to 29 in Dassow et al. (2015) for the cases of a chain, a ring, or a wheel. As a small technical detail we have to mention that for an input string w the input to the HNEPs we will construct in this paper is $q_1 w$ where q_1 is the initial state, whereas in the papers mentioned before, the input to the system is the pure input string w. On one hand, it is common in the area of molecular computing to supplement the input with some additional symbol(s) on the left and/or on the right; on the other hand, adding one symbol on the left only needs one additional insertion node. Hence, the results in this paper, even taking into account this difference, constitute a considerable improvement in comparison with the results established in Dassow et al. (2015).

In this paper, we improve previous results by showing that *HNEPs are computationally complete with five nodes*, i.e., any recursively enumerable language can already be generated or accepted by an HNEP having at most five nodes, with the underlying communication structure being a complete graph. In fact, we even show that any partial recursive relation can be computed by an HNEP with at most five nodes.[1] As it is known that the families of *HNEPs with two nodes are not computationally complete* (see Alhazov et al. 2009a), the gap for HNEPs between being computationally complete or not now has already become pretty small.

We also investigate different graph structures. Complete graphs as the underlying communication structure first are assumed to have no loops in the nodes, yet we will show that loops do not affect the results. Moreover, all our computational completeness results—computing any partial recursive relation, generating or accepting any recursively enumerable language—also hold true for the underlying structure being a linear graph. With the communication structure being a star-like graph we need one node more, i.e., in this case computational completeness could only be obtained with using six nodes.

Moreover, we will show that we can even save one more node, i.e., we can get computational completeness with only four nodes, if we extract the final results from the designated output node by only taking the terminal strings appearing in this node as the results of a computation. In this case, for extended HNEPs (HNEPs with extracting the terminal strings from the output node), all our computational completeness results can even be obtained with the underlying communication graph between the evolutionary processors being a star-like graph, too.

[1] These results were already claimed in the original paper (Alhazov et al. 2014b) presented at UCNC 2014; in this paper we not only establish a new extended proof of these results, but also consider the variants with terminal extraction and with linear and star-like communication structures.

Finally, all the computational completeness results proved in this paper, for HNEPs and extended HNEPs, hold true for two different variants chosen for the definition of the effect of substitutions or deletions on a string in a given node: the original definition as used in Alhazov et al. (2014b) also yields the underlying string as a result of a computation step whenever at least one rule assigned to the node cannot be applied to this string; the definition we have in mind first when elaborating our main proofs does not yield the underlying string as a result of a computation step in this case, which is a variant already used in the so-called *obligatory* HNEPs (for example, see Alhazov et al. 2011a, 2014a). Yet as we also show that all our constructions still work when using the original definition, our results remain directly comparable with previous results obtained for HNEPs.

2 Definitions

We start by recalling some basic notions of formal language theory. An alphabet is a non-empty finite set. A finite sequence of symbols from an alphabet V is called a *string* over V. The set of all strings over V is denoted by V^*; the *empty string* is denoted by λ; moreover, we define $V^+ = V^* \backslash \{\lambda\}$. The *length* of a string x is denoted by $|x|$, and by $|x|_a$ we denote the number of occurrences of a letter a in a string x. For a string x, $alph(x)$ denotes the smallest alphabet Σ such that $x \in \Sigma^*$. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as Salomaa (1973) and Rozenberg and Salomaa (1997).

Remark 1 In this paper, string rewriting systems as Turing machines, Post systems, etc. are called *computationally complete* if these systems are able to compute any partial recursive relation R on strings over any alphabet U, i.e., $R \subseteq (U^*)^m \times (U^*)^n$, for some $m, n \geq 0$. As input and output alphabet for these systems we assume to take $T = U \cup \{\$\}$, where \$ is a special delimiter separating the components of an input vector $(w_1, \ldots, w_m)$ and an output vector $(v_1, \ldots, v_n)$, $w_i \in U^*$, $1 \leq i \leq m$, $v_j \in U^*$, $1 \leq j \leq n$. In that sense, any relation $R \subseteq (U^*)^m \times (U^*)^n$ can also be considered as a special relation $R' \subseteq T^* \times T^*$.

Remark 2 Computational completeness in the usual sense with respect to acceptance and generation directly follows from this general kind of computational completeness, as any recursively enumerable language L can be viewed as partial recursive relation $L \times \{\lambda\}$ (acceptance) and $\{\lambda\} \times L$ (generation); λ can be replaced by any arbitrary string. For the accepting case, we can even take any relation R whose first component is L, which corresponds to taking $\{u \in U^* \mid uRv,\ v \in U^*\}$ as the accepted language and also is the usual way how acceptance is defined in the previous papers on networks of evolutionary processors. The results proved in this paper, establishing acceptance even when restricting the second component, obviously also hold true for the case when taking the more relaxed original definitions.

2.1 Hybrid networks of evolutionary processors

For introducing the notions concerning evolutionary processors and hybrid networks, we mainly follow (Csuhaj-Varjú et al. 2005). These language processors use so-called *evolutionary operations*, simple rewriting rules which abstract local gene mutations.

Definition 1 For an alphabet V, let $a \to b$ be a rewriting rule with $a, b \in V \cup \{\lambda\}$, and $ab \neq \lambda$; we call such a rule a *substitution rule* if both a and b are different from λ; such a rule is called a *deletion rule* if $a \neq \lambda$ and $b = \lambda$, and it is called an *insertion rule* if $a = \lambda$ and $b \neq \lambda$. The set of all substitution rules, deletion rules, and insertion rules over an alphabet V is denoted by Sub_V, Del_V, and Ins_V, respectively.

Given such rules $\pi \equiv a \to b \in Sub_V$, $\rho \equiv a \to \lambda \in Del_V$, and $\sigma \equiv \lambda \to a \in Ins_V$ as well as a string $w \in V^*$, we define the following *actions* of π, ρ, and σ on w:

$$\pi^*(w) = \{ubv \mid w = uav,\ u, v \in V^*\},$$
$$\pi^l(w) = \{bv \mid w = av\},$$
$$\pi^r(w) = \{ub \mid w = ua\},$$
$$\rho^*(w) = \{uv \mid w = uav,\ u, v \in V^*\},$$
$$\rho^l(w) = \{v \mid w = av\},$$
$$\rho^r(w) = \{u \mid w = ua\},$$
$$\sigma^*(w) = \{uav \mid w = uv,\ u, v \in V^*\},$$
$$\sigma^l(w) = \{aw\},$$
$$\sigma^r(w) = \{wa\}.$$

The symbol $\alpha \in \{*, l, r\}$ denotes the mode of applying a substitution, insertion or deletion rule to a string, namely, at any position ($\alpha = *$), on the left-hand end ($\alpha = l$), or on the right-hand end ($\alpha = r$) of the string, respectively.

Remark 3 The definitions given above coincide with the definitions of the evolutionary operations given for the so-called *obligatory HNEPS*, for example, see Alhazov et al. (2011a) and Alhazov et al. (2014a). In the original definitions given in the literature, even in Alhazov et al. (2014b), a slightly modified version of these definitions is used: if any of these sets results to be empty, which may happen for $\pi^l(w)$, $\pi^r(w)$, $\pi^*(w)$, $\rho^l(w)$, $\rho^r(w)$, $\rho^*(w)$, i.e., if no strings v or u or u and v satisfy the indicated condition, because the symbol a to be substituted or deleted is not present in w or it is not found at the expected position, the resulting set for

the corresponding operation is defined to contain its argument, i.e., it equals $\{w\}$.

The proofs for all the theorems established in this paper will first be only argued for the variant given in Definition 1, yet in order to make our results comparable with previous results, we then always will add the arguments needed to show that they also hold if the original definition for the results of the actions π, ρ, σ on w is used.

For any rule β, $\beta \in \{\pi, \rho, \sigma\}$, any mode $\alpha \in \{*, l, r\}$, and any $L \subseteq V^*$, we define the α-*action* of β on L by $\beta^\alpha(L) = \bigcup_{w \in L} \beta^\alpha(w)$. For a given finite set of rules M , we define the α-action of M on a string w and on a language L by $M^\alpha(w) = \bigcup_{\beta \in M} \beta^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

We notice that, as in previous papers on HNEPs, substitutions in the following will only be used at arbitrary positions, i.e., with $\alpha = *$.

For two disjoint finite subsets P and F of an alphabet V and any string w over V, the two predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ are defined as follows:

$$\varphi^{(1)}(w; P, F) \equiv (P \subseteq alph(w)) \wedge (F \cap alph(w) = \emptyset),$$
$$\varphi^{(2)}(w; P, F) \equiv ((P = \emptyset) \vee (alph(w) \cap P \neq \emptyset)) \wedge (F \cap alph(w) = \emptyset).$$

The idea of these predicates is based on *random-context conditions* defined by sets P (*permitting contexts*) and F (*forbidden contexts*). Moreover, for any $L \subseteq V^*$, we define

$$\varphi^i(L; P, F) = \{w \in L \mid \varphi^i(w; P, F)\}, i \in \{1, 2\}.$$

An evolutionary processor consists of a set of evolutionary operations (substitutions, insertions, deletions) and a filtering mechanism, i.e., we define an *evolutionary processor over* V as a 5-tuple (M, PI, FI, PO, FO) where

- either $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$, i.e., the set M represents the set of evolutionary rules of the processor (notice that every processor is dedicated to only one type of the evolutionary operations);
- $PI, FI \subseteq V$ are the *input* permitting and forbidden contexts of the processor and $PO, FO \subseteq V$ are the *output* permitting and forbidden contexts of the processor.

The set of evolutionary processors over V is denoted by EP_V.

We now are able to formally define the main computational models considered in this paper, i.e., *obligatory hybrid networks of evolutionary processors (OHNEPs)* and *hybrid networks of evolutionary processors (HNEPs)*. In order to keep notations concise, we will use the bracket notation [O]HNEP to indicate that the definitions or the results stated in a theorem or corollary are valid no matter whether in the nodes of the HNEP we obtain the results of applying the rules to a string according to Definition 1 (which is indicated by writing OHNEP) or according to Remark 3 (which is indicated by writing HNEP as usual).

Definition 2 An *obligatory hybrid network of evolutionary processors* (an *OHNEP* for short) or a *hybrid network of evolutionary processors* (an *HNEP* for short) over V is a construct

$\Gamma = (V, T, H, \mathcal{N}, C_{\text{init}}, \alpha, \beta, C_{\text{input}}, i_0)$where

- V is the alphabet of the network;
- T is the input/output alphabet, $T \subseteq V$;
- $H = (X_H, E_H)$ is an undirected graph with the set of vertices or nodes X_H and the set of (undirected) edges E_H; H is called the underlying *communication graph* of the network;
- $\mathcal{N} : X_H \longrightarrow EP_V$ is a mapping which with each node $x \in X_H$ associates the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$;
- $C_{\text{init}} : X_H \to 2^{V^*}$ is a mapping which identifies the initial configuration of the network; it associates a finite set of words with each node of graph H;
- $\alpha : X_H \longrightarrow \{*, l, r\}$; $\alpha(x)$ defines the action mode of the rules performed on the strings occurring in node x;
- $\beta : X_H \longrightarrow \{(1), (2)\}$ defines the type of the input and output filters of a node; for every node x, $x \in X_H$, and for any language L we define $\mu_x(L) = \varphi^{\beta(x)}(L; PI_x, FI_x)$ and $\tau_x(L) = \varphi^{\beta(x)}(L; PO_x, FO_x)$, i.e., $\mu_x(L)$ and $\tau_x(L)$ are the sets of strings of L that can pass the input and the output filter of x, respectively;
- $C_{\text{input}} : X_H \longrightarrow 2^{V^*}$ defines a finite set of "initial strings for the input": for any $(x, w_0(x)) \in C_{\text{input}}$, the input string is concatenated to $w_0(x)$ and added to node x of the graph H, as described below;
- $i_0 \in X_H$ is the output node of Γ.

The *size* of Γ is defined to be the number of nodes in X_H. An [O]HNEP is said to be a *complete [O]HNEP* if its underlying communication graph is a complete graph; it is called a *linear [O]HNEP* if its underlying communication graph is linear, and a *star-like [O]HNEP* if its underlying communication graph is a star-like graph, i.e., only the central node is connected with each of the other nodes.

Looking at [O]HNEPs as devices computing partial recursive relations R on an alphabet U, i.e., $R \subseteq (U^*)^m \times (U^*)^n$, for some $m, n \geq 0$, we take $T = U \cup \{\$\}$, where \$ is a special delimiter separating the components of an input vector $(w_1, \ldots, w_m)$ and an output vector $(v_1, \ldots, v_n)$, $w_i \in U^*$, $1 \leq i \leq m$, $v_j \in U^*$, $1 \leq j \leq n$.

A configuration of an [O]HNEP Γ, as defined above, is a mapping $C : X_H \longrightarrow 2^{V^*}$ which associates a set of strings over V with each node x of the graph. A component $C(x)$ of a configuration C is the set of strings that can be found in the node x of this configuration, hence, a configuration can be considered as a list of the sets of strings which are present in the nodes of the network at a given moment. For a given input vector $(w_1, \ldots, w_m)$, $w_i \in U^*$, $1 \leq i \leq m$, the initial configuration C_0 of the [O]HNEP is obtained by adding to C_{init} the strings $w_0(x)w_1\$\ldots\$w_m$ in each node x, for any $(x, w_0(x)) \in C_{\text{input}}$, i.e., we define $C_0(x) = C_{init}(x) \cup \{w_0(x)w_1\$\ldots\$w_m \mid (x, w_0(x)) \in C_{\text{input}}\}$.

A configuration can change either by an *evolutionary step* or by a *communication step*.

When the configuration changes by an *evolutionary step*, then each component $C(x)$ of the configuration C is altered in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules, $\alpha(x)$, according to Definition 1 for an OHNEP and according to Remark 3 for an HNEP. Formally, the configuration C' is obtained in one evolutionary step from the configuration C, written as $C \Longrightarrow C'$, if and only if $C'(x) = M_x^{\alpha(x)}(C(x))$ for all $x \in X_H$. We observe that $C'(x) = \emptyset$ if $M_x = \emptyset$ for both variants of getting the result of applying the rules in M_x in mode $\alpha(x)$.

When the configuration changes by a *communication step*, then each language processor $\mathcal{N}(x)$, where $x \in X_H$, sends a copy of each of its strings to every node y the node x is connected with, provided that this string is able to pass the output filter of x, and receives all the strings which are sent by the processor of any node y connected with x provided that these strings are able to pass the output filters of y and the input filter of x. Those strings which are not able to pass its output filter, remain in the node x. Formally, we say that configuration C' is obtained in one communication step from configuration C, written as $C \vdash C'$, if and only if

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

holds for all $x \in X_H$.

A *computation* in Γ is a sequence of configurations C_0, C_1, $C_2, \ldots$ where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geq 0$. Note that each configuration C_{i+1} is uniquely determined by the configuration C_i, $i \geq 0$. The *result of a computation* in Γ for an input vector $(w_1, \ldots, w_m)$, $w_i \in U^*$, $1 \leq i \leq m$, i.e., for the initial configuration C_0 with $C_0(x) = C_{init}(x) \cup \{w_0(x)w_1\$\ldots\$w_m \mid (x, w_0(x)) \in C_{\text{input}}\}$ for $x \in X_H$, is the set of all strings of the form $v_1\$\ldots\$v_n$, $v_j \in U^*$, $1 \leq j \leq n$, arriving in the output node i_0 at any computation step of Γ, i.e.,

$$L(\Gamma)((w_1, \ldots, w_m)) = \{(v_1, \ldots, v_n) \mid v_j \in U^*, 1 \leq j \leq n, v_1\$\ldots\$v_n \in C_s(i_0), s \geq 0)\}.$$

Remark 4 Consider any input $w_{in} = w_1\$\ldots\$w_m$. We first note that, since different strings do not influence each other, the strings in C_{init} do not affect the evolution of the strings in C_{input} concatenated with w_{in} and vice-versa. The results thus are the union of the strings obtained from C_{init}, which do not depend on the input, and the strings obtained from w_{in}, which do not depend on the strings in C_{init}.

Therefore, for the results elaborated in this paper we may always assume C_{init} to be empty, and even exclude it from the tuple defining the network. Moreover, we may also assume that C_{input} only consists of one string in one node, i.e., $C_{\text{input}} = \{(x_0, w_0)\}$.

As special cases, [O]HNEPs can be considered either as language generating devices (generating hybrid networks of evolutionary processors or *G[O]HNEPs*) or language accepting devices (accepting hybrid networks of evolutionary processors or *A[O]HNEPs*). In the case of G[O]HNEPs, the relation to be computed is $\{\lambda\} \times L$, i.e., the initial configuration always equals $\{(x_0, w_0)\}$; the generated language is the set of all strings which appear in the output node at some step of the computation, i.e., the language generated by a generating hybrid network of evolutionary processors Γ is $L_{gen}(\Gamma) = \bigcup_{s \geq 0} C_s(i_0)$. In the case of A[O]HNEPs, the relation to be computed is $L \times \{\lambda\}$, i.e., starting from the initial configuration $\{(x_0, w_0w_1)\}$, we accept the input string w_1 if and only if at some moment of the computation the empty string appears in the output node (and never any other string is computed), i.e., the language accepted by Γ is defined by $L_{acc}(\Gamma) = \{w_1 \in V^* \mid \exists s \geq 0 (C_s(i_0) = \{\lambda\})\}$.

2.2 Post systems and circular post machines

The left and right insertion, deletion, and substitution rules defined in the preceding subsection are special cases of string rewriting rules only working at the ends of a string; they can be seen as restricted variants of Post rewriting rules as already introduced by Post (1943): for a *simple Post rewriting rule* $\Pi_s \equiv u\$x \rightarrow y\$v$, where $u, v, x, y \in V^*$, for an alphabet V, we define

$$\pi_s(w) = \{yzv \mid w = uzx,\ z \in V^*\}.$$

A *normal Post rewriting rule* $\pi_n \equiv \$x \rightarrow y\$$ is a special case of a simple Post rewriting rule $u\$x \rightarrow y\$v$ with $u = v = \lambda$ (we also assume $xy \neq \lambda$); this normal Post rewriting rule $\$x \rightarrow y\$$ is the mirror version of the normal form rules $u\$ \rightarrow \$v$ as originally considered in Post (1943) for Post canonical systems; yet this variant has already been used

several times for proving specific results in the area of P systems, e.g., see Freund et al. (2014). A *Post system of type* X is a construct (V, T, A, P) where V is a (finite) set of *symbols*, $T \subseteq V$ is a set of *terminal symbols*, $A \in V^*$ is the *axiom*, and P is a finite set of *Post rewriting rules* of type X; for example, X can mean simple or normal Post rewriting rules. In both cases it is folklore that these Post systems of type X are computationally complete.

The basic idea of the computational completeness proofs for Post systems is the "rotate-and-simulate"-technique, i.e., the string is rotated until the string x to be rewritten appears on the right-hand side, where it can be erased and replaced by the string y on the left-hand side, which in total can be accomplished by the rule $\$x \to y\$$. By rules of the form $\$a \to a\$$ for each symbol a the string can be rotated. In order to indicate the beginning of the string in all its rotated versions, a special symbol B (different from all others) is used; B is to be erased at the end of a successful computation.

Circular Post machines are machine-like variants of Post systems using specific variants of simple Post rewriting rules; several variants named *CPMi*, $0 \leq i \leq 4$, were introduced (Kudlek and Rogozhin 2001b) and further studied in Kudlek and Rogozhin (2001a), Alhazov et al. (2002), and the variants of *CPM5* we use in this paper were investigated in Alhazov et al. (2011b). It was stated in Alhazov et al. (2011b) that CPM5 is an interesting model that deserves further attention; in the present paper we confirm that this is the case by constructing HNEPs simulating CPM5s.

Definition 3 A (non-deterministic) *CPM5* is a construct

$$M = (\Sigma, T, Q, q_1, q_0, R),$$

where Σ is a finite alphabet, $T \subseteq \Sigma$ is the set of terminal symbols, Q is the set of states, $q_1 \in Q$ is the initial state, $q_0 \in Q$ is the only terminal state, and R is a set of simple Post rewriting rules of the following types (we use the notation $Q' = Q \backslash \{q_0\}$):

- $px\$ \to q\$$ (*deletion rule*) with $p \in Q'$, $q \in Q$, $x \in \Sigma$; we also write $px \to q$ and, for any $w \in \Sigma^*$, the corresponding computation step is $pxw \overset{px \to q}{\longrightarrow} qw$;
- $p\$ \to q\$y$ (*insertion rule*) with $p \in Q'$, $q \in Q$, $y \in \Sigma$; we also write $p \to yq$ and, for any $w \in \Sigma^*$, the corresponding computation step is $pw \overset{p \to yq}{\longrightarrow} qwy$.

The CPM5 is called *deterministic* if for any two deletion rules $px \to q_1$ and $px \to q_2$ we have $q_1 = q_2$ and for any two insertion rules $p \to q_1 y_1$ and $p \to q_2 y_2$ we have $q_1 y_1 = q_2 y_2$.

The name circular Post machine comes up from the idea of interpreting the machines to work on circular strings where both deletion and insertion rules have local effects, as for circular strings the effect of the insertion rule $p\$ \to q\$y$ is the same as the effect of $p \to yq$ directly applied to a circular string, which also justifies writing $p\$ \to q\$y$ as $p \to yq$.

For a given input string w, $w \in T^*$, the CPM5 M starts with $q_1 w$ and applies rules from R until it eventually reaches a configuration $q_0 v$ for some $v \in T^*$; in this case we say that (w, v) is in the relation computed by M.

Definition 4 A CPM5 $M = (\Sigma, T, Q, q_1, q_0, R)$ is said to be in *normal form* if

- $Q \backslash \{q_0\} = Q_1 \cup Q_2$ where $Q_1 \cap Q_2 = \emptyset$;
- for every $p \in Q_1$ and every $x \in \Sigma$, there is exactly one instruction of the form $px \to q$, i.e., Q_1 is the set of states for deletion rules;
- for every insertion rule $p \to yq$ we have $p \in Q_2$, i.e., Q_2 is the set of states for insertion rules, and moreover, if $p \to y_1 q_1$ and $p \to y_2 q_2$ are two different rules in R, then $y_1 = y_2$.

Alhazov et al. (2011b), a CPM5 in normal form even obeying the constraint that for each $p \in Q_2$ there are at most two different rules $p \to yq_1$ and $p \to yq_2$ in R (and, again, M is called deterministic if $q_1 = q_2$) were shown to be computationally complete. The following result can be derived from the theorems proved in Alhazov et al. (2011b):

Theorem 1 (see Alhazov et al. 2011b) *CPM5s, even in normal form, are computationally complete.*

3 Computational completeness of [O]HNEPs with five nodes

In this section we prove our main result showing that complete [O]HNEPs with five nodes are sufficient to obtain computational completeness. Yet in order to get a more efficient description of the derivations possible in an [O]HNEP, we first need the following observations.

Remark 5 HNEPs originally are defined as (deterministic) distributed string-processing devices where the evolution rules are simultaneously applied in all possible ways to (different copies of) all possible strings. However, as already mentioned in Remark 4, there is no interaction between the strings, hence, it is sufficient to consider any possible behavior of an [O]HNEP as a *non-deterministic* distributed device processing one string w_0 in a cell x_0 to one other string w_1 in a cell x_1. Therefore, in the following, without loss of generality we will consider a configuration as (`region`, `string`), i.e., one string in one node, for any possible evolution.

Remark 6 In the proofs elaborated in the following, we first will have in mind the variant given by Definition 1, i.e., for an OHNEP, but whenever necessary, we will also argue why the construction given in the proof works for the original variant described in Remark 3, i.e., for an HNEP, too.

Theorem 2 *Any (non-deterministic) CPM5 M in normal form can be simulated by a complete [O]HNEP Γ of size* 5.

Proof Let $M = (\Sigma, T, Q, q_1, q_0, R)$ be a (non-deterministic) CPM5 in the normal form as defined in Definition 4, with symbols $\Sigma = \{a_j \mid 1 \le j \le m\}$ and states $Q = \{q_i \mid 0 \le i \le n\}$, where q_1 is the initial state and the only terminal state is $q_0 \in Q$. We now construct a complete OHNEP $\Gamma = (V, T, H, \mathcal{N}, \alpha, \beta, C_0^0, 5)$ of size 5 which simulates the given CPM5 M. The following sets are used in its description:

$$
\begin{aligned}
&J_\Sigma = \{1 \ldots m\},\\
&J_K = \{0, 1 \ldots n\},\\
&S = \{s_i \mid i \in J_K\},\\
&A = \{a_l \mid l \in J_\Sigma \cup \{0\}\} \ (= \Sigma \cup \{a_0\}),\\
&A' = \{a'_l \mid l \in J_\Sigma \cup \{0\}\},\\
&A'' = \{a''_l \mid l \in J_\Sigma\},\\
&\bar{A} = \{\bar{a}_{s,t} \mid s \in J_K,\ t \in J_\Sigma\},\\
&\hat{A} = \{\hat{a}_{s,t} \mid s \in J_K,\ t \in J_\Sigma x\},\\
&\tilde{Q} = \{\tilde{q} \mid q \in Q \backslash \{q_0\}\},\\
&\bar{Q} = \{\bar{q} \mid q \in Q\},\\
&\hat{Q} = \{\hat{q} \mid q \in Q\},\\
&Q' = \{q'_{s,t} \mid q_s \in Q,\ t \in J_\Sigma\},\\
&Q'' = \{q''_{s,t} \mid q_s \in Q,\ t \in J_\Sigma\},\\
&V = S \cup A \cup A' \cup A'' \cup \bar{A} \cup \hat{A}\\
&\qquad \cup Q \cup \tilde{Q} \cup \bar{Q} \cup \hat{Q} \cup Q' \cup Q'' \cup \{\varepsilon\}.
\end{aligned}
$$

We assume H to be the complete graph with 5 nodes (without loops), i.e., we take Γ to be a complete OHNEP. Moreover, we take $C_0^0 = \{(1, q_1)\}$, i.e., for the input string w_1, the initial configuration is $\{(1, q_1 w_1)\}$; the output node of Γ for collecting the results of a computation is node 5.

Moreover, we take $\beta(i) = 2$ for all $1 \le i \le 5$, which means that a string w can pass the filter (P, F) if and only if for a non-empty set P of permitting contexts at least one symbol of w is contained in P as well as no symbol of w is in the set F of forbidden contexts. Finally, we take $\alpha(1) = \alpha(2) = \alpha(5) = *$, as the two nodes 1 and 2 as well as the output node 5 perform substitutions, as well as $\alpha(3) = r$ and $\alpha(4) = l$. The evolutionary processors $\mathcal{N}(i) = (M_i, PI_i, FI_i, PO_i, FO_i)$, $1 \le i \le 5$, are defined as follows (for the different rules, we use labels for identifying them later in the explanations given below).

The simulation of each rule of M starts in node 1: if the current state q is an insertion or a deletion node, i.e., $q \in Q_1 \cup Q_2$, the rule **1.1** has to be applied, and in the succeeding communication step, if the state symbol q has been from Q_1, the string has to go to node 2, where the symbol to be deleted is chosen, or else, if the state symbol has been from Q_2, the string has to go to node 3, where the symbol a_0 is inserted on the right end of the string. Finally, for the current state q being the final state q_0, we have to apply rule **1.2** thus obtaining the symbol ε, which directs the string to node 4, where the deletion of ε finally will yield a terminal string in node 5.

$$
\begin{aligned}
M_1 = {} & \{\mathbf{1.1} : q \to \tilde{q} \mid q \in Q_1 \cup Q_2\}\\
& \cup \{\mathbf{1.2} : q_0 \to \varepsilon\}\\
& \cup \left\{\mathbf{1.3} : \tilde{q}_i \to q'_{k,j} \mid q_i \to a_j q_k \in R\right\}\\
& \cup \left\{\mathbf{1.4} : q''_{s,t} \to q'_{s,t} \mid q_s \in Q,\ t \in J_\Sigma\right\}\\
& \cup \left\{\mathbf{1.5} : a_0 \to a'_0\right\}\\
& \cup \left\{\mathbf{1.6} : a''_l \to a'_l \mid l \in J_\Sigma\right\}\\
& \cup \left\{\mathbf{1.7} : a''_l \to a_l \mid l \in J_\Sigma\right\}\\
& \cup \left\{\mathbf{1.8} : \hat{a}_{s,t} \to \bar{a}_{s,t} \mid s \in J_K,\ t \in J_\Sigma\right\}\\
& \cup \{\mathbf{1.9} : \hat{q}_l \to \bar{q}_l \mid l \in J_K \backslash \{0\}\}\\
& \cup \left\{\mathbf{1.10} : \hat{a}_{i,j} \to s_k \mid q_i a_j \to q_k \in R,\ \ i \in J_K \backslash \{0\}\right\}\\
& \cup \{\mathbf{1.11} : \hat{q}_0 \to \varepsilon\},\\
& \cup \{\mathbf{1.12} : s_k \to q_k \mid k \in J_K\},\\
PI_1 = {} & S \cup Q \cup \tilde{Q}_2 \cup \hat{Q} \cup Q'',\\
FI_1 = {} & A' \cup \bar{A} \cup \tilde{Q}_1 \cup \bar{Q} \cup Q' \cup \{\varepsilon\},\\
PO_1 = {} & \tilde{Q} \cup \bar{Q} \cup Q' \cup \{\varepsilon\},\\
FO_1 = {} & Q \cup \hat{Q} \cup Q'' \cup A'' \cup \hat{A} \cup \{a_0\}.
\end{aligned}
$$

The remaining rules in M_1 and the rules in M_2 mainly are used to rename the marked symbols—usually a version of a state and a version of a symbol from A—in the string and to decrement or increment the indices in a synchronized way.

Together with the rules in node 1, the rules in the second node change the labels of the marked pair (*state*, *symbol*) in a synchronized way. The second task of node 2 is to pick one symbol a_j and replace it by $\hat{a}_{0,j}$. Only if the position is correct, i.e., if the chosen symbol a_j is the first one after the state symbol q_i, then the simulation of the deletion rule will be successful at the end.

$$
\begin{aligned}
M_2 &= \left\{\mathbf{2.1} : a'_{l-1} \to a''_l \mid l \in J_\Sigma\right\} \\
&\quad \cup \left\{\mathbf{2.2} : q'_{s,t} \to q''_{s,t-1} \mid q_s \in Q,\ t \in J_\Sigma \backslash \{1\}\right\} \\
&\quad \cup \left\{\mathbf{2.3} : q'_{k,1} \to q_k \mid q_k \in Q\right\} \\
&\quad \cup \left\{\mathbf{2.4} : \bar{a}_{k,t} \to \hat{a}_{k+1,t} \mid k \in J_K \backslash \{n\}, t \in J_\Sigma\right\} \\
&\quad \cup \{\mathbf{2.5} : \bar{q}_l \to \hat{q}_{l-1} \mid l \in J_K \backslash \{0\}\}, \\
&\quad = \left\{\mathbf{2.6} : \tilde{q}_i \to \hat{q}_i \mid q_i a_j \to q_k \in R\right\}, \\
&\quad \cup \left\{\mathbf{2.7} : a_j \to \hat{a}_{0,j} \mid j \in J_\Sigma\right\}, \\
PI_2 &= A' \cup \bar{A} \cup \tilde{Q}_1, \\
FI_2 &= S \cup A'' \cup \hat{A} \cup Q \cup \tilde{Q}_2 \cup \hat{Q} \cup Q'' \cup \{\varepsilon\}, \\
PO_2 &= A'' \cup \hat{A}, \\
FO_2 &= \emptyset.
\end{aligned}
$$

As $FO_2 = \emptyset$, any string has to leave node 2 immediately in the next communication step as soon as a symbol from PO_2 has been generated.

In node 3, the insertion of any symbol from Σ starts with inserting a_0 on the right end of a string that is allowed to enter this node, which means that besides symbols from Σ only symbols (in fact one symbol) from $\tilde{Q}_2$ are allowed. The string with this new symbol a_0 on its right end has to leave node 3 immediately in the next communication step. From a_0, the desired symbol for the given state is obtained by the synchronization procedure carried out by nodes 1 and 2. We observe that we could even take $PO_3 = \emptyset$ as the rule $\mathbf{3.1} : \lambda \to a_0$ is always applicable and $FO_4 = \emptyset$, hence, every string allowed to enter node 3 has to leave it immediately in the succeeding communication step.

$$
\begin{aligned}
M_3 &= \{\mathbf{3.1} : \lambda \to a_0\}, \\
PI_3 &= \tilde{Q}_2, \\
FI_3 &= V \backslash (\tilde{Q}_2 \cup \Sigma), \\
PO_3 &= \{a_0\}, \\
FO_3 &= \emptyset.
\end{aligned}
$$

The fourth node carries out the deletion process, and there is only one symbol, i.e., ε, to be deleted. Only strings containing this special symbol ε can enter node 4, yet according to the working mode of this processor this symbol can only be erased if it appears as the left-most symbol of the string, which means that all those string which can enter this node yet not having ε on its left end cannot be processed and therefore get stuck in node 4: with respect to the original definition of the effect of a deletion operation (see Remark 3), this means that the string remains in the node, but cannot leave it as ε is prohibited by FO_4; in the case of OHNEPs (see Definition 1) this means that during the next evolutionary step the string simply vanishes. In any case, such a string can never go to another node any more.

Moreover, we notice that strings entering node 4, besides this special symbol ε may only contain symbols from Σ and symbols (in fact it will be only one symbol) from S.

$$
\begin{aligned}
M_4 &= \{\mathbf{4.1} : \varepsilon \to \lambda\}, \\
PI_4 &= \{\varepsilon\}, \\
FI_4 &= V \backslash (S \cup \Sigma \cup \{\varepsilon\}), \\
PO_4 &= \emptyset, \\
FO_4 &= \{\varepsilon\}.
\end{aligned}
$$

In the output node 5, although formally defined as a processor carrying out substitutions, in fact no operations take place any more; this node only serves for taking in the terminal strings by using the input filter of forbidding contexts $FI_5 = V \backslash T$. As $M_5 = \emptyset$, during the next evolutionary step these terminal strings will vanish. This does not matter, because a string is taken as a result if it (at least) once appears in the output node during a computation of an [O]HNEP.

$$
\begin{aligned}
M_5 &= \emptyset, \\
PI_5 &= \emptyset, \\
FI_5 &= V \backslash T, \\
PO_5 &= \emptyset, \\
FO_5 &= \emptyset.
\end{aligned}
$$

After having completed the definition of the whole system, we now explain how the [O]HNEP Γ simulates the rules of the CPM5 M.

Let $q_1 w_1$, $w_1 \in T^*$, be the initial configuration of CPM5 M and $q_0 w_0$ the final configuration of M, i.e., M starts with $q_1 w_1$ and ends with $q_0 w_0$, $w_1 \in T^*$, $w_0 \in T^*$. Then the [O]HNEP Γ starts the simulation with the initial configuration $C_0 = \{(1, q_1 w_1)\}$, and we show that the simulation in Γ only yields the string w_0 in the output node 5 of Γ, and moreover, if M never stops when starting with $q_1 w_1$, then Γ generates nothing in the output node.

Without loss of generality, we assume that the CPM5 M starts with a rule of type $q_1 \to a_j q_k$ with $q_k \in Q_2$ and halts after applying a rule of type $q_i a_j \to q_0$, and thus any sequence of consecutive rules $q_{i_1} \to a_{j_1} q_{i_2}$, $q_{i_2} \to a_{j_2} q_{i_3}$, $\cdots$, $q_{i_t} \to a_{j_t} q_{k_{t+1}}$ with $q_{k_{t+1}} \in Q_2$ from any halting computation ends with a rule $q_{k_{t+1}} a_s \to q_{k_{t+2}}$ with $q_{k_{t+2}} \in Q_1$.

The first two of the following three cases describe how an insertion rule and how a deletion rule of the CPM5 M can be simulated by the OHNEP Γ; the third case shows how finally the terminal string w_0 is obtained in the output node 5 from a string of the form $q_0 w_0$ in node 1.

Case 1 Consider an insertion rule $q_i \to a_j q_k \in R$, $q_i \in Q_2$, $q_k \in Q \backslash \{q_0\}$, $a_j \in \Sigma$, and let $q_i w \Longrightarrow q_k w a_j$ be the

corresponding computation step in M, i.e., rule $q_i \to a_j q_k$ is applied to the string $q_i w$ yielding $q_k w a_j$. Starting with the string $q_i w$ being situated in node 1 of Γ, we now describe the desired correct evolution of this string $q_i w$ in Γ and also argue why all other variants of derivations lead to a situation where the string either cannot leave its current node any more or gets lost during a communication step by having to leave its current node but cannot enter any other node.

We start with applying rule **1.1** in node 1. Due to the output filters of node 1, the resulting string has to leave node 1 in the succeeding communication step. Due to the input filters of the other nodes, this string can only go to node 3, where $PI_3 = \tilde{Q}_2$. In node 3, the single insertion rule $\mathbf{3.1} : \lambda \to a_0$ inserts the symbol a_0 on the right-hand side of the string. In the succeeding communication step, the resulting string $\tilde{q}_i w a_0$ has to leave node 3, as the condition for the output filter $PO_3 = \{a_0\}$ is fulfilled, and due to the input filters of the other nodes, this string can only go back to node 1. In node 1, the computation proceeds with the application of the rule $\mathbf{1.5} : a_0 \to a'_0$ as well as with the application of a rule $\mathbf{1.3} : \tilde{q}_i \to q'_{k,j}$ such that $q_i \to a_j q_k \in R$. The resulting string has to leave node 1 and can only be communicated to node 2.

In total, so far we have got the derivation

$$(1, q_i w) \overset{\mathbf{1.1}}{\Longrightarrow} (1, \tilde{q}_i w) \vdash (3, \tilde{q}_i w) \overset{\mathbf{3.1}}{\Longrightarrow} (3, \tilde{q}_i w a_0) \vdash$$
$$(1, \tilde{q}_i w a_0) \overset{<\mathbf{1.3},\mathbf{1.5}>}{\Longrightarrow} (1, q'_{k,j} w a'_0) \vdash (2, q'_{k,j} w a'_0).$$

In the description of the derivation above, if the application of several rules in a specific sequence is necessary in a node before a string should leave this node, we do not indicate the intermediate communication step leaving the string in the node, i.e., instead of writing

$$(1, \tilde{q}_i w a_0) \overset{\mathbf{1.3}}{\Longrightarrow} (1, q'_{k,j} w a_0) \vdash (1, q'_{k,j} w a_0) \overset{\mathbf{1.5}}{\Longrightarrow}$$
$$(1, q'_{k,j} w a'_0)$$

we simply have written

$$(1, \tilde{q}_i w a_0) \overset{<\mathbf{1.3},\mathbf{1.5}>}{\Longrightarrow} (1, q'_{k,j} w a'_0).$$

If rule **1.5** is applied first, the resulting string $\tilde{q}_i w a'_0$ would have to leave node 1 immediately, but the input filters of the other nodes would not allow the string to enter, hence this string would get lost.

The main idea of the succeeding synchronization process by toggling between nodes 2 and 1 is to guarantee that by rule **1.7** the correct symbol a_j is released together with having obtained q_k from $q'_{k,1}$ by rule **2.3**.

As long as $t > 1$, in node 2 exactly the two rules $\mathbf{2.2} : q'_{s,t} \to q''_{s,t-1}$ and $\mathbf{2.1} : a'_{l-1} \to a'_l$ (in the first step for $t = j$ and $l = 0$) have to be applied in exactly this order, and afterwards the resulting string is communicated to node 1.

$$(2, q'_{k,j} w a'_0) \overset{<\mathbf{2.2},\mathbf{2.1}>}{\Longrightarrow} (2, q''_{k,j-1} w a''_1) \vdash (1, q''_{k,j-1} w a''_1).$$

If rule **2.1** is applied first, the resulting string $q'_{k,j} w a''_1$ would have to leave node 2 immediately, but the input filters of the other nodes would not allow the string to enter, hence, this string would get lost; especially node 1 cannot take the string because it does not contain a symbol from PI_1.

The only rule which could also be applied is rule $\mathbf{2.7} : a_m \to \hat{a}_{0,m}$. The application of this rule immediately forces the resulting string $q'_{k,j} w' \hat{a}_{0,m} w'' a'_0$ to leave node 2, but then this string gets lost as it cannot enter any other node, especially node 1 cannot take the string because it still contains a symbol from A' which is not allowed due to FI_1. If rule **2.7** is applied after the application of **2.2**, the resulting string $q''_{k,j-1} w' \hat{a}_{0,m} w'' a'_0$ immediately has to leave node 2, but then again this string gets lost as it cannot enter any other node, especially node 1 cannot take the string because it still contains a symbol from A' which is not allowed due to FI_1.

In node 1, the rules **1.4** and **1.6** have to be used in any order, the resulting string then can only be sent to node 2:

$$(1, q''_{k,j-1} w a''_1) \overset{\{\mathbf{1.4},\mathbf{1.6}\}}{\Longrightarrow} (1, q'_{k,j-1} w a'_1) \vdash (2, q'_{k,j-1} w a'_1)$$

In the description of the derivation above, if more than one evolution step is necessary in a node before a string can leave this node, again we do not indicate the intermediate communication step leaving the string in the node, i.e., instead of writing

$$(1, q''_{k,j-1} w a''_1) \overset{\mathbf{1.4}}{\Longrightarrow} (1, q'_{k,j-1} w a''_1) \vdash$$
$$(1, q'_{k,j-1} w a''_1) \overset{\mathbf{1.6}}{\Longrightarrow} (1, q'_{k,j-1} w a'_1)$$

we simply have written

$$(1, q''_{k,j-1} w a''_1) \overset{\{\mathbf{1.4},\mathbf{1.6}\}}{\Longrightarrow} (1, q'_{k,j-1} w a'_1).$$

Moreover, we assume that this notation now also includes the variant where the rules are applied in reverse order, i.e., the derivation

$$(1, q''_{k,j-1} w a''_1) \overset{\mathbf{1.6}}{\Longrightarrow} (1, q''_{k,j-1} w a'_1) \vdash$$
$$(1, q''_{k,j-1} w a'_1) \overset{\mathbf{1.4}}{\Longrightarrow} (1, q'_{k,j-1} w a'_1).$$

In both cases, the output filters of M_1 only allow a string to leave after the application of both rules.

The computation now can be continued in the same way as above by toggling between node 2—there applying rule **2.1** after rule **2.2**—and node 1—there applying the rules **1.4**, **1.6**; the main idea of this construction is to decrease

the second index h of the state symbols $q''_{k,h}/q'_{k,h}$ while increasing the index g for the symbols a''_g/a'_g:

$$(2, q'_{k,j-1}wa'_1) \overset{<\mathbf{2.2},\mathbf{2.1}>}{\Longrightarrow} (2, q''_{k,j-2}wa''_2) \overset{\{\mathbf{1.4},\mathbf{1.6}\}}{\Longrightarrow} \ldots$$
$$(1, q'_{k,1}wa'_{j-1}) \vdash (2, q'_{k,1}wa'_{j-1}).$$

By construction, the index h reaches 1 at the same moment when g reaches $j-1$. At the end, in node 2, only the rules **2.3** and **2.1** are to be applied, and the resulting string $q_kwa''_j$ enters node 1, where the application of rule $\mathbf{1.7} : a''_j \to a_j$ leads to the desired correct string q_kwa_j.

$$(2, q'_{k,1}wa'_{j-1}) \overset{<\mathbf{2.3},\mathbf{2.1}>}{\Longrightarrow} (2, q_kwa''_j) \vdash (1, q_kwa''_j) \overset{\mathbf{1.7}}{\Longrightarrow}$$
$$(1, q_kwa_j).$$

So far we have shown how Γ can simulate the application of the insertion rule $q_i \to a_jq_k$ in M correctly. Due to the input and output filters of the nodes, the rules have to be applied as described above, as otherwise the resulting strings get lost:

If in node 2, rule **2.1** is applied before rule **2.2** or rule **2.3**, the resulting string $q'_{k,j-s}wa''_{s+1}$ would have to leave node 2 immediately, but the input filters of the other nodes would not allow the string to enter, hence this string would get lost; especially node 1 cannot take the string because it does not contain a symbol from PI_1.

As already argued earlier, the application of the rule $\mathbf{2.7} : a_m \to \hat{a}_{0,m}$ yields strings which immediately have to leave node 2. The resulting strings $q'_{k,j-s}w'\hat{a}_{0,m}w''a'_s$ get lost as they cannot enter any other node, especially node 1 cannot take such strings because they still contain a symbol from A' which is not allowed due to FI_1. If rule **2.7** is applied after the application of **2.2** or **2.3**, the resulting strings $q''_{k,j-s-1}w'\hat{a}_{0,m}w''a'_s$ or $q_kw'\hat{a}_{0,m}w''a'_j$ immediately have to leave node 2, but then again these strings get lost as they cannot enter any other node, especially node 1 cannot take these strings because they still contain a symbol from A' which is not allowed due to FI_1.

Another situation to be checked is to see what happens if rule **1.7** is not applied in the right moment: if rule **1.7** is applied too early, the resulting string $q'_{k,l}wa_j$ has to leave node 1, but cannot enter any other node and therefore gets lost; especially we observe that PI_2 is not fulfilled. If, on the other hand, rule **1.6** is applied to the string $q_kwa''_j$, after the application of rule **1.1** the resulting string $\tilde{q}_kwa'_j$ now can leave node 1, but cannot enter any other node and therefore gets lost.

In sum, we conclude that the OHNEP Γ has correctly simulated the application of the insertion rule $q_i \to a_jq_k$ in M.

Case 2 Consider a deletion rule $q_ia_j \to q_k \in R$, $q_i \in Q_1$, $q_k \in Q$, $a_j \in \Sigma$, and let $q_ia_jw \Longrightarrow q_kw$ be a computation step in M, i.e., rule $q_ia_j \to q_k$ is applied to the string q_ia_jw yielding q_kw.

Starting with the string q_ia_jw being situated in node 1 of Γ, we now describe the possible evolutions of this string q_ia_jw in Γ.

The simulation of the deletion rule $q_ia_j \to q_k$ also starts with the application of rule **1.1**, but the resulting string then is sent to node 2, where one symbol from Σ is chosen to be marked by the rule **2.7** after the application of a rule **2.6**. The correct position of the symbol from Σ to be marked is the first position after the state symbol. As we will argue later, making another choice will lead to a string which at some moment will get stuck without ever being able to reach the output node.

If rule **2.6** and then rule **2.7** is applied at the correct position in the string, we get the following derivation:

$$(1, q_ia_jw) \overset{\mathbf{1.1}}{\Longrightarrow} (1, \tilde{q}_ia_jw) \vdash (2, \tilde{q}_ia_jw) \overset{<\mathbf{2.6},\mathbf{2.7}>}{\Longrightarrow}$$
$$(2, \hat{q}_i\hat{a}_{0,j}w) \vdash (1, \hat{q}_i\hat{a}_{0,j}w).$$

The two rules **2.6** and **2.7** have to be applied in exactly this order. If rule **2.7** is applied first, then the resulting string $\tilde{q}_i\hat{a}_{0,j}w$ gets lost, as it has to leave node 2, but cannot enter any other node; especially FI_1 forbids strings with symbols from $\tilde{Q}_1$ to enter node 1.

In node 1, only the rules **1.8** and **1.9** can be applied, and on the other hand, both rules have to be applied before the resulting string can leave; the only node allowing the string to enter then is node 2:

$$(1, \hat{q}_i\hat{a}_{0,j}w) \overset{\{\mathbf{1.8},\mathbf{1.9}\}}{\Longrightarrow} (1, \bar{q}_i\bar{a}_{0,j}w) \vdash (2, \bar{q}_i\bar{a}_{0,j}w).$$

In node 2, the rules $\mathbf{2.5} : \bar{q}_l \to \hat{q}_{l-1}$ and $\mathbf{2.4} : \bar{a}_{s,t} \to \hat{a}_{s+1,t}$ have to be applied in this order; if **2.4** is applied first, the resulting string $\bar{q}_i\hat{a}_{1,j}w$ gets lost, as it has to leave node 2, but cannot enter any other node; especially FI_1 forbids strings with symbols from $\bar{Q}$ to enter node 1.

Rule **2.5** decreases the index l of the state symbol, whereas by rule **2.4** the index s of the symbol $\bar{a}_{s,t}$ is increased. We now toggle between node 2—there applying the rules **2.5** and **2.4** in this specific order—and node 1—there applying the rules **1.8** and **1.9**. Again, throughout the whole circle, in node 2 rule **2.4** must not be applied before rule **2.5** as otherwise, as argued above, the resulting strings would be lost, as with a symbol from $\bar{Q}$ they are not allowed to enter node 1.

Moreover, if we apply rule **2.7** once more before applying rule **2.4** in node 2, the resulting strings $\bar{q}_{i-l}\bar{a}_{l,j}w'\hat{a}_{0,m}w''$ or $\hat{q}_{i-l-1}\bar{a}_{l,j}w'\hat{a}_{0,m}w''$ get lost, as they have to leave node 2, but cannot enter any other node; especially FI_1 forbids strings with symbols from $\bar{A}$ to enter node 1.

By construction, the index l of the state symbol reaches 0 at the same moment when index s reaches k.

At the end, in node 1 the rules $\mathbf{1.11}: \hat{q}_0 \to \varepsilon$ and $\mathbf{1.10}: \hat{a}_{i,j} \to s_k$ such that $q_i a_j \to q_k \in R$ are to be applied:

$$(2, \bar{q}_i \bar{a}_{0,j} w) \overset{<\mathbf{2.5},\mathbf{2.4}>}{\Longrightarrow} (2, \hat{q}_{i-1} \hat{a}_{1,j} w) \vdash$$
$$(1, \hat{q}_{i-1} \hat{a}_{1,j} w) \overset{\{\mathbf{1.8},\mathbf{1.9}\}}{\Longrightarrow} \ldots (1, \bar{q}_1 \bar{a}_{i-1,j} w) \vdash$$
$$(2, \bar{q}_1 \bar{a}_{i-1,j} w) \overset{<\mathbf{2.5},\mathbf{2.4}>}{\Longrightarrow} (2, \hat{q}_0 \hat{a}_{i,j} w) \vdash$$
$$(1, \hat{q}_0 \hat{a}_{i,j} w) \overset{\{\mathbf{1.10},\mathbf{1.11}\}}{\Longrightarrow} (1, \varepsilon s_k w).$$

If rule $\mathbf{1.11}: \hat{q}_0 \to \varepsilon$ has to be applied, but no rule $\mathbf{1.10}: \hat{a}_{i,j} \to s_k$ can be applied and instead still rule $\mathbf{1.8}: \hat{a}_{s,t} \to \bar{a}_{s,t}$ is applied, the string can leave node 1, but cannot enter any other node and therefore gets lost: because of the symbol from $\bar{A}$, the string cannot enter any of the nodes 3, 4, and 5, but FI_2 forbids strings containing ε to enter node 2.

On the other hand, if a rule $\mathbf{1.10}: \hat{a}_{i,j} \to s_k$ is applied, but rule $\mathbf{1.11}: \hat{q}_0 \to \varepsilon$ cannot yet be applied, then the resulting string $\bar{q}_l s_k w$ can leave node 1, but also cannot enter any other node and therefore gets lost: because of the symbol from S, the string cannot enter node 2, and because of the state symbol $\bar{q}_l$ it cannot enter any of the nodes 3, 4, and 5.

Moreover, in node 1 we might also apply the sequence of rules $\mathbf{1.10}, \mathbf{1.12}, \mathbf{1.1}, \mathbf{1.11}$ to the string $\hat{q}_0 \hat{a}_{i,j} w$ thus obtaining the string $\varepsilon \tilde{q}_k w$, but the input filters of the nodes do not allow this string to enter, especially ε is forbidden by FI_2 and $\tilde{q}_k$ is forbidden by FI_4.

The simulation of the deletion operation ends with the following derivation steps in Γ:

$$(1, \varepsilon s_k w) \vdash (4, \varepsilon s_k w) \overset{\mathbf{4.1}}{\Longrightarrow} (4, s_k w) \vdash (1, s_k w) \overset{\mathbf{1.12}}{\Longrightarrow} (1, q_k w).$$

With the application of the rule $\mathbf{1.12}: s_k \to q_k$ we have successfully completed the simulation of the application of the deletion rule $q_i a_j \to q_k \in R$, $q_i \in Q_1$, $q_k \in Q$, $a_j \in \Sigma$, to the string $q_i a_j w$, i.e., the computation step $q_i a_j w \Longrightarrow q_k w$ in M yielding $q_k w$ from $q_i a_j w$.

If we apply rule $\mathbf{2.7}$ at a position $i > 2$, i.e., to the a_j in $q_i w' a_j w''$ with $|w'| > 0$, then, at the end of the simulation (carried out as described above) we have the string $w' q_k w''$ in node 1.

First consider $q_k \in Q_1$; then $w' q_k w''$ has to be transformed into $w_1 \varepsilon w_2$, $w_1, w_2 \in V^*$, $|w_1| > 0$, and at the end the derived string gets lost in node 4, as the rule $\varepsilon \to \lambda$ can only be applied at the left end of a string in node 4; hence, such a string will not "survive" the next evolutionary step of the OHNEP Γ (whereas an HNEP Γ would keep the string in the node, but cannot let it out because of FO_4).

If $q_k \in Q_2$, then in several circles of the computation the developing string will look as $w_1 q_t w_2$, $|w_1| > 0$, $q_t \in Q_1$, and we return to the case considered before.

If $q_k = q_0$, then, as we will discuss in the next case, we get $w' \varepsilon w''$ in node 1 and then in node 4, but again rule $\varepsilon \to \lambda$ cannot be applied as ε is not at the left end of the string and therefore such a string gets lost in node 4.

In sum, we conclude that the OHNEP Γ correctly simulates the application of rule $q_i a_j \to q_k$ in M.

Case 3 As soon as a string $q_0 w_0$ with the output node q_0 at its beginning (and $w_0 \in T^*$) appears in node 1, which in fact means that the circular Post machine M has stopped with having computed w_0, we can apply rule $\mathbf{1.2}: q_0 \to \varepsilon$ and send the resulting string εw_0 to node 4 where ε is erased by rule $\mathbf{4.1}: \varepsilon \to \lambda$; the resulting terminal string $w_0 \in T^*$ then can enter the output node 5, i.e., we get w_0 as the result of this computation in Γ:

$$(1, q_0 w_0) \overset{\mathbf{1.2}}{\Longrightarrow} (1, \varepsilon w_0) \vdash (4, \varepsilon w_0) \overset{\mathbf{4.1}}{\Longrightarrow} (4, w_0) \vdash (5, w_0).$$

So far we implicitly have assumed that the result of the application of a substitution or a deletion operation to a given string v yields the empty set if the operation is not applicable, i.e., above we mostly have described all possible evolutions of a string based on Definition 1 only, i.e., for Γ being an OHNEP. Finally, we now are going to investigate what happens in case Γ is an HNEP, i.e., if we use the original definition (see Remark 3) yielding the string v itself, i.e., $\{v\}$ in such cases where one of the operations assigned to a node is not applicable to v.

Already when defining the evolutionary processors, we have explained what happens to a string with respect to these two different definitions in node 4: using the original definition as explained in Remark 3, a string containing ε at another position than the leftmost one remains in the node, but cannot leave it as ε is prohibited by FO_4, whereas when using obligatory HNEPs according to Definition 1, such a string is "killed" during the next evolutionary step. In both cases, the string gets lost in node 4.

In node 3, the rule $\mathbf{3.1}: \lambda \to a_0$ is always applicable, hence, for such rules there is no difference between the two definitions.

As the rule set M_5 is empty, any string having entered node 5 will be "killed" during the next evolutionary step, in both variants.

Only for the remaining two nodes, node 1 and node 2, we have the situation that for any string v entering these nodes there will always be a rule which is not applicable, i.e., with the original definition as explained in Remark 3 we always also get v as a possible result of the next evolutionary step.

As for node 2 we have $FI_2 \supset A'' \cup \hat{A} = PO_2$, a string having been able to enter node 2 cannot leave it without having undergone an evolution by one of the rules from M_2.

For node 1, the situation is slightly different, as here we have $FI_1 \supset \tilde{Q}_1 \cup \bar{Q} \cup Q' \cup \{\varepsilon\} = PO_1 \backslash \tilde{Q}_2$.

If the string containing a symbol from $\tilde{Q}_2$ has been coming back from node 3, then it also contains the symbol a_0; hence, FO_1 forbids the string to leave node 1 before it has been changed to a_0'.

If the string is the one obtained directly after the application of rule **1.1**, containing only one symbol from $\tilde{Q}_2$ and only symbols from Σ, then no further rule can be applied in node 1, the string is not affected during the next evolutionary step, but in the succeeding communication step it again can try to leave node 1 and enter node 3.

In all cases, the validity of the proof does not depend on the definitions of the results after the application of a substitution or deletion operation. These observations will carry over to the further results stated in this paper.

In sum, we observe that every computation of the CPM5 in normal form M can be simulated correctly by the [O]HNEP Γ, yielding exactly the same results; any other computation paths in Γ not correctly simulating the computation steps of M do not yield any result, which observation completes the proof. □

Remark 7 So far we have assumed that the underlying communication graph in the [O]HNEP is a complete graph without loops. Yet it is easy to see that the whole construction elaborated in the proof of Theorem 2 still works if we allow a loop in every node:

Let us first repeat some of the arguments already exhibited when defining the evolutionary processors in the proof given for Theorem 2:

As already explained above, no string having entered node 5 will ever be able leave it.

In node 3, the rule $\mathbf{3.1} : \lambda \rightarrow a_0$ is always applicable, yet the resulting string then contains the symbol a_0 and therefore cannot re-enter the node because of the input filter of forbidden contexts FI_3.

Due to the interplay of the input filter of permitting contexts $PI_4 = \{\varepsilon\}$ and the output filter of forbidden contexts $FO_4 = \{\varepsilon\}$, a string which can leave node 4 cannot re-enter the node.

A similar argument can be applied for node 2, because the input filter of forbidden contexts $FI_2 \supset A'' \cup \hat{A}$ does not allow strings having been sent out by containing one of the symbols in the output filter of permitting contexts $PO_2 = A'' \cup \hat{A}$ to re-enter the node.

For node 1, the situation is slightly different; the interplay of the input filter of forbidden contexts $FI_1 \supset \tilde{Q}_1 \cup \bar{Q} \cup Q' \cup \{\varepsilon\}$ and the output filter of permitting contexts $PO_1 = \tilde{Q} \cup \bar{Q} \cup Q' \cup \{\varepsilon\}$ does not allow strings having been sent out by containing one of the symbols in the output filter of permitting contexts to re-enter the node except for symbols from $\tilde{Q}_2$:

If the string containing a symbol from $\tilde{Q}_2$ is coming back from node 3, then it also contains the symbol a_0; FO_1 forbids the string to leave node 1 before it has been changed to a_0', but then FI_1 forbids the resulting string to re-enter the node.

If the string is the one obtained directly after the application of rule **1.1**, containing only one symbol from $\tilde{Q}_2$ and only symbols from Σ, we have to distinguish two cases depending on the underlying definition of the result of substitution operations on a string: taking the original definition (see Remark 3), as no rule can be applied, the string is not affected during the next evolutionary step, but in the succeeding communication step again can try to leave node 1 and enter node 3; using OHNEPs according to Definition 1, the string is "killed" during the next evolutionary step.

In both cases, the validity of the proof given for Theorem 2 is not affected if we allow complete graphs to have loops. Obviously, this also holds true for the further results stated in this section.

As an obvious consequence of Theorem 2 we obtain the desired completeness result for complete [O]HNEPs with only five nodes:

Corollary 1 *Complete hybrid networks of evolutionary processors (complete [O]HNEPs) with* 5 *nodes are computationally complete.*

Proof As the circular Post machines of type 5 (CPM5) in normal form are computationally complete (see Theorem 1), the result directly follows from our main result, Theorem 2. □

The following two results are immediate consequences of Corollary 1, as any recursively enumerable language L can be viewed as partial recursive relation $L \times \{\lambda\}$ (acceptance) and $\{\lambda\} \times L$ (generation), see Remark 2.

Corollary 2 *Any recursively enumerable language L can be accepted by a complete A[O]HNEP of size* 5.

Corollary 3 *Any recursively enumerable language L can be generated by a complete G[O]HNEP of size* 5.

4 [O]HNEPs with terminal extraction

As can be seen from the proof of our main result, Theorem 2, the fifth node is only needed to collect the terminal strings which come from node 4. Hence, collecting all the

terminal strings appearing there, we could completely avoid the fifth node. Therefore, we now define *extended HNEPs (HNEPs with terminal extraction)* which allow for this variant of getting the terminal results. Again we will use the bracket notation with [O] to capture both variants EOHNEPs and EHNEPs.

Definition 5 An *extended hybrid network of evolutionary processors* (an *E[O]HNEP* for short) over V is an [O]HNEP $\Gamma = (V, T, H, \mathcal{N}, C_{\text{init}}, \alpha, \beta, C_{\text{input}}, i_0)$ where the results of computations are extracted as the terminal strings from the output node i_0.

The following result is an immediate consequence of Theorem 2:

Theorem 3 *Any (non-deterministic) CPM5 M in normal form can be simulated by a complete E[O]HNEP Γ of size* 4.

Proof For a given (non-deterministic) CPM5 in normal form we construct a complete E[O]HNEP Γ of size 4 by taking over the whole construction from the proof of Theorem 2 with the following small modifications:

Of course, we only take the first four processors as described there, omitting the fifth node. Node 4 now is the new output node, defined as follows:

$$
\begin{aligned}
M_4 &= \{\mathbf{4.1} : \varepsilon \to \lambda\},\\
PI_4 &= \{\varepsilon\},\\
FI_4 &= V \backslash (S \cup \Sigma \cup \{\varepsilon\}),\\
PO_4 &= V \backslash T,\\
FO_4 &= \{\varepsilon\}.
\end{aligned}
$$

In contrast to the previous construction, we now take the output filter of permitting contexts $PO_4 = V \backslash T$ instead of taking $PO_4 = \emptyset$, which guarantees that the terminal strings cannot leave node 4 any more. Taking the original definition for the result of the deletion operation (see Remark 3), eventually as some other "wrong" strings, any terminal string will stay in node 4 forever and can be extracted from there at any moment. When using OHNEPs according to Definition 1, terminal strings will vanish in the next evolutionary steps, but this makes no difference for the result, as according to the definitions, a terminal string just has to appear for one moment in the output node. □

Remark 8 Again the underlying communication graph in the E[O]HNEP can be a complete graph with or without loops. All the arguments elaborated in Remark 7 are still valid; changing $PO_4 = \emptyset$ to $PO_4 = V \backslash T$ does not affect the validity of the proof.

The following results are immediate consequences of Theorem 3:

Corollary 4 *Complete E[O]HNEPs with* 4 *nodes are computationally complete.*

Corollary 5 *Any recursively enumerable language L can be accepted by a complete AE[O]HNEP of size* 4.

Corollary 6 *Any recursively enumerable language L can be generated by a complete GE[O]HNEP of size* 4.

5 [O]HNEPs on star-like graphs

Another careful look into the proofs of Theorems 3 and 2 shows that the communicating of strings only happens between node 1 and any of the other nodes, but there is no communicating of strings between any two of the nodes 2–4 or 2–5, respectively, with the only exception that in the last step of the proof of Theorem 2 the terminal string has to be sent from the deletion node 4 directly to the output node 5. Hence, we conclude that the proofs have already been designed in such a way that by taking node 1 as the central node, we only need a star-like communication structure for extended [O]HNEPs.

The following results therefore are immediate consequences of these observations and the results established earlier in this paper:

Theorem 4 *Any (non-deterministic) CPM5 M in normal form can be simulated by a star-like E[O]HNEP Γ of size* 4.

Corollary 7 *Star-like E[O]HNEPs with* 4 *nodes are computationally complete.*

Corollary 8 *Any recursively enumerable language L can be accepted by a star-like AE[O]HNEP of size* 4.

Corollary 9 *Any recursively enumerable language L can be generated by a star-like GE[O]HNEP of size* 4.

In the case of non-extended star-like [O]HNEPs the main problem is that for allowing a terminal string to pass from the deletion node to the output node through the central node, the permitting input filter of this central node has to be the empty set. Unfortunately, our proof technique could not be adapted to achieve a computational completeness result with only five nodes due to this special required feature. On the other hand, by adding a new sixth node as the central node, we can easily obtain the following results:

Theorem 5 *Any (non-deterministic) CPM5 M in normal form can be simulated by a star-like [O]HNEP Γ of size* 6.

Proof For a given (non-deterministic) CPM5 in normal form we construct a star-like [O]HNEP Γ of size 6 by taking over the whole construction from the proof of Theorem 2 with just adding a new node 6 as the central node:

$M_6 = \{\mathbf{6.1} : a \to a \mid a \in V\}$,
$PI_6 = \emptyset$,
$FI_6 = \emptyset$,
$PO_6 = \emptyset$,
$FO_6 = \emptyset$.

Obviously, any non-empty string (the empty string can only be terminal and therefore only appear in the output node 5, but no string can ever leave the output node) can enter this central node and leave it immediately in the next communication step without having been changed by the application of any of the substitution rules **6.1**. Hence, all our arguments for the complete [O]HNEP constructed in the proof of Theorem 2 can be taken over for this star-like [O]HNEP, as in fact the new central node allows to have the same derivations as in the complete [O]HNEP with just an intermediate communication and evolutionary step (which does not affect the string passing through the central node) for any communication step there.

As a final observation we mention that node 6 also allows strings to go back to the node they have come from, but this just resembles the case of having complete graphs with loops as the underlying communication structure, yet as already argued in Remark 7 this does not affect the correctness of the proof of Theorem 2. Obviously, the underlying communication structure in this proof can be complete graphs without or with loops, too; the same also holds true for all further results stated in this section. □

Remark 9 In the case of star-like OHNEPs, the empty string cannot pass the central node to reach the output node, so computational completeness here can only be obtained without taking into account the empty string. Observe that the central node itself cannot serve as output node, as every string passing through it then would be a result of a computation. These restrictions do not only apply to our construction given in the proof of Theorem 5, but are valid for any other construction, too, i.e., for star-like OHNEPs computational completeness means not taking into account the empty string as a result.

On the other hand, it is easy to see that for AOHNEPs and GOHNEPs the empty string can be in the language accepted or generated by the OHNEP: In the accepting case, the input string is q_1 in node 1, i.e., a non-empty string, and without loss of generality we may assume that acceptance is obtained with some non-empty string. In the generating case, λ can be put directly into the output node 5 if the empty string is in the language to be generated.

With respect to the considerations of Remark 9, the following results then are immediate consequences of the preceding theorem:

Corollary 10 *Star-like [O]HNEPs with* 6 *nodes are computationally complete.*

Corollary 11 *Any recursively enumerable language L can be accepted by a star-like A[O]HNEP of size* 6.

Corollary 12 *Any recursively enumerable language L can be generated by a star-like G[O]HNEP of size* 6.

6 [O]HNEPs on linear graphs

With some changes in the proof of Theorem 2, we are able to obtain a similar result for linear [O]HNEPs. With respect to the notions used there, we will establish the result for linear [O]HNEPs on the linear structure of processors $3 - 1 - 2 - 4 - 5$.

The main difference is that the strings where the first symbol should be deleted cannot go directly to the deletion node, but have to pass through node 2 and then go back from node 4 to node 1 through node 2 again. The rest of the construction can be taken over with only few changes, but for the sake of completeness we will give a complete definition of the linear HNEP in the proof of the following theorem.

Theorem 6 *Any (non-deterministic) CPM5 M in normal form can be simulated by a linear [O]HNEP Γ of size* 5.

Proof Let $M = (\Sigma, T, Q, q_1, q_0, R)$ be a (non-deterministic) CPM5 in the normal form as defined in Definition 4, with symbols $\Sigma = \{a_j \mid 1 \le j \le m\}$ and states $Q = \{q_i \mid 0 \le i \le n\}$, where q_1 is the initial state and the only terminal state is $q_0 \in Q$. We now construct a linear [O]HNEP $\Gamma = (V, T, H, \mathcal{N}, \alpha, \beta, C_0^0, 5)$ of size 5 which simulates the given CPM5 M based on the linear communication graph $3 - 1 - 2 - 4 - 5$.

The sets used in the description of the system are nearly the same as in the proof of Theorem 2; we only need the additional symbol $\bar{\varepsilon}$ and the primed versions of the symbols in S, i.e., we now have

$$V = S \cup S' \cup A \cup A' \cup A'' \cup \bar{A} \cup \hat{A} \cup Q \cup \tilde{Q} \cup \bar{Q} \cup \hat{Q} \cup Q' \cup Q'' \cup \{\varepsilon, \bar{\varepsilon}\},$$
$$S' = \{s' \mid s \in S\} = \{s'_i \mid i \in J_k\}.$$

We take $C_0^0 = \{(1, q_1)\}$, and the output node of Γ for collecting the results of a computation is node 5.

Moreover, we take $\beta(i) = 2$ for all $1 \leq i \leq 5$ as well as $\alpha(1) = \alpha(2) = \alpha(5) = *$, $\alpha(3) = r$, and $\alpha(4) = l$. The evolutionary processors $\mathcal{N}(i) = (M_i, PI_i, FI_i, PO_i, FO_i)$, $1 \leq i \leq 5$, are defined in a similar way as in the proof of Theorem 2; the labels for identifying the rules are primed if they differ from the rules defined there or are new.

Again the simulation of each rule of M starts in node 1: if the current state q is from $q \in Q_1 \cup Q_2$, rule **1.1** is applied. In the succeeding communication step, if $q \in Q_1$, the string has to go to node 2, where the symbol to be deleted is chosen; if $q \in Q_2$, the string has to go to node 3, where the symbol a_0 is inserted on the right end of the string. If $q = q_0$, we now have to apply rule **1.2**$'$ thus obtaining the symbol $\bar{\varepsilon}$, which directs the string to node 4, yet now passing through node 2, where the application of rule **2.9**$'$ yields ε from $\bar{\varepsilon}$, and after the deletion of ε a terminal string is obtained in node 5.

$$
\begin{aligned}
M_1 = {} & \{\mathbf{1.1} : q \to \tilde{q} \mid q \in Q_1 \cup Q_2\} \\
& \cup \{\mathbf{1.2}' : q_0 \to \bar{\varepsilon}\} \\
& \cup \left\{\mathbf{1.3} : \tilde{q}_i \to q'_{k,j} \mid q_i \to a_j q_k \in R\right\} \\
& \cup \left\{\mathbf{1.4} : q''_{s,t} \to q'_{s,t} \mid q_s \in Q,\ t \in J_\Sigma\right\} \\
& \cup \left\{\mathbf{1.5} : a_0 \to a'_0\right\} \\
& \cup \left\{\mathbf{1.6} : a''_l \to a'_l \mid l \in J_\Sigma\right\} \\
& \cup \left\{\mathbf{1.7} : a''_l \to a_l \mid l \in J_\Sigma\right\} \\
& \cup \left\{\mathbf{1.8} : \hat{a}_{s,t} \to \bar{a}_{s,t} \mid s \in J_K,\ t \in J_\Sigma\right\} \\
& \cup \{\mathbf{1.9} : \hat{q}_l \to \bar{q}_l \mid l \in J_K \backslash \{0\}\} \\
& \cup \left\{\mathbf{1.10} : \hat{a}_{i,j} \to s_k \mid q_i a_j \to q_k \in R,\quad i \in J_K \backslash \{0\}\right\} \\
& \cup \{\mathbf{1.11}' : \hat{q}_0 \to \bar{\varepsilon}\}, \\
& \cup \left\{\mathbf{1.12}' : s'_k \to q_k \mid k \in J_K\right\}, \\
PI_1 = {} & S' \cup Q \cup \tilde{Q}_2 \cup \hat{Q} \cup Q'', \\
FI_1 = {} & S \cup A' \cup \bar{A} \cup \tilde{Q}_1 \cup \bar{Q} \cup Q' \cup \{\varepsilon, \bar{\varepsilon}\}, \\
PO_1 = {} & S \cup \tilde{Q} \cup \bar{Q} \cup Q' \cup \{\bar{\varepsilon}\}, \\
FO_1 = {} & S' \cup Q \cup \hat{Q} \cup Q'' \cup A'' \cup \hat{A} \cup \{a_0\}.
\end{aligned}
$$

With respect to the rules needed for the simulation of insertion rules, we use the newly introduced symbols from S' at the end of the simulation; therefore, we have to replace rules **2.3** by the rules **2.3**$'$ and add the new rules **1.12**$'$ in M_1. Moreover, the filters PI_1 and FO_1 as well as FI_2 therefore are extended by S'.

With respect to the rules needed for the simulation of deletion rules, rules **1.12** have been taken away from M_1, instead in M_2 we have added the rules **2.8**$'$; rule **2.9**$'$ in M_2 and rules **1.12**$'$ in M_1 have been added, too. Moreover, the new structure causes some small modifications of the input and output filters, especially involving the symbols $\varepsilon, \bar{\varepsilon}$, as well as some modifications in the description of the simulation of deletion rules.

$$
\begin{aligned}
M_2 = {} & \left\{\mathbf{2.1} : a'_{l-1} \to a''_l \mid l \in J_\Sigma\right\} \\
& \cup \left\{\mathbf{2.2} : q'_{s,t} \to q''_{s,t-1} \mid q_s \in Q,\ t \in J_\Sigma \backslash \{1\}\right\} \\
& \cup \left\{\mathbf{2.3}' : q'_{k,1} \to s'_k \mid k \in J_k\right\} \\
& \cup \left\{\mathbf{2.4} : \bar{a}_{k,t} \to \hat{a}_{k+1,t} \mid k \in J_K \backslash \{n\}, t \in J_\Sigma\right\} \\
& \cup \{\mathbf{2.5} : \bar{q}_l \to \hat{q}_{l-1} \mid l \in J_K \backslash \{0\}\}, \\
& = \left\{\mathbf{2.6} : \tilde{q}_i \to \hat{q}_i \mid q_i a_j \to q_k \in R\right\}, \\
& \cup \left\{\mathbf{2.7} : a_j \to \hat{a}_{0,j} \mid j \in J_\Sigma\right\}, \\
& \cup \{\mathbf{2.8}' : s_k \to q_k \mid k \in J_K\}, \\
& \cup \{\mathbf{2.9}' : \bar{\varepsilon} \to \varepsilon\}, \\
PI_2 = {} & S \cup A' \cup \bar{A} \cup \tilde{Q}_1 \cup \{\bar{\varepsilon}\}, \\
FI_2 = {} & S' \cup A'' \cup \hat{A} \cup Q \cup \tilde{Q}_2 \cup \hat{Q} \cup Q'' \cup \{\varepsilon\}, \\
PO_2 = {} & A'' \cup \hat{A} \cup Q \cup \{\varepsilon\}, \\
FO_2 = {} & \emptyset.
\end{aligned}
$$

In node 3, the insertion of any symbol from Σ starts with inserting a_0 on the right end of a string allowed to enter this node.

$$
\begin{aligned}
M_3 = {} & \{\mathbf{3.1} : \lambda \to a_0\}, \\
PI_3 = {} & \tilde{Q}_2, \\
FI_3 = {} & V \backslash (\tilde{Q}_2 \cup \Sigma), \\
PO_3 = {} & \{a_0\}, \\
FO_3 = {} & \emptyset.
\end{aligned}
$$

The fourth node carries out the deletion process, with the only symbol, i.e., ε, to be deleted on the left end of a string.

$$
\begin{aligned}
M_4 = {} & \{\mathbf{4.1} : \varepsilon \to \lambda\}, \\
PI_4 = {} & \{\varepsilon\}, \\
FI_4 = {} & V \backslash (S \cup \Sigma \cup \{\varepsilon\}), \\
PO_4 = {} & \emptyset, \\
FO_4 = {} & \{\varepsilon\}.
\end{aligned}
$$

In the output node 5, the terminal strings are collected.

$$
\begin{aligned}
M_5 = {} & \emptyset, \\
PI_5 = {} & \emptyset, \\
FI_5 = {} & V \backslash T, \\
PO_5 = {} & \emptyset, \\
FO_5 = {} & \emptyset.
\end{aligned}
$$

The simulation of an insertion rule is performed in a similar way as in the proof of Theorem 2, starting in node 1 with the application of rule **1.1**, continuing with rule **3.1** in node 3, and then toggling between node 1 and node 2, but now first ending up in node 2 with $q'_{k,1} w a'_{j-1}$. The simulation then continues as follows:

$$(2, q'_{k,1} w a'_{j-1}) \overset{<\mathbf{2.3'},\mathbf{2.1}>}{\Longrightarrow} (2, s'_k w a''_j) \vdash$$
$$(1, s'_k w a''_j) \overset{\{\mathbf{1.7},\mathbf{1.12'}\}}{\Longrightarrow} (1, q_k w a_j).$$

Again, we have obtained a correct simulation of the insertion rule from R in Γ.

If $q_0 w_0$ for $w_0 \in T^*$ is obtained in node 1, as already explained earlier in this proof, we have the following derivation, thus finally getting w_0 in node 5 as the result of this computation in Γ:

$$(1, q_0 w_0) \overset{\mathbf{1.2'}}{\Longrightarrow} (1, \bar{\varepsilon} w_0) \vdash (2, \bar{\varepsilon} w_0) \overset{\mathbf{2.9'}}{\Longrightarrow} (2, \varepsilon w_0) \vdash$$
$$(4, \varepsilon w_0) \overset{\mathbf{4.1}}{\Longrightarrow} (4, w_0) \vdash (5, w_0).$$

If rule **2.7** is applied instead of **2.9′** in this derivation, the resulting strings have to leave node 2, but then cannot enter node 1 or node 4.

Therefore, we now only have to discuss the changes for the simulation of deletion rules in more detail:

The simulation of the deletion rule $q_i a_j \to q_k \in R$ also starts with the application of rule **1.1**, the resulting string then is sent to node 2, where one symbol from Σ is chosen to be marked by the rule **2.7** after the application of a rule **2.6**; the correct position of the symbol from Σ to be marked is the first position after the state symbol.

The further simulation of the deletion rule now proceeds by toggling between node 2 and node 1 as already described in the proof of Theorem 2:

$$(1, q_i a_j w) \overset{\mathbf{1.1}}{\Longrightarrow} (1, \tilde{q}_i a_j w) \vdash$$
$$(2, \tilde{q}_i a_j w) \overset{<\mathbf{2.6},\mathbf{2.7}>}{\Longrightarrow} (2, \hat{q}_i \hat{a}_{0,j} w) \vdash$$
$$(1, \hat{q}_i \hat{a}_{0,j} w) \overset{\{\mathbf{1.8},\mathbf{1.9}\}}{\Longrightarrow} (1, \bar{q}_i \bar{a}_{0,j} w) \vdash$$
$$(2, \bar{q}_i \bar{a}_{0,j} w) \overset{<\mathbf{2.5},\mathbf{2.4}>}{\Longrightarrow} (2, \hat{q}_{i-1} \hat{a}_{1,j} w) \vdash$$
$$(1, \hat{q}_{i-1} \hat{a}_{1,j} w) \overset{\{\mathbf{1.8},\mathbf{1.9}\}}{\Longrightarrow} \ldots (1, \bar{q}_1 \bar{a}_{i-1,j} w) \vdash$$
$$(2, \bar{q}_1 \bar{a}_{i-1,j} w) \overset{<\mathbf{2.5},\mathbf{2.4}>}{\Longrightarrow} (2, \hat{q}_0 \hat{a}_{i,j} w) \vdash$$
$$(1, \hat{q}_0 \hat{a}_{i,j} w) \overset{\{\mathbf{1.10},\mathbf{1.11}\}}{\Longrightarrow} (1, \bar{\varepsilon} s_k w).$$

If rule $\mathbf{1.11}: \hat{q}_0 \to \bar{\varepsilon}$ has to be applied, but no rule $\mathbf{1.10}: \hat{a}_{i,j} \to s_k$ can be applied and instead still rule $\mathbf{1.8}: \hat{a}_{s,t} \to \bar{a}_{s,t}$ is applied, the string can leave node 1 and enter node 2; if there rule **2.4** is applied, the resulting string has to leave node 2, but cannot go back to node 1 because of the symbol $\bar{\varepsilon}$ which is not allowed by the filter FI_1, yet because of the symbol $\bar{a}_{s,t}$ it also cannot enter node 4, hence, this string gets lost. Again, if rule **2.7** is applied, the resulting strings are lost, too.

On the other hand, if a rule $\mathbf{1.10}: \hat{a}_{i,j} \to s_k$ is applied, but rule $\mathbf{1.11'}: \hat{q}_0 \to \bar{\varepsilon}$ cannot yet be applied, then the resulting string $\bar{q}_l s_k w$ can leave node 1 and enter node 2. If **2.7** is applied, the resulting strings again are lost. If rule **2.8′** is applied immediately, the resulting string $\bar{q}_l q_k w$ cannot pass the input filters FI_1 or FI_4. If rule **2.5** is applied before rule **2.8′**, then the resulting string $\hat{q}_{l-1} q_k w$ cannot enter node 4, but can go back to node 1. If $k > 0$, then q_k prohibits the string to leave, hence, at some moment rule **1.1** has to be applied, which would require to send the resulting string to node 3, yet the symbol $\hat{q}_{l-1}$ does not allow the string to pass FI_3. Furthermore, if we apply a rule to $\hat{q}_{l-1}$ before, the resulting barred symbols keep the string away from node 3, but on the other hand, the symbol $\tilde{q}_k$ does not allow the string to enter node 2. Finally, if $k = 0$, then q_0 prohibits the string to leave, hence, at some moment rule **1.2′** has to be applied.

If $l - 1 > 0$, then we obtain the string $\bar{q}_{l-1} \bar{\varepsilon} w$, which can only be communicated to node 2, yet there the strings $\bar{q}_{l-1} \varepsilon w$ or $\hat{q}_{l-2} \varepsilon w$ resulting from the application of the rule **2.9′** or of the rules **2.5** and **2.9′** can enter neither node 1 nor node 4. If $k = l - 1 = 0$, the string $\bar{\varepsilon}\bar{\varepsilon} w$ resulting from the application of rules **1.2′** and **1.11′** can enter node 2, yet one application of rule **2.9′** already forces the resulting string to leave the node, but the second symbol $\bar{\varepsilon}$ keeps it away from both nodes 1 and 4.

To complete our arguments, we mention that again the (additional) application of rule **2.7** during any of the derivations described above cannot yield strings which would not get lost.

The correct simulation of the deletion operation ends with the following derivation steps in Γ:

$$(1, \bar{\varepsilon} s_k w) \vdash (2, \bar{\varepsilon} s_k w) \overset{\mathbf{2.9'}}{\Longrightarrow} (2, \varepsilon s_k w) \vdash (4, \varepsilon s_k w) \overset{\mathbf{4.1}}{\Longrightarrow}$$
$$(4, s_k w) \vdash (2, s_k w) \overset{\mathbf{2.8'}}{\Longrightarrow} (2, q_k w) \vdash (1, q_k w).$$

If rule **2.8′** is applied instead of rule **2.9′** in this derivation, the resulting string $\bar{\varepsilon} q_k w$ has to leave node 2, but cannot enter node 1 or node 4. If rule **2.7** is applied instead of rule **2.9′** or rule **2.8′** in this derivation, the resulting strings cannot enter node 1 or node 4, too.

The remaining arguments exhibited in the proof of Theorem 2 for the correctness of the construction for the simulation of a deletion rule are still valid for the construction elaborated in this proof, hence, in sum we conclude that Γ correctly simulates the application of rule $q_i a_j \to q_k$ in M.

In sum, we observe that also with the linear communication structure every computation of the CPM5 in normal form M can be simulated correctly by the [O]HNEP Γ, yielding exactly the same results, which observation completes the proof. □

As already argued previously, the following results can easily be obtained from Theorem 6:

Corollary 13 *Linear [O]HNEPs with* 5 *nodes are computationally complete.*

Corollary 14 *Any recursively enumerable language L can be accepted by a linear A[O]HNEP of size* 5.

Corollary 15 *Any recursively enumerable language L can be generated by a linear G[O]HNEP of size* 5.

In the same way as we have got Theorem 3 from Theorem 2, the following result can immediately be derived from Theorem 6:

Theorem 7 *Any (non-deterministic) CPM*5 *M in normal form can be simulated by a linear E[O]HNEP* Γ *of size* 4.

Proof Again we omit the fifth processor and designate the deletion node 4 as the new output node, now on the linear communication structure

$3 - 1 - 2 - 4.$

Moreover, again we replace the output filter $PO_4 = \emptyset$ of permitting contexts by $PO_4 = V \backslash T$. □

Corollary 16 *Linear E[O]HNEPs with* 4 *nodes are computationally complete.*

Corollary 17 *Any recursively enumerable language L can be accepted by a linear AE[O]HNEP of size* 4.

Corollary 18 *Any recursively enumerable language L can be generated by a linear GE[O]HNEP of size* 4.

7 Conclusions

We have shown that for hybrid networks of evolutionary processors (HNEPs) even computational completeness can already be obtained with only 5 nodes in case of complete or linear graphs being the communication structure between the processors of the network. Any partial recursive relation can be computed by a (complete or linear) HNEP with 5 nodes, and any recursively enumerable language can be accepted by a complete or linear AHNEP with 5 nodes or even be generated by a complete or linear GHNEP with only 5 nodes. The same results also hold true for the corresponding variants of obligatory HNEPs (OHNEPs).

When extracting the results of computations as the terminal strings appearing in the output node, we can even save one more node, i.e., computational completeness in all cases can already be achieved with only four nodes in the case of complete or linear extended [O]HNEPs. For extended [O]HNEPs having a star-like communication graph, all these results can be obtained with (at most) four nodes, too. For (non-extended) [O]HNEPs having a star-like communication graph, we needed six nodes.

Finally, we are pretty confident that our results with needing only five nodes for complete and linear [O]HNEPs and only four nodes for complete, linear, and star-like extended [O]HNEPs are already optimal: we anyway need one insertion and one deletion node; informally speaking, for any reasonable computations one additional substitution node is not sufficient, hence, we need at least two. As we have shown in this paper, one insertion and one deletion node plus two substitution nodes (plus one additional output node, which formally has to be a substitution node or a deletion node in the case of non-extended [O]HNEPs) are already sufficient. Therefore, reducing the number of processors needed for (non-extended) [O]HNEPs having a star-like communication graph to five is one of the main challenging problems for future research.

Acknowledgments The work of the first author and the fourth author was supported by Project STCU-5384 awarded by the Science and Technology Center in the Ukraine. The authors gratefully acknowledge the suggestions and comments of the two anonymous referees.

References

Alhazov A, Kudlek M, Rogozhin Yu (2002) Nine universal circular Post machines. Comput Sci J Moldova 10(3(30)):247–262

Alhazov A, Martín-Vide C, Rogozhin Yu (2006) On the number of nodes in universal networks of evolutionary processors. Acta Inform 43(5):331–339

Alhazov A, Martín-Vide C, Rogozhin Yu (2007) Networks of evolutionary processors with two nodes are unpredictable. In: Loos R, Fazekas SZ, Martín-Vide C (eds) LATA 2007. Proceedings of the 1st international conference on language and automata theory and applications, Report, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, vol 35/07, pp 521–528

Alhazov A, Csuhaj-Varjú E, Martín-Vide C, Rogozhin Yu (2008a) About universal hybrid networks of evolutionary processors of small size. In: Martín-Vide C, Otto F, Fernau H (eds) Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain. Revised papers, Lecture Notes in Computer Science, Springer, Berlin, vol 5196, pp 28–39, 13–19 March 2008

Alhazov A, Csuhaj-Varjú E, Martín-Vide C, Rogozhin Yu (2008) Computational completeness of hybrid networks of evolutionary processors with seven nodes. In: Câmpeanu C, Pighizzini G (eds) 10th international workshop on descriptional complexity of formal systems, DCFS 2008, Charlottetown, Prince Edward Island, Canada, University of Prince Edward Island, 16–18 July 2008, pp 38–47

Alhazov A, Csuhaj-Varjú E, Martín-Vide C, Rogozhin Yu (2009a) On the size of computationally complete hybrid networks of evolutionary processors. Theoret Comput Sci 410(35):3188–3197

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

Alhazov A, Bel-Enguix G, Rogozhin Yu (2011a) About a new variant of HNEPs: obligatory hybrid networks of evolutionary processors. In: Bel-Enguix G, Jiménez-López MD (eds) Bio-inspired

models for natural and formal languages. Cambridge Scholars Publishing, Cambridge, pp 191–204

Alhazov A, Krassovitskiy A, Rogozhin Yu (2011b) Circular Post machines and P systems with exo-insertion and deletion. In: Gheorghe M, Păun Gh, Rozenberg G, Salomaa A, Verlan S (eds) Membrane computing—12th international conference, CMC 2011, Fontainebleau, France, 23–26 August 2011, Revised Selected Papers, Lecture Notes in Computer Science, Springer, Berlin, vol 7184, pp 73–86

Alhazov A, Bel-Enguix G, Rogozhin Yu (2014a) Smallest filters in complete obligatory hybrid networks of evolutionary processors. J Autom Lang Comb 19(1–4):5–16

Alhazov A, Freund R, Rogozhin Yu (2014b) Five nodes are sufficient for hybrid networks of evolutionary processors to be computationally complete. In: Ibarra OH, Kari L, Kopecki S (eds) Unconventional computation and natural computation—13th international conference, UCNC 2014, London, ON, Canada, 14–18 July 2014, Proceedings, Lecture Notes in Computer Science, Springer, Berlin, vol 8553, pp 1–13

Castellanos J, Martín-Vide C, Mitrana V, Sempere JM (2001) Solving NP-complete problems with networks of evolutionary processors. In: Mira J, Prieto A (eds) Connectionist models of neurons, learning processes and artificial intelligence, 6th international work-conference on artificial and natural neural networks, IWANN 2001 Granada, Spain, 13–15 June 2001, Proceedings, Part I, Lecture Notes in Computer Science. Springer, Berlin, vol 2084, pp 621–628

Csuhaj-Varjú E, Martín-Vide C, Mitrana V (2005) Hybrid networks of evolutionary processors are computationally complete. Acta Inform 41(4–5):257–272

Dassow J, Manea F (2010) Accepting hybrid networks of evolutionary processors with special topologies and small communication. In: McQuillan I, Pighizzini G (eds) Proceedings twelfth annual workshop on descriptional complexity of formal systems, DCFS 2010, Saskatoon, Canada, 8–10th August 2010, EPTCS, vol 31, pp 68–77

Dassow J, Manea F, Truthe B (2015) On the power of accepting networks of evolutionary processors with special topologies and random context filters. Fundam Inform 136(1–2):1–35

Freund R, Rogozhin Yu, Verlan S (2014) Generating and accepting P systems with minimal left and right insertion and deletion. Nat Comput 13(2):257–268

Kudlek M, Rogozhin Yu (2001a) New small universal circular post machines. In: Freivalds R (ed) Fundamentals of computation theory, 13th international symposium, FCT 2001, Riga, Latvia, Proceedings, Lecture Notes in Computer Science, Springer, Berlin. vol 2138, pp 217–226, 22–24 August 2001

Kudlek M, Rogozhin Yu (2001b) Small universal circular post machines. Comput Sci J Moldova 9(1(25)):34–52

Loos R, Manea F, Mitrana V (2010) Small universal accepting hybrid networks of evolutionary processors. Acta Inform 47(2):133–146

Manea F, Martín-Vide C, Mitrana V (2007) On the size complexity of universal accepting hybrid networks of evolutionary processors. Math Struct Comput Sci 17(4):753–771

Manea F, Mitrana V (2007) All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. Inf Process Lett 103(3):112–118

Margenstern M, Mitrana V, Pérez-Jiménez MJ (2005) Accepting hybrid networks of evolutionary processors. In: Ferretti C, Mauri G, Zandron C (eds) DNA computing, 10th international workshop on DNA computing, DNA 10, Milan, Italy, Revised selected papers, Lecture Notes in Computer Science, Springer, Berlin, vol 3384, pp 235–246, 7–10 June 2004

Martín-Vide C, Mitrana V, Pérez-Jiménez MJ, Sancho-Caparrini F (2003) Hybrid networks of evolutionary processors. In: Cantú-Paz E, Foster JA, Deb K, Davis L, Roy R, O'Reilly U, Beyer H, Standish RK, Kendall G, Wilson SW, Harman M, Wegener J, Dasgupta D, Potter MA, Schultz AC, Dowsland KA, Jonoska N, Miller JF (eds) Genetic and evolutionary computation—GECCO 2003, Genetic and evolutionary computation conference, Chicago, IL, USA. Proceedings, Part I, Lecture Notes in Computer Science, Springer, Berlin, vol 2723, pp 401–412, 12–16 July 2003

Post EL (1943) Formal reductions of the general combinatorial decision problem. Am J Math 65(2):197–215

Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages, vols 1–3. Springer, Berlin

Salomaa A (1973) Formal languages. Academic Press, New York

Truthe B (2013) Computationally complete chains of evolutionary processors with random context filters. In: Freund R (ed) Fifth workshop on non-classical models of automata and applications (NCMA 2013), books@ocg.at, vol 294, pp 225–241