ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **12**, Number 2, 2009, 219–233

Complexity Aspects of the Recognition of Regular and Context-Free Languages by Accepting Hybrid Networks of Evolutionary Processors

Peter LEUPOLD[1], Remco LOOS[2], Florin MANEA[3]

[1] Department of Mathematics, Faculty of Science, Kyoto Sangyo University
Kyoto 603-8555, Japan
E-mail: leupold@cc.kyoto-su.ac.jp

[2] EMBL - European Bioinformatics Institute
Wellcome Trust Genome Campus, Hinxton, Cambridge, UK, CB10 1SD
E-mail: remco.loos@ebi.ac.uk

[3] Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, Bucharest, Romania, 010014
E-mail: flmanea@gmail.com

Abstract. In this paper we address the size complexity of Accepting Hybrid Networks of Evolutionary Processors (AHNEPs) that recognize regular and context-free languages. We present AHNEPs of small constant size for both classes. We show that any regular language can be accepted by an AHNEP of size 6, while AHNEPs of size 9 suffice for all context-free languages, moreover accepting them in linear time. Both bounds constitute significant improvements of the best known upper bounds.

1. Introduction

An interesting question for any type of computational model is how concisely it can represent given formal languages. This is the topic of the field of descriptional complexity. We present results from this area for Hybrid Networks of Evolutionary Processors (HNEPs for short, [1]), that is we try to find the least complex instance

of this model that can describe a given language. As for most of the computational models defined so far, there are different ways in which the complexity of an HNEP can be measured. In this paper we focus mainly on size complexity (i.e. the number of processors in the network), though we will also consider time complexity, i.e. the number of steps performed by the network during the computation on an input word).

In the case of Accepting Hybrid Networks of Evolutionary Processors (AHNEPs), the results obtained so far regard the size of networks accepting recursively enumerable ([5, 7]), context-free and regular languages ([9]). In this paper we present improved bounds for the latter two classes.

First, we propose an AHNEP of size 6 for the acceptance of a regular language. This also allows us to introduce the techniques we will then use to show that AHNEPs of size 9 can accept all context-free languages, by simulating the push-down automata recognizing them. We obtain these constant bounds, independent of the specific language, by direct construction rather than through an encoding into a 2-letter alphabet, as in [5].

Interestingly, the constructed networks do not only provide a concise description of the given languages, but are also computationally efficient. From [6, 5, 8] we know that $NP = PTIME_{AHNEP}$, thus for every language L in the NP complexity class, decided by a non-deterministic one-tape Turing machine in time $P(n)$ for some polynomial P, there exists an AHNEP that decides L in $\mathcal{O}(P(n))$ steps. So far, this was also the best known bound for context-free languages. We stress that the AHNEPs in our constructions need only a linear amount of steps to accept a word of their accepted languages. Thus they are not only small in size, but they are also computationally efficient.

2. Preliminaries

In this section we define the most important notions used throughout this paper. For a more detailed presentation we refer to [2]. Also, we mention that for the general definitions, notations and results concerning push-down automata, finite automata, context-free languages and regular languages we refer to [4]. We only briefly recall that for every context-free language L there exists a non-deterministic push-down automaton, accepting with final states, and without λ-transitions, recognizing L.

An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called *string (word)* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by λ. The length of a string x is denoted by $|x|$, while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\lambda\}$ is a *substitution rule* if both a and b are not λ; it is a *deletion rule* if $a \neq \lambda$ and $b = \lambda$; it is an *insertion rule* if $a = \lambda$ and $b \neq \lambda$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule as above σ and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$
- If $\sigma \equiv a \to \lambda \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : \ w = ua\}, \\ \{w\}, \text{ otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v : \ w = av\}, \\ \{w\}, \text{ otherwise} \end{cases}$$

- If $\sigma \equiv \lambda \to a \in Ins_V$, then

$$\sigma^*(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}, \ \sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}.$$

The parameter $\alpha \in \{*, l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

$$M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w) \ \text{ and } \ M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),$$

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as language theoretical formulations of local gene mutations. Given two disjoint and nonempty subsets P and F of an alphabet V and a word over V, the following predicates are defined:

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

In these predicates the two sets P (*permitting contexts*) and F (*forbidding contexts*) define the networks' *random-context conditions*.

For every language $L \subseteq V^*$ and $\beta \in \{(1), (2)\}$, we define:

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

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

- $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only.
- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor.

We denote the set of evolutionary processors over V by EP_V. An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-tuple $\Pi = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

- V and U are the input and network alphabet, respectively, and $V \subseteq U$.

- $G = (X_G, E_G)$ is an undirected graph with the set of nodes X_G and the set of edges E_G. G is called the *underlying graph* of the network.
- $N : X_G \longrightarrow EP_U$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $N(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $\beta : X_G \longrightarrow \{(1), (2)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, the following filters are defined:

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

  That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the string w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of strings of L that can pass the input (resp. output) filter of x.
- x_I and $x_O \in X_G$ is the *input node*, and the *output node*, respectively, of the AHNEP.

We say that $card(X_G)$ is the size of Π, and we denote this by $size(\Pi)$. If $\alpha(x) = \alpha(y)$ and $\beta(x) = \beta(y)$ for any pair of nodes $x, y \in X_G$, then the network is said to be *homogeneous*. In the theory of networks some types of underlying graphs are common, e.g., *rings, stars, grids* etc. Networks of evolutionary processors with underlying graphs having these special forms have been considered in a series of papers [1, 2, 10, 3]. We focus here on *complete* AHNEPs, i.e. AHNEPs whose underlying graph is a complete one. For n nodes this graph is denoted by K_n; it has edges between every pair of nodes including loops from a node to itself.

A *configuration* of an AHNEP Π as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of strings with every node of the graph. These sets consist of those strings which are present in the respective node at a given moment. A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

$$C'(x) = M_x^{\alpha(x)}(C(x)) \text{ for all } x \in X_G.$$

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each string it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the strings sent by any node processor connected with x providing that they can pass its input filter.

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

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

Let Π be an AHNEP. The computation of Π on the input string $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Π defined by $C_0^{(w)}(x_I) = w$, where $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G$ if $x \neq x_I$, and where $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$; thus, each computation in an AHNEP is deterministic. A computation as above immediately halts if one of the following two conditions holds:

(i) There exists a configuration in which the set of strings existing in the output node x_O is non-empty. In this case, the computation is said to be an *accepting computation*.

(ii) The configurations obtained after two consecutive evolutionary or communication steps are identical.

In the aforementioned cases the computation is said to be finite. The language accepted by Π is

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

We say that an AHNEP Π decides the language $L \subseteq V^*$ iff $L(\Pi) = L$ and the computation of Π on every $x \in V^*$ halts.

We also define the *time complexity* of the finite computation $C_0^{(x)}$, $C_1^{(x)}$, $C_2^{(x)}$, $\ldots C_m^{(x)}$ of Π on $x \in V^*$ is denoted by $Time_\Pi(x)$ and equals m. The time complexity of Π is the partial function from $\mathbf{N}$ to $\mathbf{N}$,

$$Time_\Pi(n) = \max\{Time_\Pi(x) \mid x \in V^*, |x| = n\}.$$

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\begin{aligned} \mathbf{Time}_{AHNEP}(f(n)) \quad = \quad & \{L \mid \text{ there exists an AHNEP } \Pi \text{ which decides } L \\ & \text{and } n_0 \in N \text{ such that } \forall n \geq n_0 (Time_\Pi(n) \leq f(n))\}. \end{aligned}$$

Moreover, we write

$$\mathbf{PTime}_{AHNEP} = \bigcup_{k \geq 0} \mathbf{Time}_{AHNEP}(n^k).$$

Further we present already known results regarding AHNEPs that accept context-free languages. Since each context-free language is a recursively enumerable language, one can construct an AHNEP that simulates a Turing Machine accepting that language. This approach leads to the following theorem:

Theorem 1. *For any recursively enumerable language L, recognized by a one-tape Turing Machine $M = (Q, V_1, V_2, \delta, q_0, B, F)$, there exists an AHNEP Π of size 24 accepting L. Also, if M makes $f(|w|)$ steps on the acceptance of w then Π makes $\mathcal{O}(f(|w|))$ steps on the acceptance of w.*

Another way to obtain an AHNEP architecture for the recognition of context-free languages is to simulate the computation of a non-deterministic push-down automaton accepting the given context-free language. The result was as follows.

Proposition 1. *If L is a context-free language and $\Gamma = (Q, V, \Sigma, q_0, Z_0, \delta)$ is a non-deterministic push-down automaton, accepting with empty stack, without λ-transitions, such that $L(\Gamma) = L$, then there exists an AHNEP Π such that $size(\Pi) = 3|Q| + 2|\Sigma| + 5$ and $L(\Pi) = L$.*

In contrast to Theorem 1, this results in AHNEPs of variable size. Depending on the push-down automaton's and the alphabet's size, either Theorem 1 or Proposition 1 can provide the smaller AHNEP for a given context-free language. Since the implementation of a push-down store on a Turing tape will require additional states and/or alphabet symbols, Proposition 1's bound will be better in most cases.

4. Accepting Regular Languages

The main goal of the investigations presented here is to improve the descriptional complexity bounds for the acceptance of context-free languages as given by Proposition 1 and also by Theorem 1. First, however, we will take a small detour via regular languages. For these we can present a class of very small AHNEPs. The way in which they simulate a finite automaton uses the same mechanisms that we will use later on to simulate push-down automata. Therefore these AHNEPs also offer an opportunity to understand these techniques in an environment that is less complex and easier to oversee.

This said we now proceed to present a class of AHNEPs over complete graphs with six nodes that can accept every regular language.

Theorem 2. *For every regular language L there exists an AHNEP over the complete graph with six nodes that accepts L.*

Proof. We look at the alphabet as an ordered set $V = \{a_1, a_2, \ldots, a_k\}$ with k elements. For a given regular language L let $A = (Q, V, \delta, q_0, F)$ be a deterministic finite automaton that accepts this language; Q is the set of states, $\delta : (Q \times V) \to Q$ is the transition function, q_0 is the start state, and $F \subseteq Q$ the set of final states. The functioning of a finite automaton is considered as common knowledge here.

We now construct an AHNEP $\Pi = (V, U, G, \mathcal{N}, \alpha, \beta, Ini, Fin)$ that accepts L based on the deterministic finite automaton for this language. The network alphabet U will not be defined explicitly, as it consists of all the symbols used below in the processors' definition. Rather we describe the meaning of a class of new symbols we introduce: a symbol $[q, a, q']$ shall mean that the automaton is now in state q and will go to state q' by reading a. Thus these symbols store the complete information of a transition of the automaton. The simulation of the automaton's operation will then consist of

- introducing the symbol for a transition starting from the initial state,

- checking whether the first letter of the input word is the same as read in that transition,
- deleting that letter and replacing the transition's symbol by the one of a possible following transition.

The main technical problem here is matching the input word's letters with the ones in the symbols corresponding to transitions. Because none of the rules have left hand sides of length two or longer, it is not possible to let one rule application verify their equality. Rather, this is done by first marking both letters and then decreasing them simultaneously according to the order of the alphabet. If they matched in the beginning, then they will reach a_1 in the same cycle, and only then the computation should proceed. The processors that are used to do this are the following:

• Node Ini (the network's input node):
– $M(Ini) = \{\lambda \to [q_0, a, q] \mid \delta(q_0, a) = q\} \cup \{\lambda \to F \mid \text{if } q_0 \text{ is final, i.e. } \lambda \in L\}$,
– $\alpha(Ini) = r$, $\beta(Ini) = (2)$,
– $PI(Ini) = \emptyset$, $FI(Ini) = U$,
– $PO(Ini) = \{[q_0, a, q] \mid \delta(q_0, a) = q\}$, $FO(Ini) = \emptyset$.

• Node $Mark$:
– $M(Mark) = \{a \to a' \mid a \in V\}$,
– $\alpha(Mark) = *$, $\beta(Mark) = (2)$,
– $PI(Mark) = PO(Ini)$, $FI(Mark) = \{a' \mid a \in V\} \cup \{F, A_1\}$,
– $PO(Mark) = \{a' \mid a \in V\}$, $FO(Mark) = \{a'' \mid a \in V\}$.

• Node Dec:
– $M(Dec) = \{a'_1 \to a''_1, [q, a_1, q'] \to [q, a'_1, q'] \mid q, q' \in Q\} \cup$
$\cup\{a'_i \to a''_{i-1}, [q, a_i, q'] \to [q, a'_{i-1}, q'] \mid 1 < i \leq k, q, q' \in Q\}$,
– $\alpha(Dec) = *$, $\beta(Dec) = (2)$,
– $PI(Dec) = PO(Mark)$, $FI(Dec) = \{F\}$,
– $PO(Dec) = \{a'' \mid a \in V\}$, $FO(Dec) = \{[q, a, q'] \mid q, q' \in Q, a \in V\}$.

• Node $Trans$:
– $M(Trans) = \{a''_i \to a'_i \mid 1 < i < k\} \cup$
$\cup\{[q, a'_i, q'] \to [q, a_i, q'] \mid 1 < i \leq k, q, q' \in Q\} \cup$
$\cup\{[q, a'_1, q'] \to [q', a, q''] \mid \delta(q', a) = q''\} \cup \{a''_1 \to A_1\} \cup$
$\cup\{[q, a'_1, q'] \to F \mid q \in Q, q' \in F\}$,
– $\alpha(Trans) = *$, $\beta(Trans) = (2)$,
– $PI(Trans) = PO(Dec)$, $FI(Trans) = \{F\}$,
– $PO(Trans) = \{A_1\} \cup \{a' \mid a \in V\}$,
$FO(Trans) = \{a'' \mid a \in V\} \cup \{[q, a', q'] \mid a \in V, q, q' \in Q\}$.

• Node Del:
– $M(Del) = \{A_1 \to \lambda\}$,
– $\alpha(Del) = l$, $\beta(Del) = (2)$,

– $PI(Del) = \{A_1\}$, $FI(Del) = \{[q, a'_i, q'] \mid 1 < i \leq k, q, q' \in Q\}$,
– $PO(Del) = \emptyset$, $FO(Del) = \{A_1\}$.

• Node Fin (the network's output node):
– $PI(Fin) = \{F\}$, $FI(Fin) = U \setminus \{F\}$.

We will not prove the correctness of the construction in all detail. Rather we will trace a possible input string and argue that it can produce the string F in node $Trans$ if and only if it is accepted by the original automaton. From node Ini the input string can exit only after a symbol $[q_0, a, q]$ is added on its right side. Afterwards no string can enter node Ini anymore; therefore we do not need to consider it anymore. Similarly, no string can exit node Fin, and only strings from $\{F\}^+$ can enter this node; therefore we will leave it aside for the time being. Finally also the node Del will be left aside for now, because its input filters let pass only strings containing A_1, which will not occur during the first steps.

The resulting string from node Ini cannot enter node $Trans$, because none of the symbols required by $PI(Trans)$ is present, similar for node Des. So it only enters node $Mark$. There one letter must be marked before it can exit, the resulting string enters only in node $Trans$. Note that only marking of the first letter lets us simulate a transition reading it; this is ensured by the fact that marking of a different letter will not let this mark be deleted in node Del, which works only on the left-most symbol. From node Dec the string can exit only after two rules have been applied. Either the initial letter has received a second prime (which fulfills $PO(Dec)$), and also the letter a in the symbol $[q, a, q']$ has been changed (to pass $FO(Dec)$). Both letters are decreased one step in the alphabet's order and receive another prime. Since both steps have to be taken, the two letters can be decreased in a synchronized manner in this way. The second case is where the letter is already a_1, then it is simply marked without further decreasing it.

The exiting string can only enter node $Trans$. There different things can happen. If the marked symbol is not a_1, then the second prime and the mark in the symbol of type $[q, a, q']$ are removed. $FO(Trans)$ ensures that both things are done before the string can exit. If the marked symbol is a_1, then it is changed to A_1 and the symbol of type $[q, a, q']$ is replaced by one representing a possible next transition. The format of the resulting string differs in the format of the first letter. If it does not contain A_1 it can go directly to node Dec to further decrease the marked symbol.

If the string contains A_1, then the only node that can receive this string is Del, and there A_1 is deleted. As stated before, note that the deleting rule works only on the left-most symbol. If the symbol appears in any other position, the resulting word cannot pass $FO(Dec)$, and remains trapped in this node. After the deletion of A_1, the string can enter node $Mark$ to mark the (new) first letter and start the simulation of the next transition. Strings obtained as in the first case described above for node $Trans$ already carry a letter with one prime and cannot enter node $Mark$ because of $FI(Mark)$.

It is also important to note that the derivation can only continue if the a in the appended symbol $[q, a, q']$ is the same as the first symbol of the word. Indeed, if

we have a string of the form $a_t x[q, a_k, q']$ with $t \neq k$ entering in node $Mark$, the decreasing process will take place as described, until arriving at a string $A_1 x[q, a_j, q']$, with $j > 1$, if $t < k$, or a string $a_j x[q, a_1, q']$, $j > 1$ otherwise, in node $Trans$. Both strings can leave the node, but cannot pass any input filter, so they are lost. In this way, only correct simulations can lead to an accepting computation.

If the simulated transition results in an accepting state of the automaton, then the symbol of type $[q, a, q']$ can also be rewritten to F in node $Trans$. The resulting string can only enter node Del. From there it can only go to node Fin after the deletion of A_1, and this only if no other letter is left. This means that the entire input has been read when the final state is reached, which is exactly the acceptance condition of the finite automaton. □

We illustrate the construction used in the proof by a small example.

Example 1. We start out from the finite automaton $A = (\{q_0, q_b, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$, where $\delta(q_0, a) = q_b$. $\delta(q_b, a) = q_f$, $\delta(q_b, b) = q_b$, and δ is undefined for all other parameters. This automaton accepts the language ab^+a. For the alphabet we assume the order $a < b$, this means a is a_1 and b is a_2 following the nomenclature in the proof. Let Π_A be the AHNEP constructed from A using the technique presented in Theorem 2.

We refrain from reproducing the mechanical exercise of constructing the resulting AHNEP Π_A. Rather, we present an example of the computation of Π_A on a given input to show how it simulates the moves of A.

Figure 1 shows the accepting computation for the input word *abba*. The lines represent the configurations, i.e. the current contents of the nodes in one column each. $\Downarrow$ stands for a computation step, i.e. for the application of a rule. $\hookrightarrow$ and $\hookleftarrow$ stand for communication steps, where the unique non-empty string present in the network enters the node whose column contains this arrow. Recall from the argumentation in the proof above that at any given time exactly one string will be present in the network. Further, we omit the nodes Ini and Fin, because they will not contain any string during the phase depicted in the table.

The first step is the application of the rule $\lambda \to [q_0, a, q_b]$ in the node Ini. The resulting string $abba[q_0, a, q_b]$ is then communicated to $Mark$, it cannot pass the input filters of any other node. Figure 1 follows the steps in the computation from this point, until a string F is reached. This string will now be communicated to node Fin and thus the computation ends with the acceptance of *abba*.

The minimal size for an AHNEP is two, because at least input and output node need to be present. With these alone, however, one cannot really perform any significant computation. At the very least one more node will be necessary. Intuitively, it also seems clear that one node will not yield much computational power, since one node alone cannot make use of the control that input and output filters provide. It cannot use the different mechanisms of rewriting on just one side or anywhere, either. Therefore the bound of six nodes must be very close to the optimum.

Mark	Dec	Trans	Del
$abba[q_0, a, q_b]$	–	–	–
$\Downarrow$			
$a'bba[q_0, a, q_b]$	$\hookrightarrow$	–	–
	$\Downarrow$		
–	$a''bba[q_0, a, q_b]$	–	–
	$\Downarrow$		
–	$a''bba[q_0, a', q_b]$	$\hookrightarrow$	–
		$\Downarrow$	
–	–	$A_1bba[q_0, a', q_b]$	–
		$\Downarrow$	
–	–	$A_1bba[q_b, b, q_b]$	$\hookrightarrow$
			$\Downarrow$
$\hookleftarrow$	–	–	$bba[q_b, b, q_b]$
$\Downarrow$			
$b'ba[q_b, b, q_b]$	–	–	–
$\vdots$			
$b'a[q_b, b, q_b]$	–	–	–
$\vdots$			
$a'[q_b, a, q_f]$	–	–	–
	$\ddots$		
–	–	$[q_b, a', q_f]$	–
		$\Downarrow$	
–	–	F	–

Fig. 1. The accepting computation of AHNEP Π_A on input $abba$.

5. Accepting Context-free Languages

In this section we extend our approach to the design of an AHNEP accepting a context-free language, effectively described by the push-down automaton that accepts it.

Let L be a context-free language, and $\Gamma = (Q,\ F,\ V,\ \Sigma,\ q_0,\ Z_0,\ \delta)$ be a nondeterministic push-down automaton, accepting with final states, and without λ-transitions, recognizing this language. In this paper we use the following convention: if $(q, \alpha) \in \delta(q', a, Z)$ is a transition of the push-down automaton Γ, we assume that the rightmost symbol of α is placed highest on the stack, while the leftmost symbol of α is placed lowest on the stack. Moreover, we can assume without loss of generality that if $(q, \alpha) \in \delta(q', a, Z)$ then α does not contain the symbol Z_0. Finally, let $K_\Gamma = 1 + \max\{|\alpha| \mid (q', \alpha) \in \delta(q, a, Z), \forall a \in V, q \in Q, Z \in \Sigma\}$.

Theorem 3. *There exists an AHNEP* $\Pi = (V, U, G, \mathcal{N}, \alpha, \beta, 1, 9)$ *such that* $L(\Pi) = L$*, and* G *is the complete graph with* 9 *nodes.*

Proof. As before we look upon the input and stack alphabet as ordered sets, so

that $V = \{a_1, \ldots, a_n\}$ and $\Sigma = \{Z_0, Z_1, \ldots, Z_m\}$. We define the working alphabet of Π as follows. Let $U = V \cup \Sigma \cup \{\$, \$^\bullet, \#, \#^\bullet\} \cup \{a', a^\bullet \mid a \in V\} \cup \{Z', Z^\bullet \mid a \in V\}$ $\cup$ $\{[q_1, a, Z, q_2, \alpha],\ [q_1, \$, Z, q_2, \alpha],\ [q_1, \bar{a}, Z, q_2, \alpha]',\ [q_1, \bar{a}, Z, q_2, \alpha]^\bullet,\ [q_1, \$, \overline{Z}, q_2, \alpha]',$ $[q_1, \$, \overline{Z}, q_2, \alpha]^\bullet,\ [q_1, \$, \#, q_2, \alpha],\ [q_1, \$, \#, q_2, \alpha]^\diamond,\ [q_1, \$, \#, q_2, \alpha]^\circ \mid q_1, q_2 \in Q,\ a \in V,$ $Z \in \Sigma,\ \alpha \in \Sigma^*,\ |\alpha| \leq K_\Gamma\}$.

The processors placed in the 9 nodes of the network are defined as follows:

• Node 1 (the input node of the network):
– $M(1) = \{\lambda \to [q_0, a, Z_0, q', \alpha] \mid q' \in Q, a \in V, \alpha \in \Sigma^*, (q', \alpha) \in \delta(q_0, a, Z_0)\}$,
– $\alpha(1) = r$, $\beta(1) = (2)$
– $PI(1) = \emptyset$, $FI(1) = U$,
– $PO(1) = U$, $FO(1) = \emptyset$.

• Node 2:
– $M(2) = \{\lambda \to Z_0\}$,
– $\alpha(2) = r$, $\beta(2) = (2)$
– $PI(2) = \{[q_0, a, Z_0, q', \alpha] \mid q' \in Q, a \in V, \alpha \in \Sigma^*, (q', \alpha) \in \delta(q_0, a, Z_0)\}$,
$FI(2) = U \setminus (PI(2) \cup V)$.
– $PO(2) = U$, $FO(2) = \emptyset$,

• Node 3:
– $M(3) = \{[q, a, Z, q', \alpha] \to [q, \bar{a}, Z, q', \alpha]',\ [q, \$, Z, q', \alpha] \to [q, \$, \overline{Z}, q', \alpha]',\ [q, \$, \#, q',$ $Z_0\alpha] \to [q, \$, \#, q', \alpha]^\circ \mid q, q' \in Q, a \in V, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup \{\diamond \to \circ\} \cup$
$\cup\{[q, \$, \#, q', \alpha] \to [q, \$, \#, q', \alpha]^\circ \mid q, q' \in Q, a \in V, Z \in \Sigma, \alpha \in \Sigma^*, 0 < |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \#, q', \lambda] \to [q', \bar{a}, Z, q'', \alpha]' \mid (q'', \alpha) \in \delta(q', a, Z)\} \cup \{Z^\circ \to Z \mid Z \in \Sigma\}$,
– $\alpha(3) = *$, $\beta(3) = (2)$
– $PI(3) = \{[q, a, Z, q', \alpha],\ [q, \$, Z, q', \alpha] \mid q, q' \in Q, a \in V, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma,$ such that α does not contain $Z_0\} \cup \{[q, \$, \#, q', Z_0]\} \cup \{\diamond\}$, $FI(3) = \{\$, \#\}$.
– $PO(3) = U$, $FO(3) = PI(3) \cup \{Z^\diamond \mid Z \in \Sigma\} \cup \{[q, \$, \#, q', Z_0\alpha] \mid q' \in Q, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\}$,

• Node 4:
– $M(4) =$
$\{[q, \bar{a}_i, Z, q', \alpha]' \to [q, \bar{a}_{i-1}, Z, q', \alpha]^\bullet \mid q, q' \in Q, 1 < i \leq k, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\{[q, \bar{a}_1, Z, q', \alpha]' \to [q, \$, Z, q', \alpha]^\bullet \mid q, q' \in Q, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\{[q, \$, \bar{Z}_i, q', \alpha]' \to [q, \$, \bar{Z}_{i-1}, q', \alpha]^\bullet \mid q, q' \in Q, 0 < i \leq m, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\{[q, \$, \bar{Z}_0, q', \alpha]' \to [q, \$, \#, q', \alpha]^\bullet \mid q, q' \in Q, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\{[q, \$, \#, q', Z_i\alpha]^\circ \to [q, \$, \#, q', Z_{i-1}\alpha]^\diamond \mid q, q' \in Q, 0 < i \leq m, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\{\circ \to Z_1^\diamond\} \cup \{Z_i^\circ \to Z_{i+1}^\diamond \mid 1 \leq i \leq m-1\} \cup$
$\{a_i \to a_{i-1}^\bullet, a_i' \to a_{i-1}^\bullet \mid 1 < i \leq k\} \cup \{a_1 \to \$^\bullet, a_1' \to \$^\bullet\} \cup$
$\{Z_i \to Z_{i-1}^\bullet, Z_i' \to Z_{i-1}^\bullet \mid 1 \leq i \leq k\} \cup \{Z_0 \to \#^\bullet, Z_0' \to \#^\bullet\}$
– $\alpha(4) = *$, $\beta(4) = (2)$
– $PI(4) = \{[q, \bar{a}, Z, q', \alpha]',\ [q, \$, \overline{Z}, q', \alpha]',\ [q, \$, \#, q', \alpha]^\circ \mid q, q' \in Q, 1 < i \leq k, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup \{\dagger\} \cup \{a' \mid a \in V\} \cup \{Z', Z^\circ \mid Z \in \Sigma\}$, $FI(4) = U \setminus (V \cup \Sigma \cup PI(4) \cup)$.

– $PO(4) = \{a^\bullet \mid a \in V\} \cup \{Z^\bullet, Z^\diamond \mid Z \in \Sigma\} \cup \{\$^\bullet, \#^\bullet\}$, $FO(4) = PI(4)$,

• Node 5:
– $M(5) =$
$\{[q, \bar{a}_i, Z, q', \alpha]^\bullet \to [q, \bar{a}_i, Z, q', \alpha]' \mid q, q' \in Q, 1 \le i \le k, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, Z, q', \alpha]^\bullet \to [q, \$, Z, q', \alpha] \mid q, q' \in Q, Z \in \Sigma, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \overline{Z_i}, q', \alpha]^\bullet \to [q, \$, \overline{Z_i}, q', \alpha]' \mid q, q' \in Q, 0 \le i \le m, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \#, q', \alpha]^\bullet \to [q, \$, \#, q', Z_0\alpha] \mid q, q' \in Q, \alpha \in \Sigma^*, 0 < |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \#, q', \lambda]^\bullet \to [q, \$, \#, q', \lambda] \mid q, q' \in Q\} \cup$
$\cup\{[q, \$, \#, q', Z_0\alpha]^\diamond \to [q, \$, \#, q', Z_0\alpha] \mid q, q' \in Q, \alpha \in \Sigma^*, 0 < |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \#, q', Z_i\alpha]^\diamond \to [q, \$, \#, q', Z_i\alpha]^\circ \mid q, q' \in Q, 0 < i \le m, \alpha \in \Sigma^*, |\alpha| < K_\Gamma\} \cup$
$\cup\{[q, \$, \#, q', Z_0]^\diamond \to [q', a, Z, q'', \alpha] \mid q, q', q'' \in Q, a \in V, Z \in \Sigma, \alpha \in \Sigma^*, (q'', \alpha) \in \delta(q', a, Z)\} \cup$
$\cup\{Z_i^\diamond \to Z_i^\circ \mid 1 \le i \le m\} \cup \{a_i^\bullet \to a_i' \mid 1 \le i \le k\} \cup \{Z_i^\bullet \to Z_i' \mid 0 \le i \le k\} \cup$
$\cup\{\#^\bullet \to \#\} \cup \{\$^\bullet \to \$\}$
– $\alpha(5) = *$, $\beta(5) = (2)$
– $PI(5) = \{\$^\bullet, \#^\bullet\} \cup \{a^\bullet \mid a \in V\} \cup \{Z^\bullet, Z^\diamond \mid Z \in \Sigma\} \cup \{[q_1, \bar{a}, Z, q_2, \alpha]^\bullet, [q_1, \$, Z, q_2, \alpha]^\bullet, [q_1, \$, \overline{Z}, q_2, \alpha]^\bullet, [q_1, \$, \#, q_2, \alpha]^\bullet, [q_1, \$, \#, q_2, \alpha]^\diamond \mid q_1, q_2 \in Q,\ a \in V,\ Z \in \Sigma,\ \alpha \in \Sigma^*, |\alpha| \le K_\Gamma\}$, $FI(5) = U \setminus (V \cup \Sigma \cup PI(5))$.
– $PO(5) = U$, $FO(5) = PI(5)$,

• Node 6:
– $M(6) = \{\$ \to \lambda\}$
– $\alpha(6) = l$, $\beta(6) = (2)$
– $PI(6) = \{\$\}$, $FI(6) = U \setminus (V \cup \Sigma \cup \{\$\} \cup \{[q, \$, Z, q', \alpha]\} \mid q, q' \in Q,\ Z \in \Sigma,\ \alpha \in \Sigma^*, |\alpha| < K_\Gamma\})$.
– $PO(6) = U$, $FO(6) = \emptyset$,

• Node 7:
– $M(7) = \{\# \to \lambda\}$
– $\alpha(7) = r$, $\beta(7) = (2)$
– $PI(7) = \{\#\}$, $FI(7) = U \setminus (V \cup \Sigma \cup \{\#\} \cup \{[q, \$, \#, q', \alpha]\} \mid q, q' \in Q,\ \alpha \in \Sigma^*, |\alpha| < K_\Gamma\})$.
– $PO(7) = U$, $FO(7) = \emptyset$,

• Node 8:
– $M(8) = \{\lambda \to \diamond\}$
– $\alpha(8) = r$, $\beta(8) = (2)$
– $PI(8) = \{[q, \$, \#, q', Z_0\alpha] \mid q, q' \in Q,\ \alpha \in \Sigma^*,\ 0 < |\alpha| < K_\Gamma\}$, $FI(8) = U \setminus (V \cup \Sigma \cup \{Z^\circ \mid Z \in \Sigma\} \cup PI(8))$.
– $PO(8) = U$, $FO(8) = \emptyset$,

• Node 9 (the output node of the network):
– $PI(9) = \{[q, \$, \#, f, Z_0\alpha], [q, \$, \#, f, \lambda] \mid q \in Q,\ f \in F,\ \alpha \in \Sigma^*,\ 0 < |\alpha| < K_\Gamma\}$, $FI(9) = U \setminus (\Sigma \cup PI(9))$.

We show that Π accepts a word w iff $w \in L$. The construction uses the idea, presented in Theorem 2, of synchronously decreasing an index to correctly simulate a move of the automaton. A symbol $[q, a, Z, q', \alpha]$ represents a transition $(q', \alpha) \in \delta(q, a, Z)$ of Γ. Now, 3 steps of matching are needed; one, as in Theorem 2, to match a to the currently read input symbol, one to match the symbol on top of the stack to Z and finally one to write α onto the stack. Again, we will trace a possible input string and show that it can produce the string in the output node if and only if it is accepted by the original automaton.

1. $L \subseteq L(\Pi)$.

Let w be a word from L. Assume that w is present in the node 1 at the beginning of the computation. In this node the string becomes $w[q_0, a, Z_0, q_1, \alpha]$, with a the first symbol of w and $(q_1, \alpha) \in \delta(q_0, a, Z_0)$. The string exits the node 1 and enters node 2, where it becomes $w[q_0, a, Z_0, q_1, \alpha]Z_0$ and is communicated to node 3. Now a so-called iterative phase is started. Assume, for the sake of generality, that at the beginning of the iterative phase a string of the form $a_k x[q, a_k, Z_i, q', \alpha]\beta Z_i$ is found in node 3 (this assumption holds after the preprocessing phase). In node 3 the string becomes $a_k x[q, \overline{a_k}, Z_i, q', \alpha]'\beta Z_i$, and is communicated in the network; it can only enter node 4. The first cycle of the iterative phase begins now. In this node, the string is transformed into $a^{\bullet}_{k-1} x[q, \overline{a_{k-1}}, Z_i, q', \alpha]^{\bullet}\beta Z_i$ and is further communicated; the string enters node 5. Now the string becomes $a'_{k-1} x[q, \overline{a_{k-1}}, Z_i, q', \alpha]'\beta Z_i$ and goes back to node 4. This cycle continues until the string $a'_1 x[q, \overline{a_1}, Z_i, q', \alpha]'\beta Z_i$ enters node 4, where it is transformed into $\$^{\bullet} x[q, \$, Z_i, q', \alpha]^{\bullet}\beta Z_i$. This string goes to node 5 where it becomes $\$x[q, \$, Z_i, q', \alpha]\beta Z_i$. This string can only enter node 6 where the leftmost symbol \$ is deleted. This finishes the matching of the input symbol.

The resulting string $x[q, \$, Z_i, q', \alpha]\beta Z_i$ enters node 3 where it is transformed into $x[q, \$, \overline{Z_i}, q', \alpha]'\beta Z_i$, marking the start the second cycle, which matches the symbol on top of the stack. Then the string enters node 4, where $x[q, \$, \overline{Z_{i-1}}, q', \alpha]^{\bullet}\beta Z^{\bullet}_{i-1}$ is obtained. It then goes to node 5 where it is transformed into $x[q, \$, \overline{Z_{i-1}}, q', \alpha]'\beta Z'_{i-1}$ and is communicated back to node 4. Again, this cycle is iterated until the string becomes $x[q, \$, \overline{Z_0}, q', \alpha]'\beta Z'_0$ and enters node 4; here it is transformed into $x[q, \$, \#, q', \alpha]^{\bullet}\beta\#^{\bullet}$ and is communicated to node 5.

In this node we obtain $x[q, \$, \#, q', Z_0\alpha]\beta\#$, if $\alpha \neq \lambda$, or $x[q, \$, \#, q', \lambda]\beta\#$ otherwise; these strings can only enter node 7, where the rightmost symbol # is deleted. Thus we may obtain the strings $x[q, \$, \#, q', Z_0\alpha]\beta$, for $\alpha \neq \lambda$, or $x[q, \$, \#, q', \lambda]\beta$, otherwise. In both cases, if $x = \lambda$ and $q' \in F$, the string enters the output node 9. If $\alpha = \lambda$, the third (write-in-stack) cycle can be skipped, and the string enters the node 3 where the simulation of the next transition starts by rewriting the string as $x[q', b, Z', q'', \alpha']\beta$, where b is the first symbol of x, Z' is the last symbol of β, and $(q'', \alpha') \in \delta(q', a', Z')$. Otherwise, if $\alpha \neq \lambda$, the string enters node 8 and becomes $x[q, \$, \#, q', Z_0\alpha]\beta\diamond$. It is then communicated to node 3 where it is transformed into $x[q, \$, \#, q', \alpha]^{\circ}\beta\circ$ and is sent to node 4. Now, the write-in-stack section of the iterative phase begins. Assume that $\alpha = Z_t\alpha''$. In node 4 the string is transformed into $x[q, \$, \#, q', Z_{t-1}\alpha'']^{\diamond}\beta Z^{\diamond}_1$, which enters node 5. Here we obtain the string $x[q, \$, \#, q', Z_{t-1}\alpha'']^{\circ}\beta Z^{\circ}_1$, which goes back to node 4. In node 4, the string is transformed into $x[q, \$, \#, q', Z_{t-2}\alpha'']^{\circ}\beta Z^{\diamond}_2$,

and goes back to node 5, and the cycle is iterated until the string $x[q, \$, \#, q', Z_0\alpha'']^\diamond \beta Z_t^\diamond$ enters node 5.
If $\alpha'' = \lambda$, this string is transformed into $x[q', b, Z', q'', \alpha']\beta$, where b is the first symbol of x, Z' is the last symbol of β, and $(q'', \alpha') \in \delta(q', a', Z')$. This string enters node 3 and the iterative phase is restarted for this next transition. Otherwise, if $\alpha'' \neq \lambda$, the string is transformed into $x[q, \$, \#, q', Z_0\alpha'']\beta Z_t$ and enters node 8 where a $\diamond$ symbol is inserted to the right. The write-in-stack phase then continues for the first symbol of α''. It is clear that in one full iteration of the iterative phase we obtain from the string $a_k x[q, a_k, Z_i, q', \alpha]\beta Z_i$ the string $x[q', a_t, Z_h, q'', \alpha']\beta\alpha$, where $(q', \alpha) \in \delta(q, a_k, Z_i)$. Thus, for an input word $w \in L$, in the $|w|$-th iteration of this phase, we will have obtained from the word $w[q_0, a, Z_0, q_1, \alpha]Z_0$ the word $[q, \$, \#, f, \alpha]y$ with f final state, and this words enters the output node. To conclude $w \in L(\Pi)$ and, consequently $L \subseteq L(Pi)$.

2. $L \supseteq L(\Pi)$.
In most cases the filters ensure that the derivation can be performed only as described above. Moreover, by the same mechanism as in Theorem **??**, if the matching process is unsuccessful, the resulting string will be lost. However, some cases require some closer attention. First, we analyze the first cycle. Assume that other symbol a_t, with $t \leq n$, than the first symbol of $x[q, a_k, Z_i, q', \alpha]\beta$ is transformed into a_t'. If $t \leq k$ this symbol is transformed into \$ in t iterations of the cycle, and the string, communicated by node 5, can enter node 6 (only in the case when $t = k$, otherwise it cannot enter any node and is lost). Here the \$ is not deleted, and the string is, once again, lost.
Also, to see that there cannot be harmful interference between different cycles, assume that during the execution of the first cycle a symbol Z_t, $t \leq m$, is transformed into Z_t'. Again, in at most t steps either the string will contain symbols $[q, \$, Z_i, q', \alpha]$ and Z' with $Z \in \Sigma$, so cannot enter any node and is lost, or alternatively it will contain the symbol # and is also lost. Consequently, the only symbols that can be transformed in the first cycle are the first symbol of the string and the symbol $[q, a_i, Z_i, q', \alpha]$.
Similar arguments show that during the second cycle, if a symbol a is rewritten to a', no accepting computation can follow. Also if during the write-in-stack section of the computation, a symbol a or Z is transformed into a' or Z', respectively, the string will be eventually lost. These considerations show that only the strings that are processed during the iterative phase as described in the proof of the inclusion 1 can be accepted by the network. Thus $L \supseteq L(\Pi)$ and we have proved that $L = L(\Pi)$. □

Corollary 1. *Any context-free language L can be accepted by an AHNEP Π of size 9, such that for each $w \in L$, $TIME_{AHNEP}(\Pi) \in \mathcal{O}(|w|)$.*

Proof. During the iterative phase, a constant number of steps is performed. This number depends on the size of the alphabet and on the maximum length of a string that is written in a transition on the stack. Since the iterative phase is performed for $|w|$ times, given that w is the input word, it follows that the total number of steps performed by the network on a input of length n is $\mathcal{O}(n)$. □

This shows that our construction is not only efficient from a descriptional point of view, but from the computational point of view as well. This result seems interesting

considering that we already knew (from Theorem 1) that L can be accepted using a constant size AHNEP with 24 nodes simulating a one-tape Turing Machine accepting L, and now we were able not only to improve the size of the network to 9, but also we have shown that the time complexity of the acceptance of the words in L, using this construction, is linear (we are not aware of an one-tape Turing Machine algorithm working in linear time for the acceptance of context-free languages, thus we cannot use Theorem 1 to produce an AHNEP working in linear time for the acceptance of context-free languages, as well).

Acknowledgments. This work was done while Peter Leupold was funded as a post-doctoral fellow by the Japanese Society for the Promotion of Science under grant number P07810. Remco Loos' work is supported by research grant ES-2006-0146 of the Spanish Ministry of Education and Science. Florin Manea acknowledges partial support from the Romanian Ministry of Education and Research (PN-II Program, Project *GlobalComp - Models, semantics, logics and technologies for global computing*). Finally, all three authors would like to express their gratitude to Victor Mitrana for the guidance and support provided as their doctoral advisor.

References

[1] CASTELLANOS J., MARTÍN-VIDE C., MITRANA V., SEMPERE J., *Solving NP-complete problems with networks of evolutionary processors*, Lect. Notes in Comput. Sci., **2084**, pp. 621–628, 2001.

[2] CASTELLANOS J., MARTÍN-VIDE C., MITRANA V., SEMPERE J., *Networks of evolutionary processors*, Acta Inform., **39**, pp. 517–529, 2003.

[3] CASTELLANOS J., LEUPOLD P., MITRANA V., *Descriptional and computational complexity aspects of hybrid networks of evolutionary processors*, Theor. Comput. Sci., **330**, pp. 205–220, 2005.

[4] HOPCROFT J.E., ULLMAN J.D., *Introduction to Automata Theory*, Languages and Computation, Addison-Wesley, 1979.

[5] MARGENSTERN M., MITRANA V., PEREZ-JIMENEZ M., *Accepting hybrid networks of evolutionary systems*, Lect. Notes in Comput. Sci., **3384**, pp. 235–246, 2005.

[6] MANEA F., MARGENSTERN M., MITRANA V., PEREZ-JIMENEZ M., *A New Characterization of NP, P, and PSPACE With Accepting Hybrid Networks of Evolutionary Processors*, in press, Theory of Computing Systems, Springer, doi:10.1007/s00224-008-9124-z.

[7] MANEA F., MARTÍN-VIDE C., MITRANA V., *On the Size Complexity of Universal Accepting Hybrid Networks of Evolutionary Processors*, Math. Struct. in Comput. Sci., **17:4**, pp. 753–771, 2007.

[8] MANEA F., MITRANA V., *All NP-problems Can Be Solved in Polynomial Time by Accepting Hybrid Networks of Evolutionary Processors of Constant Size*, Inf. Process. Lett., **103:3**, pp. 112–118, 2007.

[9] MANEA F., *On the recognition of Context-Free languages using AHNEPs*, Int. J. of Comput. Math., **84:3**, pp. 273–285, 2007.

[10] MARTÍN-VIDE C., MITRANA V., PEREZ-JIMENEZ M., SANCHO-CAPARRINI F., *Hybrid networks of evolutionary processors*, Lect. Notes in Comput. Sci., **2723**, pp. 401–412, 2002.