Networks of Evolutionary Processors as Natural Language Parsers

June 24, 2008

Abstract

Networks of Evolutionary Processors (NEPs) –introduced in Castellanos et al. (2001)– are a new computing mechanism directly inspired from the behavior of cell populations. In the paper, we we explore the possibility of Networks of Evolutionary Processors (NEPs) to be suitable for modelling natural language – an entity generated in parallel by a modular architecture – and specially syntax – a modular device of specialized processors inside the modular construct of language. An implementation of NEPs for parsing of simple structures $[[NP]\ V\ [NP]]$ is also suggested. Moreover, we introduce the concepts of parallel processing and linearity in the formalization of NEPs as accepting devices, and suggest a new line of research by applying these networks to natural language.

1 Introduction

Networks of Evolutionary Processors (NEP) are a new computing mechanism directly inspired from the behavior of cell populations. Every cell is described by a set of words (DNA) evolving by mutations, which are represented by operations on these words. At the end of the process, only the cells with correct strings will survive. The global framework for the development of NEPs has to include DNA computing [13], membrane computing [12] – that focalizes also in the behavior of cells –, and specially with the theory of grammar systems [5], which share with NEPs the idea of several devices working together and exchanging results.

First precedents of NEPs as generating devices can be found in [7] and [6]. The topic was introduced in [2], and further developed in [3, 4, 11].

NEPs can be defined as graphs whose nodes are processors performing some very simple operations on strings and sending the resulting strings to other nodes. Every node has filters that block some strings from being sent and/or received. This functioning allows the specialization of each processor.

In this paper we suggest that hybrid networks of evolutionary processors (NEPs) can also be used as a modular description device in linguistics. The

1

idea to use NEPs as recognizers is not original as a preliminary approach to accepting NEPs was already introduced in [10] and developed in a series of contributions [1, 9].

As was shown in [3], NEPs with regular filters are Turing-equivalent. This fact suggests that they can be used as a formal base for a "programming language" for processing linguistic information on all levels. In this context, NEPs have the following advantages:

- the basic structures of NEPs, i.e., nodes, filters and the graph-based structure, can be used in a very natural way for modelling different kinds of linguistics objects and their interaction;

- a NEP simulator can be easily implemented as a software environment for constructing language-processing modules, and in particular, syntactic parsers where a parser is implemented in our "NEP language" (i.e., in fact, as a NEP);

- we can specify the output format of such a parser according to the needs of the setting for which it is being developed.

In this paper, we suggest a formalisation of NEPs adjusted for linguistic purposes. We also illustrate the applicability of this framework for natural language parsing with an example of a NEP recognizing simple syntactic structures using linear input and output.

To do so, Section 2 is devoted to give a general definition of NEPs, pointing out some of their main features. The suitability of NEPs for linguistics is discussed in section 3. In Section 4, a new variant of NEPs is introduced, which allows sentence recognition is introduced. Finally, in Section 5 we give an example.

2 NEPs: Definitions and Key Features

Following [3], we introduce the basic definition of NEPs.

Definition 1. *A Network of Evolutionary Processors of size n is a construct:*

$$\Gamma = (V, \{N_1, N_2, ..., N_n\}, G),$$

where:

- *V is an alphabet and for each $1 \leq i \leq n$,*

- *$N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:*

    - *M_i is a finite set of evolution rules of one of the following forms only*

>   *i. $a \rightarrow b$, where $a, b \in V$ (substitution rules),*
>
>   *ii. $a \rightarrow \varepsilon$, where $a \in V$ (deletion rules),*
>
>   *iii. $\varepsilon \rightarrow a$, where $a \in V$ (insertion rules).*
>
> – *A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i-th node.*
>
> – *PI_i and PO_i are subsets of V^* representing the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i(w \in PO_i)$.*

- *$G = (\{N_1, N_2, \ldots, N_n\}, E)$ is an undirected graph called the underlying graph of the network. The edges of G, that is the elements of E, are given in the form of sets of two nodes. The complete graph with n vertices is denoted by K_n.*

A configuration of a NEP is an n-tuple $C = (L_1, L_2, \ldots, L_n)$, with $L_i \subseteq V^*$ for all $1 \leq i \leq n$. It represents the sets of strings which are present in any node at a given moment.

A given configuration of a NEP can change either by an evolutionary step or by a communicating step. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i. The change in the configuration by an evolutionary step is written as $C_1 \Rightarrow C_2$.

When changing by a communication step, each node processor N_i sends all copies of the strings it has, able to pass its output filter, to all the node processors connected to N_i and receives all copies of the strings sent by any node processor connected with N_i, if they can pass its input filter. The change in the configuration by a communication step is written as $C_1 \vdash C_2$.

3   NEPs for Modelling Linguistics

This formal construct can provide a good framework for attempting a new description and formalization of linguistics. We think three features can be taken from NEPs and are crucial for their application to language processing and, especially, to parser technologies. NEPs are a) *modular* b) *communicating* systems that work in c) *parallel*.

NEPs are *modular* devices because they can be described as distributed systems of contributing nodes, each one of them carrying out just one type of operation. Every node should be defined depending on the specific domain we aim to tackle. Moreover, the processors of the network are *specialized*, since each one of them is designed for a specific task, in a way that the final success of the system depends on the correct working of every one of the agents and on the correct interacting between different agents.

Communication is the feature that accounts for the social competences of the modules. By means of communication, agents can interact to achieve common goals, work for their own interests, or even isolate. Although different processes and operations are done in parallel, NEPs also need to define some type of *coordination* between nodes, since alternative steps of *communication* have to be synchronized.

Nodes communication is said to be *graph-supported* and *filter-regulated*.

We say that the communication among nodes is *graph-supported* because the edges of the graph govern its social interactions. Therefore, if there are non-connected nodes in a NEP, these nodes do not communicate with other processors.

The fact that the communication is *filter-regulated* means that it is driven by means of input and output filters.

- The objective of the *input filter* is to control the information or structures entering the node. This helps the specialization, by the selection of the strings/structures the node can process, and protects the module from possible harmful items.

- By the *output filter* the node selects the information it wants to share and also when it wants to share it.

Parallelism allows the system to perform different tasks at the same time by different processors. Some of linguistic processes, as well as language generation in general, are considered to be parallel. For apparently sequential interactions (i.e. dialogue) parallelism allows working with multi-modal exchanges.

Therefore, taking the four main modules usually considered in linguistics – syntax, semantics, phonology, morphology – and considering the edges of the underlying graph as a way for communicating or not communicating, we can draw the simple scheme of a "language generation NEP" that is shown in Figure 1. Ph stands for phonetics, M represents morphology, Sy is syntax and Se refers to semantics. The semantic node is only communicating with the phonological and the syntactic ones because it seems there is no interaction between semantics and phonetics.
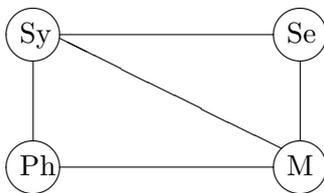

Figure 1: Modular Linguistic NEP

4 Using NEPs for specifying parsers

In the sequel we will try to make some modifications in the computational definition of NEPs for them to work as parsers of natural language. For the sake of simplicity, we establish a methodological restriction: we will focus on simple sentences with the shape $[S \ V \ O]$, where $S \rightarrow [NP]$, $O \rightarrow [NP]$, that is, sentences with the form $[[NP] \ V \ [NP]]$. Our purpose is to check if this mechanism is powerful and simple enough to be used as a model for parsing of complex linguistic strings.

As it has been already highlighted, the model hereby presented takes advantage of the main features of NEPs: *modularity*, *specialization* and *parallelism*. Adopting these characteristics in the modelling of our device one can improve its efficiency and decrease the complexity.

Modularity and specialization can be useful because they allow designing processors which only accept, recognize and label a single type of syntactic phrases or functions. Such strategy makes easier the first work of classifying the lexical pieces according to their grammatical category. By parallelism, all lexical items will be taken and analyzed at the same time, and afterwards, grammatical units can be packed in different processors.

In general, the system has to perform two main tasks: a) to recognize correct strings, like an automaton could do, and b) to give an output with labeled elements that give account to the syntactic structure of the input sentence.

To be able to accept and analyze a sentence, the NEP has to perform the following steps: 1) to recognize every word, 2) to make a map of its linguistic features, 3) to gather the non-terminal units in phrases establishing their beginning and end, and finally 4) to give a linear structural version of the whole sentence.

In order to implement such a NEP, we propose several types of specialized nodes:

- nodes specialized in accepting lexical items recognizing their grammatical categories;

- nodes specialized for accepting and operating with different types of phrases, ie., nominal phrases (NP), prepositional phrases (PP);

- nodes for accepting sentences.

The structure of the graph is given by the class of sentences that have to be processed. Many different NEPs could be designed for the parsing of the same type of syntactic structures, but we are looking for the best NEP for every structure in terms of number of nodes and complexity.

Moreover, since the structure we are working with has just two types of sub-structures, namely NP and V, at least three specialized nodes are

needed, one for sub-components of NP (just one element in a minimal NP), one for the recognition and labeling of NP and the other one for the recognition and analysis of V. The sub-components of NP include nodes for lexical units (N, and ART). Since a node for the packing of the final output is also necessary, then at least a graph of four nodes has to be designed.

Therefore our device will consist of the following nodes: a) a node for recognizing articles, ART b) three processors for labeling nouns Nc, Np, Nv, c) a node for recognizing nominal phrases, NP, d) a node for analyzing verbal structures, V, e) a node for analyzing nominal phrases, NP and f) a node for labeling sentences, P. Beside these nodes we will also need an output node.

In the input filter of specialized nodes, the only elements accepted will be those that can be part of phrases they can recognize. In the output filter of these nodes, only labeled phrases will be allowed to pass and be sent to the other filters.

To perform the recognition process, two types of alphabets are necessary: V, the alphabet of the input symbols, which are terminal strings, i.e., lexical items, and Σ the alphabet of grammatical types symbols – which correspond to grammatical categories – plus feature symbols. For the simple sentences we are dealing with, we recognize several strings symbols belonging to Σ^* which are needed to process the sentence: $[\ ]_N$, $[\ ]_V$, $[\ ]_{ART}$, $[\ ]_{NP}$.

In order to model the subject-verb agreement, some of these symbols will be provided with morphological markers. First of all, two different marks will be established for the category $[V]$ in order to distinguish between the two different forms of the English verb in present: s stands for the general form, and p for the third person. This way, when the node is receiving a lexical item x, it analyzes it, and inserts the grammatical types symbols giving us $[x]_{vs}$ or $[x]_{vp}$.

Moreover, in order to fulfill the agreement with the verb, $[N]$ has to be recognized with the same parameters as the verb and moreover with some that give us the agreement with the article, $\{c, p, v\}$. On the other hand, in order to model agreement inside a phrase, we introduce separate nodes for the same phrase with different morphological characteristics.

For distinguishing the article "a" from the articles "an" and "the", the feature $[ART]$ will be $[\ ]_{a1}$ for "a", $[\ ]_{a2}$ for "an" and $[\ ]_{a3}$ for "the", where the absence of any symbol means it works for both singular and plural. If the agreement is not accomplished inside NP or in the NP at the left of the verb and V, then the sentence will not be recognized.

For actually constructing the phrases as the gathering of several elements belonging to different nodes, we introduce in our NEP a rule that will combine these symbols together. This rule will be called adjunction and the agreement will be done according to some predefined requirement.

With the elements we have just explained, a NEP for sentence analysis can be defined as follows:

Definition 2. *A NEP for the analysis of simple sentences* $[[NP]V[NP]]$ *is a general structure:*

$$\Gamma = (V, \Sigma, \{ART, N_v, N_c, N_p, V, NP, P, Out\}, G)$$

where:

- *V is the input vocabulary,*

- *Σ is the grammatical type vocabulary,*

- *$\{ART, N_v, N_c, N_p, V, NP, P, Out\}$ are the node processors N_1, $N_2, \ldots, N_7, Out$ of the network with the following definition. For every node $N_i = (M_i, A_i, PI_i, PO_i)$:*

  - *M_i is the finite set of evolution rules of the form:*
    - *i. $a \rightarrow [a]_n$, where $a \in \{V \cup \Sigma\}^*$ (insertion rule with n indicating the index of the node),*
    - *ii. $a \rightarrow \varepsilon$, where $a \in V$ (deletion rule in which all elements of V are erased), or*
    - *iii. $a, b \rightarrow [ab]_n$, where $a, b \in V \cup \Sigma$ (adjunction rule in which two elements coming from different nodes are wrapped together);*
  - *A_i is the set of strings over V in the initial configuration,*
  - *PI_i are the input filters over $\{V \cup \Sigma\}^*$, and*
  - *PO_i are the output filters over $\{V \cup \Sigma\}^*$.*

- *Out is the output node that has a special input filter that compares the initial phrase shuffled with Σ^*, with the words that are trying to enter the node ($Inp \sqcup \sqcup \Sigma^* = inputword$)*

- *$G = (Vx, Ev)$ is the network graph where*

  - *$Vx = \{N_1, N_2, \ldots, N_7, Out\}$ are its nodes, and*
  - *$Ev = (N_1 N_6, N_2 N_6, N_3 N_6, N_4 N_6, N_5 N_6, N_6 N_7, N_7 Out)$ are the arcs.*

The computation works almost like in a regular NEP, combining evolutionary steps and communication steps. Moreover, the system is totally parallel, even in the input mechanism, and every node applies, during evolutionary steps, as many rules as it can (one rule for each of its elements).

This system stops when no operation can be performed by the NEP or the Out node receives an input. In the latter case the recognition process ends, the initial sentence is accepted as correct and its structure is displayed.
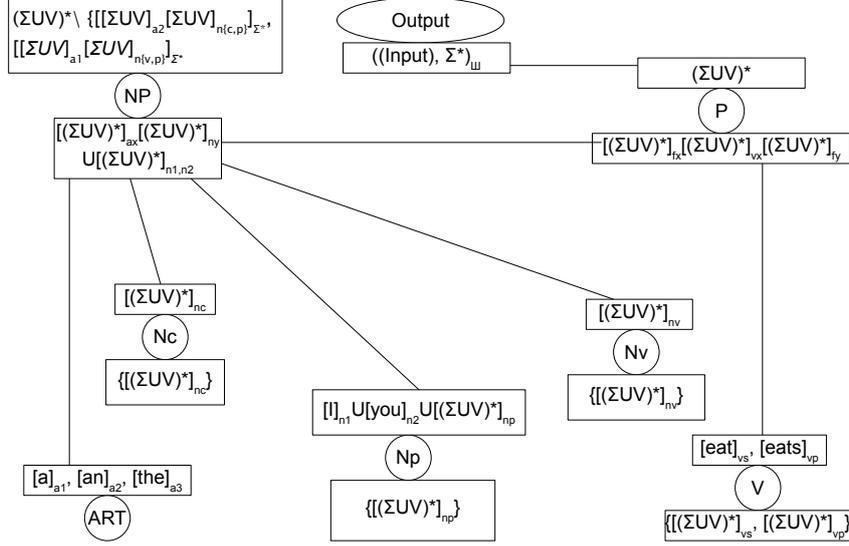
Figure 2: Example: Simple structure NP V NP

5 Two examples

In this section, we introduce two examples on how the parsing works. Both examples have only phrases $[NP]$ and $[V]$ but, while the first one fits to the simple structure $[[NP]V[NP]]$, the second can be considered one of the non-context free pieces of natural language.

5.1 Example 1

The example we use is the sentence *The boy eats the apple.*.

The NEP consists of 8 nodes. The alphabet Σ consists of the symbols $\{[,],1,2,3,a,c,f,n,p,s,v\}$ that indicate the type of the structure. Note that for this example the extension to recursive model is simple due to the input filters. These will also allow us to introduce adjectives.

The general definition of the system is as follows:

$$\Gamma = (V, \Sigma, \{ART, N_v, N_c, N_p, V, NP, P, Out\}, G)$$

where:

- $V = \{apple, apples, boy, boys, eat, eats, a, an, the\}$

- $\Sigma = \{[,],1,2,3,a,c,f,n,p,s,v\}$

- $Out = \{\{(Inp \sqcup x) \to x,\ \text{where}\ x \in \Sigma^*\},\ \emptyset,\ \{The\ boy\ eats\ an\ apple\} \sqcup \Sigma^*,\ \Sigma^*\}$

- $ART = \{(a \to [a]_{a1},\ an \to [an]_{a2},\ the \to [the]_{a3}),\ \{a, an, the\},\ \emptyset,\ \{[a]_{a1}, [an]_{a2}, [the]_{a3})\}\}$

- $N_v = \{(apple \to [apple]_{nv}),\ \{apple\},\ \{[(V \cup \Sigma^)*]_{nv}\},\ \{[(V \cup \Sigma^)*]_{nv}\}\}$, that is, all third person singular nouns starting with a vowel

- $N_c = \{(boy \to [boy]_{nc}),\ \{boy\},\ \{[(V \cup \Sigma)^*]_{nc},\ \{[(V \cup \Sigma)^*]_{nc}\}\}$, that is, all third person singular nouns starting with a consonant

- $N_p = \{(apples \to [apples]_{np},\ boys \to [boys]_{np}),\ I \to [I]_{np},\ you \to [you]_{np},\ \{apples, boys\},\ \{[(V \cup \Sigma)^*]_{np},\ \{[(V \cup \Sigma)^*]_{np}, [I]_{np}, [you]_{np}\}\}$, that is all plural nouns and singular ones that are not third person

- $V = \{(eats \to [eats]_{vs},\ eat \to [eat]_{vp}),\ \{eat, eats\},\ \{[(V \cup \Sigma)^*]_{vs}, [(V \cup \Sigma)^*]_{vp}\},\ \{[(V \cup \Sigma)^*]_{vs}, [(V \cup \Sigma)^*]_{vp}\}\}$

- $NP = \{(([x]_{ax_1}[y]_{ny_1} \to [[x]_{ax_1}[y]_{ny_1}]_{fy_1},\ ([y]_{np} \to [[y]_{np}]_{fp})),\ \emptyset,\ (V \cup \Sigma)^*,\ \{(V \cup \Sigma)^* \setminus \{[[V \cup \Sigma]_{a2}[V \cup \Sigma]_{n\{c,p\}}]_{\Sigma^*}, [[V \cup \Sigma]_{a1}[V \cup \Sigma]_{n\{v,p\}}]_{\Sigma^*}\}\}\}$

- $P = \{(([x]_{nx_1}[y]_{vx_1}[z]_{nz_1} \to [[x]_{nx_1}[y]_{vx_1}[z]_{nz_1}]_{x_1}),\ \emptyset,\ ([(V \cup \Sigma)^*]_{fx}[V \cup \Sigma)^*]_{vx}[V \cup \Sigma)^*]_{fy},\ (V \cup \Sigma)^*\}$

- $G = (V, Ev)$

  - $V = \{ART, N_v, N_c, N_p, V, NP, P, Out\}$
  - $Ev = (ART\ NP,\ N_v NP,\ N_c NP,\ N_p NP,\ V\ P,\ NP\ P,\ P\ Out)$

Here $x, y, z, x_1, y_1, z_1 \in \Sigma \cup V$.

In order to understand how our example program works we will "run" it on two different sentences. The first one will be the sentence: "The boy eats an apple". Before the first computation step, ART will contain the words *the* and *an*, Nc will contain the words *boy* and *apple*, and V the verb *eats*. The second step is a communication one. The third step, which is a computation step, will produce in NP the combinations $\{[the]_{a3}[boy]_{nc}, [an]_{a2}[boy]_{nc}, [the]_{a3}[apple]_{nv}, [an]_{a2}[apple]_{nv}\}$. Using the output filter, NP is able to exclude from the next computation the second and the third component. Hence, after the fifth computation step we will have in P the string set $\{[[the]_{a3}[boy]_{nc}]_{fs}\ [eats]_{vs}\ [[an]_{a2}[apple]_{nv}]_{fs}, [[an]_{a2}[apple]_{nv}]_{fs}\ [eats]_{vs}\ [[the]_{a3}[boy]_{nc}]_{fs}\}$. The last computation step, the seventh one, is going to output $[[\ ]_{a3}[\ ]_{nc}]_{fs}\ [\ ]_{vs}\ [[\ ]_{a2}[\ ]_{nv}]_{fs}$ which is the structure of the input sentence. This implies that the sentence was correct.

The second sentence we will look at is "The apples eats a boy". It is easy to observe that this sentence is a wrong one. Before the first computation step ART will contain the words *the* and *a*, Nc the words *boy* and *apples* and

V the verb *eats*. The third computation step will produce in NP the string set $\{[[the]_{a3}[apples]_{np}]_{fp},\ [[the]_{a3}[boy]_{nc}]_{fs},\ [[a]_{a1}[boy]_{nc}]fs,\ [[apples]_{np}]_{fp}\}$. By looking at the verb we observe that in the fifth computation step we will get the string set $\{[[the]_{a3}[boy]_{nc}]_{fs}\ [eats]_{vs}\ [[the]_{a3}[apples]_{np}]_{fp},\ [[a]_{a1}[boy]_{nc}]_{fs}$ $[eats]_{vs}\ [[the]_{a3}[apples]_{np}]_{fp},\ [[the]_{a3}[boy]_{nc}]_{fs}\ [eats]_{vs}\ [[apples]_{np}]_{fp},\ [[a]_{a1}$ $[boy]_{nc}]_{fs}\ [eats]_{vs}\ [[the]_{a3}[apples]_{np}\}$. As no more computation steps are possible and no output is produced (none of the possible outputs represent a shuffle between the initial phrase and Σ) the automaton stops and concludes that the input sentence was wrong.

5.2 Example 2

Let us consider the case of non context-free examples. Let us take a well known model of Dutch sentence that has these properties: *"Wim Jan Marie de kinderen zag helpen leren zwemmen"*. The English word by word translation would be *"Wim Jan Marie the children saw to help to teach to swim"*, while the normal translation is: *"Win saw Jan help Marie teach the children to swim"*. Hence this kind of phrases have a structure of the form $s_1s_2s_3s_4v_1v_2v_3v_4$, where s is subject and v is verb. For the sake of simplicity we consider "de kinderen" as one word.

A solution for the processing of this kind of sentences would be the specialization adjunction nodes. This node would do a virtual adjunction of a subject and a verb. This would consist actually in a marking of the two with the same index $(a_s, b_v \rightarrow a_{s1}, b_{v1})$. Future adjunction nodes would concatenate several subjects (verbs) together, taking care that the indices are in an increasing order. A final node would just put together sequences of subjects and verbs. Since the output node would compare our result with the input sentence, only a correct version of this would be accepted. This implies that the subjects, respectively the verbs, are indexed in an increasing order, and via a morphism they are connected.

$$\Gamma = (V, \Sigma, \{Inp, ART, N_v, N_c, N_p, V, NP, P, Out\}, G)$$

where:

- $V = \{Wim, Jan, Marie, de\ kinderen, zag, helpen, leren, zwemmen\}$

- $\Sigma = \{[,], 1, 2, 3, a, c, f, n, p, s, v\}$

- $Out = \{\{(Inp \sqcup\!\sqcup x) \rightarrow x,\ \text{where } x \in \Sigma^*\},\ \emptyset,\ \{Wim\ Jan\ Marie\ de\ kinderen\ zag\ helpen\ leren\ zwemmen\} \sqcup\!\sqcup \Sigma^*, \Sigma^*\}$

- $N_c = \{((Wim \rightarrow [Wim]_{nc1}),\ (Wim \rightarrow [Wim]_{nc2}),\ \ldots,\ (Wim \rightarrow [Wim]_{nc5}),\ \ldots,\ (de\ kinderen \rightarrow [de\ kinderen]_{nc1}),\ \ldots,\ (de\ kinderen \rightarrow [de\ kinderen]_{nc5})),\ \{Wim,\ Jan,\ Marie,\ de\ kinderen\},\ \emptyset,\ \{[V]_{nci}\ |\ i \leq |V|/2\}\}$
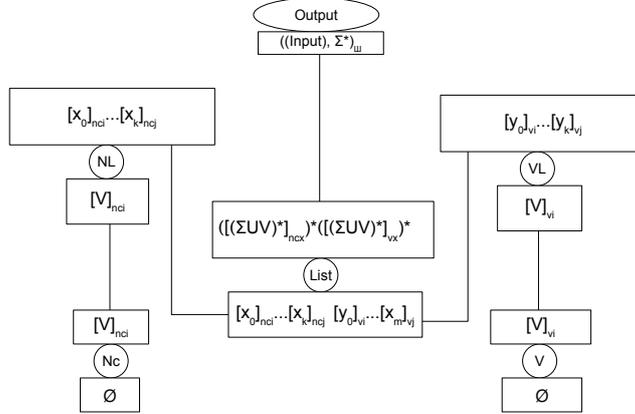
Figure 3: Example 2: Cross-serial dependencies in Dutch

- $V = \{((zag \rightarrow [zag]_{v1}), (zag \rightarrow [zag]_{v2}), \ldots, (zag \rightarrow [zag]_{v5}), \ldots, (zwemmen \rightarrow [zwemmen]_{v1}), \ldots, (zwemmen \rightarrow [zwemmen]_{v5})), \{zag, helpen, leren, zwemmen\}, \emptyset, \{[V]_{vi} \mid i \leq |V|/2\}\}$

- $NL = \{[x]_{nc_i}[y]_{nc_j} \rightarrow [x]_{nc_i}[y]_{nc_j}, \text{ for } i < j\}, \emptyset, \{[V]_{nci} \mid i \leq |V|/2\}, \{[x_0]_{nc_i} \ldots [x_k]_{nc_j} \mid 0 \leq k, 0 \leq i < j \leq |V|/2 \text{ and for any factor } [x]_{nc_{k_1}}[x]_{nc_{k_2}}, k_1 < k_2\}\}$

- $VL = \{[x]_{v_i}[y]_{v_j} \rightarrow [x]_{v_i}[y]_{v_j}, \text{ for } i < j\}, \emptyset, \{[V]_{vi} \mid i \leq |V|/2\}, \{[x_0]_{v_i} \ldots [x_k]_{v_j} \mid 0 \leq k, 0 \leq i < j \leq |V|/2 \text{ and for any factor } [x]_{v_{k_1}}[x]_{v_{k_2}}, k_1 < k_2\}\}$

- $List = \{([x_0]_{nc_i} \ldots [x_k]_{nc_j} \ [y_0]_{v_i} \ldots [y_m]_{v_j} \rightarrow [x_0]_{nc_i} \ldots [x_k]_{nc_j}[y_0]_{v_i} \ldots [y_m]_{v_j}\}, \emptyset, (([V \cup \Sigma)^*]_{ncx})^*([V \cup \Sigma)^*]_{vx})^*), (V \cup \Sigma)^*\}$

- $G = (V, Ev)$

  - $V = \{N_c, V, NL, VL, List, Out\}$
  - $Ev = (N_c \ NL, V \ VL, NL \ List, VL \ List, List \ Out)$

6 Discussion and Future Work

In this paper, we present an application of NEPs for the analysis and recognition of sentences of natural language. In the NEPs we have modeled, each processor is specialized in the processing of different syntactic patterns: NP, VP, S. An important feature of the system is that both input and output

are linear, the lexical items are the input and syntactic structures are the output.

We have highlighted the advantages of such constructs because of their characteristics and functionality. For the future, a more precise analysis of the components of the NEPs for parsing of natural language would be necessary as well as a detailed study of their connection with real neuronal capabilities.

There exist also some problems regarding the proposed NEP-based approach to the language processing. As has been mentioned before, the accepting power of NEPs is that of a Turing machine which, on the one hand, allows us to specify a program without any restriction regarding the linguistic framework we want to simulate and the linguistic facts we want to deal with. But on the other hand, it excludes any possibility of the automatic validation of the properties of a program and makes the checking of a large program's correctness very difficult and time-consuming. This concerns the polynomial-time (Turing-)complexity of the running time of a program that is generally considered to be crucially important for automatic language processing.

At the same time, the specific structure of the elementary objects used in NEPs, makes out of this formalism a very convenient tool for modelling linguistic objects. From the practical perspective, it means that NEPs can be used as a formal base for specialized task-oriented programming environments for natural language processing.

In this light, a possible future line of research could be the possible restrictions to the NEPs that, on the one hand, will guarantee that any program executes in polynomial time, based on the length of the input and, on the other hand, will not hinder too much the framework's expressivity for linguistic purposes.

We claim that NEPs are a very convenient system, not only for explaining natural language processing, but also for simulating knowledge representation and cognitive mechanisms. Moreover, NEPs provide a consistent theoretical framework for the formalization of human-computer interfaces. In this model, the human capacity of transforming the world by means of the word and knowledge can be approached by a computational device significantly simple in terms of size, structure and implementation.

References

[1] Castellanos, J., Manea, F., Mingo López, L.F. de & Mitrana, V., Accepting Networks of Splicing Processors with Filtered Connections, MCU (2007): 218-229.

[2] Castellanos, J., C. Martín-Vide, V. Mitrana, J.M. Sempere, Solving NP-complet problems with networks of evolutionary processors, in

Mira, J. & A. Prieto (eds.), *IWANN 2001*, LNCS 2084, Springer-Verlag (2001): 621-628.

[3] Castellanos, J., C. Martín-Vide, V. Mitrana & J.M. Sempere, Networks of Evolutionary processors, *Acta Informatica.* 39 (2003): 517-529.

[4] Castellanos, J., Leupold, P. & Mitrana. V., Descriptional and Computational Complexity Aspects of Hybrid Networks of Evolutionary Processors. *Theoretical Computer Science* (2004).

[5] Csuhaj-Varjú, E., Dassow, J., Kelemen, J. & Păun, G., *Grammar Systems*, London, Gordon and Breach (1993).

[6] Csuhaj-Varjú, E., & Mitrana, V., Evolutionary Systems: A Language Generating Device Inspired by Evolving Communities of Cells, *Acta Informatica* 36 (2000): 913-926.

[7] Csuhaj-Varjú, E. & Salomaa, A., Networks of Parallel Language Processors, in Păun, Gh. & A. Salomaa, *New Trends in Formal Languages*, LNCS 1218, Berlin, Springer (1997): 299-318.

[8] Fodor, J., *The Modularity of Mind*, Cambridge (MA), The MIT Press (1983).

[9] 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) (2007): 112-118

[10] Margenstern, M., Mitrana, V., & Perez-Jimenez, M., Accepting Hybrid Networks of Evolutionary Processors, in Ferreti, C., Mauri, G. & Zandron, C., *DNA 10. Preliminary Proceedings*, Milan, University of Milano-Biccoca (2004): 107–117

[11] Martín-Vide, C., Mitrana, V., Perez-Jimenez, M. & Sancho-Caparrini, F., Hybrid Networks of Evolutionary Processors, in *Proceedings of GECCO 2003* (2003): 401-412.

[12] Păun, Gh.: Computing with Membranes. *Journal of Computer and System Sciences*, 61 (2000): 108-143.

[13] Păun, Gh., Rozenberg, G., & Salomaa, A., *DNA Computing. New Computing Paradigms*, Berlin, Springer (1998).