PNEPS FOR SHALLOW PARSING
NEPs Extended For Parsing Applied To Shallow Parsing

Emilio del Rosal, Alfonso Ortega de la Puente
Departamento de Ingeniería Informática de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid, Spain

Diana Perez Marin
Departamento de Lenguajes y Sistemas I, Universidad Rey Juan Carlos, Madrid, Spain

Keywords: Natural computing, Natural language processing, Shallow parsing, Nets of evolutionary processors.

Abstract: PNEPs (Parsing Networks of Evolutionary Processors) extend NEPs with context free (instead of substituting) rules, leftmost derivation, bad terminals check and indexes to rebuild the derivation tree. It is possible to build a PNEP from any context free grammar without additional constraints, able to generate all the different derivations for ambiguous grammars with a temporal performance bound by the depth of the derivation tree. One of the main difficulties encountered by parsing techniques when building complete parsing trees for natural languages is the spatial and temporal performance of the analysis. Shallow parsing tries to overcome these difficulties. The goal of shallow parsing is to analyze the main components of the sentences (for example, noun groups, verb groups, etc.) rather than complete sentences. The current paper is mainly focused on testing the suitability of PNEPs to shallow parsing.

1 MOTIVATION

Syntactic analysis is one of the classical problems related to language processing, and applies both to *artificial* and to *natural* languages.

There is an ample range of parsing tools that computer scientists and linguists can use.

The characteristics of the particular language determine the suitability of the parsing technique. Two of the main differences between natural and formal languages are ambiguity and the size of the required representation. Ambiguity introduces many difficulties to parsing, therefore programming languages are usually designed to be non ambiguous. On the other hand, ambiguity is an almost implicit characteristic of natural languages. To compare the size of different representations, the same formalism should be used. Context-free grammars are widely used to describe the syntax of languages. It is possible to informally compare the sizes of context free grammars for some programming languages and for some natural languages. We conjecture that the representations needed for parsing natural languages are frequently greater than those we can use for high level imperative programming languages.

Parsing techniques for programming languages usually restrict the representation (grammar) used in different ways: it must be unambiguous, recursion is restricted, erasing rules must be removed, they must be written in a normal form, etc. These conditions mean extra work for the designer of the grammar, difficult to understand for non-experts in the field of formal languages. This may be one of the reasons why formal representations such as grammars are little used or even unpopular. Natural languages usually do not fulfill these constraints.

The current paper is focused on formal representations (based on Chomsky grammars) that can be used for syntactic analysis, specially those which do not comply with these kinds of constraints. In this way, our approach will be applicable at the same time for natural and formal languages.

Formal parsing techniques for natural languages are inefficient. The length of the sentences that these techniques are able to parse is usually small (usually less than a typical computer program).

This work is also focused on new models to increase the efficiency of parsing for languages with non-restricted context free grammars.

Conventional computers are based on the well

del Rosal E., Ortega de la Puente A. and Perez-Marin D. (2010).
PNEPS FOR SHALLOW PARSING - NEPs Extended For Parsing Applied To Shallow Parsing.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*, pages 403-410
DOI: 10.5220/0002789004030410

known von Neumann architecture, which can be considered an implementation of the Turing machine. One of the current topics of interest in Computer Science is the design of new abstract computing devices that can be considered alternative architectures for the design of new families of computers and algorithms. Some of them are inspired by the way in which Nature efficiently solves difficult tasks; almost all of them are intrinsically parallel. They are frequently called *natural or unconventional computers*. Nets of Evolutionary Processors (NEPs) are one of these massively parallel new natural computers. Their structure will be described later.

The authors have previously proposed PNEPs: an extension to NEPs that makes them suitable for efficient parsing of any kind of context free grammars, specially applicable to those languages that share characteristics with natural languages (inherent ambiguity, for example). The goal of the current paper is to modify and use PNEPs for shallow parsing. Shallow parsing will be described later. It is a parsing technique frequently used in natural language processing to overcome the inefficiency of other approaches to syntactic analysis.

Some of the authors of this contribution have previously developed IBERIA, a corpus of scientific Spanish which is able to process the sentences at the morphological level.

We are very interested in adding syntactic analysis tools to IBERIA. The current contribution has this goal.

In the following sections we will introduce all the areas involved in this work (syntactic analysis, natural languages, NEPs, PNEPs, jNEP and shallow parsing). Then we will introduce FreeLing, a well-known free platform that offers parsing tools such as a Spanish grammar and shallow parsers for this grammar. Then we will describe how PNEPs can be used for shallow parsing and describe a jNEP implementation. Finally some examples will be given, and conclusions and further research lines are discussed.

2 INTRODUCTION

2.1 Introduction to NEPs

Networks of evolutionary processors (NEPs (Castellanos et al., 2003)) are a new computing mechanism directly inspired in the behaviour of cell populations. Each cell contains its own genetic information (represented by a set of strings of symbols) that is changed by some *evolutive* transformations (implemented as elemental operations on strings). Cells are interconnected and can exchange information (strings) with other cells.

The Connection Machine (Hillis, 1985), the Logic Flow paradigm (Errico and Jesshope, 1994) and the biological background of DNA computing (Paun et al., 1998), membrane computing (Paun, 2000), and specially the theory of grammar systems (Csuhaj-Varjú et al., 1993) can be considered precedents to NEPs.

A NEP can be defined as a graph whose nodes are processors which perform very simple operations on strings and send the resulting strings to other nodes. Every node has filters that block some strings from being sent and/or received.

Parsing NEPs (PNEPs) were introduced in (Ortega et al., 2009) to handle context free grammars.

2.1.1 NEPs and PNEPs: Definitions and Key Features

Following (Castellanos et al., 2003) we introduce the basic definition of NEPs.

Definition. 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 just one of the following forms:
    i. $a \to b$, where $a, b \in V$ (substitution rules),
    ii. $a \to \varepsilon$, where $a \in V$ (deletion rules),
    iii. $\varepsilon \to a$, where $a \in V$ (insertion rules),
    iv. $r : A \to s$, where $s \in V^*$ (context free rules applied to change a symbol by a string) PNEPs replace substitution rules by this kind of rules.
    v. $r : A \to_l s$, where $s \in V^*$ (context free rules applied to the leftmost non terminal) PNEPs add this kind of rule to reduce the amount of equivalent derivations.
  - 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^* respectively representing the input and the output filters. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i (w \in PO_i)$. In this paper we will use two kind of filters:

  - ∗ Those defined as two components (P,F) of *P*ermitting and *F*orbidding contexts (a word w passes the filter if (alphabet of $w \subseteq P) \wedge (F \cap$ alphabet of $w = \emptyset)$).
  - ∗ Those defined as regular expressions r (a word w passes the filter if $w \in L(r)$, where $L(r)$ stands for the language defined by the regular expression r).
- $G = (\{N_1, N_2, \ldots, N_n\}, E)$ 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.
- The algorithm to build PNEPs from context free grammars imposes a standard structure to PNEPs (concerning the topology of the graph and the types of nodes needed). Further details can be found in (Ortega et al., 2009).

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 means of a communication step, each node processor N_i sends all the copies of the strings it has, able to pass its output filter, to all the node processors connected to N_i, and receives all the 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 means of a communication step is written as $C_1 \vdash C_2$.

2.2 Introduction to Analysis of Natural Languages with NEPs and PNEPs

Computational Linguistics researches linguistic phenomena that occur in digital data. Natural Language Processing (NLP) is a subfield of Computational Linguistics that focuses on building automatic systems able to interpret or generate information written in natural language (Volk, 2004). Machine Translation was the first NLP application in the fifties (Weaver, 1955).

A typical NLP system has to cover several linguistic levels:

phonological (sound processing), morphological (extracting the part of speech and morphological characteristics of words), semantic-pragmatic (both levels related to the meaning of the sentences) and syntactical.

Our current work is focused on this last level in which parsers are used to detect valid structures in the sentences, usually in terms of a certain grammar.

Syntactical analysis for natural language requires a big amount of computational resources. Parsers usually are able to completely analyze only short sentences. Shallow parsing tries to overcome this difficulty. Instead of a complete derivation tree for the sentence, this parsing technique actually builds partial derivation trees for its elemental components. This paper is focused on this approach.

Typical NLP systems usually cover the linguistic levels previously described in the following way:

OCR/Tokenization ⇒ Morphologycal analysis ⇒
Syntax analysis ⇒ Semantic interpretation ⇒
Discourse text processing

A computational model that can be applied to NLP tasks is PNEPs. PNEPs are an extension to NEPs. NEP as a generating device was first introduced in (Csuhaj-Varju and Salomaa, 1997; Csuhaj-Varjú and Mitrana, 2000). The topic is further investigated in (Castellanos et al., 2001), while further different variants of the generating machine are introduced and analyzed in (Castellanos et al., 2005; Manea, 2004; Manea and Mitrana, 2007; Margenstern et al., 2005; Martin-Vide et al., 2003).

In (Bel Enguix et al., 2009), a first attempt was made to apply NEPs for syntactic NLP parsing. In (Ortega et al., 2009) context free (instead of substituting) rules, leftmost derivation, bad terminals check and indexes to rebuild the derivation tree were added to NEPs.

(Ortega et al., 2009) also proposes an algorithm to build a PNEP from any context free grammar without additional constraints, able to generate all the different derivations for ambiguous grammars with a temporal performance bound by the depth of the derivation tree.

Our current contribution focuses on the use of PNEPs for shallow parsing. It is a mandatory step to compare the performance of PNEPs with other standard tools for the syntactical analysis of natural languages.

2.3 Introduction to jNEP

The jNEP (del Rosal et al., 2008) Java program, freely available at http://jnep.edelrosal.net, can simulate almost every type of NEP in the literature. The software has been developed under three main principles: 1) it rigorously complies with the formal definitions found in the literature; 2) it serves as a general tool, by allowing the use of the different NEP variants and can easily be adapted to possible future extensions of the NEP concept; 3) it exploits the inherent parallel/distributed nature of NEPs.

jNEP consists of three main classes (*NEP*, *EvoluionaryProcessor* and *Word*), and three Java interfaces (*StoppingCondition*, *Filter* and *EvolutionaryRule*) This design mimics the NEP model definition. In jNEP, a NEP is composed of evolutionary processors, stopping conditions and an underlying graph (attribute *edges*), used to define the net topology. Likewise, an evolutionary processor contains a set of rules and filters.

Java interfaces are used for those components which more frequently change between different NEP variants. jNEP implements a wide set of these three components and more can be easily added in the future.

The *NEP* class coordinates the main dynamics of the computation and manages the processors (instances of the *EvolutionaryProcessor* class), forcing them to perform alternate evolutionary and communication steps. Furthermore, the computation is stopped whenever it is needed.

jNEP reads the definition of the NEP from an XML configuration file that contains special tags for any relevant components in the NEP (alphabet, stopping conditions, the complete graph, every edge, the evolutionary processors with their respective rules, filters and initial contents).

Although some fragments of these files will be shown in this paper, all the configuration files mentioned here can be found at (http://jnep.e-delrosal.net). Despite the complexity of these XML files, the interested reader can see that the tags and their attributes have self-explaining names and values.

2.4 Introduction to FreeLing and Shallow Parsing

We can summarize some of the main difficulties encountered by parsing techniques when building complete parsing trees for natural languages:

- Spatial and temporal performance of the analysis. The Early algorithm and its derivatives (Earley, 1970; Seifert and Fischer, 2004; Zollmann and Venugopal, 2006) are one of the most efficient approaches. They, for example, provide parsing in polynomial time, with respect to the length of the input. Its time complexity for parsing context-free languages is linear in the average case, while in the worst case it is n^2 and n^3, respectively, for unambiguous and ambiguous grammars.
- The size and complexity of the corresponding grammar, which is, in addition, difficult to design. Natural languages, for instance, usually are ambiguous.

The goal of shallow parsing is to analyze the main components of the sentences (for example, noun groups, verb groups, etc.) rather than complete sentences. It ignores the actual syntactic structure of the sentences, which are considered just as sets of these basic blocks. Shallow parsing tries to overcome, in this way, the performance difficulties that arise when building complete derivation trees.

Shallow parsing produces sequences of subtrees. These subtrees are frequently shown as children of a fictitious root node. This way of presenting the results of the analysis can confuse the inexperienced reader, because the final tree is not a real derivation tree: neither its root is the axiom of the grammar nor its branches correspond to actual derivation rules.

Shallow parsing includes different particular algorithms and tools (for instance FreeLing (TALP, 2009) or cascades of finite-state automata (Harris, 1962))

FreeLing is *An Open Source Suite of Language Analyzers* that provides the scientist with several different tools and techniques. FreeLing includes a context-free grammar of Spanish, adapted for shallow parsing, that does not contain a real axiom. This grammar has almost two hundred non-terminals and approximately one thousand rules. The actual number of rules is even greater, because they use regular expressions rather than terminal symbols. Each rule, in this way, represents a set of rules, depending on the terminal symbols that match the regular expressions.

The terminals of the grammar are *part-of-speech* tags produced by the morphological analysis. So they include labels like "plural adjective", "third person noun" etc.

Figure 4 shows the output of FreeLing for a very simple sentence like "Él es ingeniero"[1].

FreeLing built three subtrees: two noun phrases and a verb. After that, FreeLing just joins them under the fictitious axiom. Figure 1 shows a more complex example.

3 PNEP EXTENSION FOR SHALLOW PARSING

The main difficulty to adapt PNEPs to shallow parsing is the fictitious axiom. PNEPs is designed to handle context free grammars that must have an axiom.

[1] *He is an engineer*

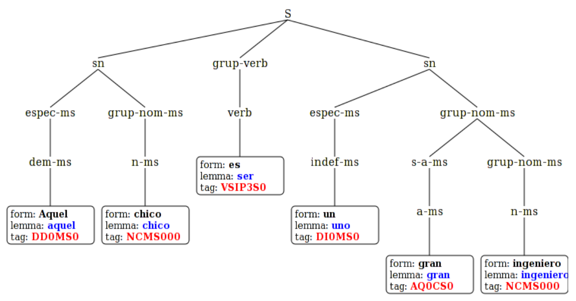

Figure 1: FreeLing output for “Aquel chico es un gran ingeniero” (*That guy is a great engineer*).

We have also found additional difficulties in the way in which FreeLing reduces the number of needed derivation rules of its grammar. As we have previously introduced, FreeLing uses regular expressions rather than terminal symbols. This kind of rules actually represents a set of rules: those whose terminals match the regular expressions. We have also added this mechanism to PNEPs in the corresponding filters that implement the matching.

In the following paragraphs we will explain both problems with more detail.

The virtual root node and the partial derivation trees (for the different components of the sentence) force some changes in the behavior of PNEPs. Firstly, we have to derive many trees at once, one per each constituent, instead of only one tree for the complete sentence. Therefore, all the nodes that will apply derivation rules for the nonterminals associated with the components in which the shallow parser is focused will contain their symbol in the initial step. In (Ortega et al., 2009) the nodes of the axiom were the only non empty nodes. In a more formal way:

- Initially, in the original PNEP (Ortega et al., 2009), the only non empty node is associated with the axiom and contains a copy of the axiom. Formally (N_A and Σ_N stand respectively for the node associated with the axiom and the set of nonterminal symbols of the grammar under consideration)
  $I_{N_A} = A$
  $\forall N_i \in \Sigma_N, i \neq A \rightarrow I_{N_i} = \emptyset$
- The initial conditions of the PNEP for shallow parsing are:
  $\forall N_i, I_{N_i} = i$

In this way, the PNEP produces every possible derivation sub-tree beginning from each nonterminal, as if they were axioms of a virtually independent grammar. However, those sub-trees have to be concatenated and, after that, joined to the same parent node (virtual root node of the fictitious axiom). We get this behavior with splicing rules (Choudhary and Krithivasan, 2005), (Manea and Mitrana, 2007) in the following way: (1) the PNEP marks the end and the beginning of the sub-trees with the symbol %, (2) splicing rules are applied to concatenate couples of sub-trees, taking the beginning of the first one and the end of the second as the splicing point.

To be more precise, a special node is responsible of the first step. Its specification in jNEP is the following:

```
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="insertion" actionType="RIGHT"
                                 symbol="%"/>
    <RULE ruleType="insertion" actionType="LEFT"
                                 symbol="%"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="2" permittingContext=
    "SET_OF_VALID_TERMINALS" forbiddingContext=""/>
    <OUTPUT type="RegularLangMembershipFilter"
            regularExpression="%%.*|%.*%|.*%%"/>
  </FILTERS>
</NODE>
```

During the second step the splicing rules concatenate the sub-trees. We could choose a specialized node (just one node) or a set of nodes depending on the degree of parallelism we prefer. The needed splicing rule could be defined as follows:

```
<RULE ruleType="splicingChoudhary" wordX="terminal1"
             wordY="%" wordU="%" wordV="terminal2"/>
```

Where terminal2 follows terminal1 in the sentence at any place. It should be remembered that % marks the end and beginning of the derivation trees. If the sentence has n words, there are n-1 rules/points for concatenation. It is important to note that only splicing rules that create a valid sub-sentence are actually concatenated. [2]

For example, if the sentence to parse is a_b_c_d, we would need the following rules:

```
<RULE ruleType="splicingChoudhary" wordX="a" wordY="%"
                                   wordU="%" wordV="b"/>
<RULE ruleType="splicingChoudhary" wordX="b" wordY="%"
                                   wordU="%" wordV="c"/>
<RULE ruleType="splicingChoudhary" wordX="c" wordY="%"
                                   wordU="%" wordV="d"/>
```

They could concatenate two sub-sentences like b_c and d, resulting in b_c_d.

[2]In fact, we are using Choudhary splicing rules (Choudhary and Krithivasan, 2005) with a little modification to ignore the symbols that belong to the trace of the derivation.

3.1 Our PNEP for the FreeLing's Spanish Grammar

The jNEP configuration file for our PNEP adapted to the FreeLing's grammar is large. It has almost 200 hundred nodes and some nodes have tens of rules. We will show, however, some of its details. Let the sentence to be parsed be "Él es ingeniero". The output node has the following definition:

```
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
  </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter"
         regularExpression=
         "%[0-9\-]*(PP3MS000|PP\*)[0-9\-]*(VSIP3S0|VSI\*)
                    [0-9\-]*(NCMS000|NCMS\*|NCMS00\*)%"/>
      <OUTPUT type="1" permittingContext=""
              forbiddingContext="PP*_PP3MS000_VSI*_VSIP3S0
                                   _NCMS*_NCMS00*_NCMS000"/>
    </FILTERS>
</NODE>
```

We have previously explained that the input sentence includes part-of-speech tags instead of actual Spanish words. This sequence of tags together with the indexes of the rules that will be used to build the derivation tree, are in the input filter for the output node. We can also see some tags written as regular expressions. We have added this kind of tags because FreeLing uses also regular expressions to reduce the size of the grammar.

As an example, we show the specification of one of the deriving nodes. We can see below that the non-terminal grup-verb has many rules, the one with trace ID 70-7 is actually needed to parse our example.

```
<NODE initCond="grup-verb" id="70">
 <EVOLUTIONARY_RULES>
  <RULE ruleType="leftMostParsing" symbol="grup-verb"
        string="70-0_grup-ve[...]
  <RULE ruleType="leftMostParsing" symbol="grup-verb"
        string="70-1_grup-ve[...]
  <RULE ruleType="leftMostParsing" symbol="grup-verb"
        string="70-7_verb" [...]
   [...]
 </EVOLUTIONARY_RULES>
 <FILTERS>
   <INPUT type="1" permittingContext="grup-verb"
                   forbiddingContext=""/>
 </FILTERS>
</NODE>
```

The output of jNEP is also large. However, we can show at least the main dynamic of the process. Figures 2 and 3 show it. Comments between brackets help to understand it.

As jNEP shows, the output node contains more than one derivation tree. We design the PNEP in this way, because ambiguous grammars have more than one possible derivation tree for the same sentence. In that case, our PNEP will produce all the possible derivation trees, while FreeLing is only able to show the most likely.

```
***************NEP INITIAL CONFIGURATION***************
--- Evolutionary Processor 0 ---
[THE INITIAL WORD OF EVERY DERIVATION NODE IS ITS
             CORRESPONDING NON-TERMINAL IN THE GRAMMAR]
[...]
--- Evolutionary Processor 70 ---
grup-verb
[...]
--- Evolutionary Processor 112 ---
sn
[...]
--- Evolutionary Processor 190 ---
[THE OUTPUT NODE IS EMPTY]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP -
TOTAL STEPS: 1 ***************
[FIRST EXPANSION OF THE TREES]
[...]
--- Evolutionary Processor 70 ---
70-6_verb-pass 70-7_verb
                 70-0_grup-verb_patons_patons_patons[...]
[...]
--- Evolutionary Processor 112 ---
112-104_grup-nom 112-103_grup-nom-ms 112-97_pron-mp
                                     112-95_pron-ns[...]
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP -
TOTAL STEPS:  2 ***************
--- Evolutionary Processor 0 ---
[THE FIRST TREES WITH ONLY TERMINALS APPEAR AT THE
BEGINNING OF SPLICING SUB-NET]
--- Evolutionary Processor 178 ---
57-3_NCMS00* 151-35_VSI* 1-2_PP3MS000 99-0_NCMS* 121-2_VSI*
[...]
[THE REST GO TO THE PRUNING NODE]
--- Evolutionary Processor 189 ---
112-87_psubj-mp_indef-mp 8-3_s-a-ms 44-6_prep_s-a-fp [...]
```

Figure 2: jNEP output for "Él es ingeniero". 1 of 2.

It is easy to realize that figure 4 corresponds also to the output of jNEP running our PNEP for shallow parsing.

4 CONCLUSIONS AND FURTHER RESEARCH LINES

Formal syntactical analysis techniques for natural languages (LL, LR, Early families, for example) suffer from inefficiency when they try to build derivation trees for complete sentences. Shallow parsing is an approach focused on the basic components of the sentence instead on its complete structure. It is extensively used to overcome performance difficulties.

```
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
[THE PROCESS OF MARKING THE END AND THE BEGINNING STARTS]
[...]
--- Evolutionary Processor 178 ---
1-2_PP3MS000_% %_151-35_VSI* 57-3_NCMS00*_% %_1-2_PP3MS000 %_99-0_NCMS* 99-0_NCMS*_% 151-35_VSI*_% 121-2_VSI*_%
%_121-2_VSI* %_57-3_NCMS00*
[...]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
[THE SPLICING SUB-NET STARTS TO CONCATENATE THE SUB-TREES]
[...]
--- Evolutionary Processor 178 ---
156-3_1-2_PP3MS000_% 77-13_57-3_NCMS00*_% %_70-7_151-35_VSI* 34-11_99-0_NCMS*_% %_111-4_1-2_PP3MS000
111-4_1-2_PP3MS000_% 70-7_151-35_VSI*_% %_77-13_57-3_NCMS00* %_34-11_99-0_NCMS* %_156-3_1-2_PP3MS000
[...]
--- Evolutionary Processor 187 ---
%_121-2_VSI*_99-0_NCMS*_% %_% %_151-35_VSI*_% %_99-0_NCMS*_% %_121-2_VSI*_% %_151-35_VSI*_99-0_NCMS*_%
--- Evolutionary Processor 188 ---
%_121-2_VSI*_57-3_NCMS00*_% %_151-35_VSI*_57-3_NCMS00*_% %_% %_151-35_VSI*_% %_121-2_VSI*_% %_57-3_NCMS00*_%
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  18 ***************
[THE OUTPUT NODE RECEIVES THE RIGHT DERIVATION TREE. IT IS THE SAME AS THE ONE OUTPUT BY FREELING]
--- Evolutionary Processor 190 ---
[THE FIRST ONE IS THE OUTPUT DESIRED]
%_112-99_111-4_1-2_PP3MS000_70-7_151-35_VSI*_112-103_77-13_57-3_NCMS00*_% %_1-2_PP3MS000_151-35_VSI*_57-3_NCMS00*_%
[...]
```

Figure 3: jNEP output for “Él es ingeniero”. 2 of 2.

FreeLing is one of the most popular free packages and it includes grammars for different natural languages and shallow parsers for them. Some of the main characteristics of shallow parsing are summarized below:

- It actually builds a set of derivation trees that are shown to the user as if they were sons of a fictitious pseudo axiom that does not belong to the grammar
- It is not a pure formal technique, so several tricks are frequently used to save resources. One of them is the use of regular expressions instead of terminal symbols. Each rule, in this case, represents the set of rules whose terminals match the regular expressions. The morphological analyzers have also to take into account this kind of matching.

We have added to PNEPs (an extensions to NEPs for parsing any kind of context free grammars) some features to deal with these characteristics.

We have also added them to jNEP (a NEP simulator written in Java and able to run on parallel platforms). We have also used the FreeLing grammar for Spanish to shallow parse some very simple examples.

We have shown, in this way, that it is possible to use variants of NEPs for shallow parsing.

In the future we plan to test our proposal with more realistic examples, to improve the accuracy and performance of the basic PNEP model and to incorporate syntactical analysis (both, complete and shallow) to IBERIA corpus for Scientific Spanish by means of PNEPs.

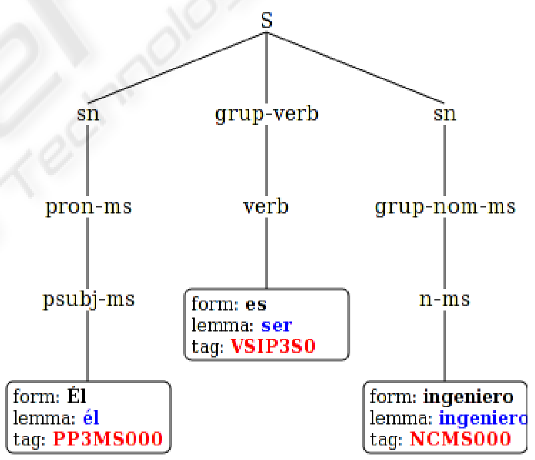

Figure 4: Shallow parsing tree for “Él es ingeniero”.

Further on, we plan to extend PNEPs with formal representations able to handle semantics (attribute grammars, for example) We also plan to use this new model as a tool for compiler design and as a new approach to tackle some tasks in the semantic level of natural language processing.

ACKNOWLEDGEMENTS

This work was partially supported by MEC, project TIN2008-02081/TIN and by DGUI CAM/UAM, project CCG08-UAM/TIC-4425. The authors thank Dr. Manuel Alfonseca for his help while preparing this document.

REFERENCES

Bel Enguix, G., Jimenez-Lopez, M. D., Mercaş, R., and Perekrestenko, A. (2009). Networks of evolutionary processors as natural language parsers. In *Proceedings ICAART 2009*.

Castellanos, J., Leupold, P., and Mitrana, V. (2005). On the size complexity of hybrid networks of evolutionary processors. *THEORETICAL COMPUTER SCIENCE*, 330(2):205–220.

Castellanos, J., Martin-Vide, C., Mitrana, V., and Sempere, J. M. (2003). Networks of evolutionary processors. *Acta Informatica*, 39(6-7):517–529.

Castellanos, J., Martn-Vide, C., Mitrana, V., and Sempere, J. M. (2001). Solving np-complete problems with networks of evolutionary processors. In *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence : 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15, 2001, Proceedings, Part I*, pages 621–.

Choudhary, A. and Krithivasan, K. (2005). Network of evolutionary processors with splicing rules. *MECHANISMS, SYMBOLS AND MODELS UNDERLYING COGNITION, PT 1, PROCEEDINGS*, 3561:290–299.

Csuhaj-Varjú, E., Dassow, J., Kelemen, J., and Paun, G. (1993). *Grammar Systems*. London, Gordon and Breach.

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

Csuhaj-Varju, E. and Salomaa, A. (1997). *Lecture Notes on Computer Science 1218*, chapter Networks of parallel language processors.

del Rosal, E., Nuez, R., Castaeda, C., and Ortega, A. (2008). Simulating neps in a cluster with jnep. In *Proceedings of International Conference on Computers, Communications and Control, ICCCC 2008,*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Errico, L. and Jesshope, C. (1994). Towards a new architecture for symbolic processing. In Plander, I., editor, *Artificial Intelligence and Information-Control Systems of Robots '94*. Singapore, World Sci. Publ.

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.

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

Manea, F. (2004). Using ahneps in the recognition of context-free languages. In *In Proceedings of the Workshop on Symbolic Networks ECAI*.

Manea, F. and Mitrana, V. (2007). All np-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. *Information Processing Letters*, 103(3):112–118.

Margenstern, M., Mitrana, V., and Perez-Jimenez, M. J. (2005). Accepting hybrid networks of evolutionary processors. *DNA COMPUTING*, 3384:235–246.

Martin-Vide, C., Mitrana, V., Perez-Jimenez, M. J., and Sancho-Caparrini, F. (2003). Hybrid networks of evolutionary processors. *Genetic and Evolutionary Computation. GECCO 2003, PT I, Proceedings*, 2723:401–412.

Ortega, A., del Rosal, E., Prez, D., Merca, R., Perekrestenko, A., and Alfonseca, M. (2009). *PNEPs, NEPs for Context Free Parsing: Application to Natural Language Processing*, chapter Bio-Inspired Systems: Computational and Ambient Intelligence, pages 472–479. LNCS.

Paun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143.

Paun, G., Rozenberg, G., and Salomaa, A. (1998). *DNA Computing. New Computing Paradigms*. Berlin, Springer.

Seifert, S. and Fischer, I. (2004). *Parsing String Generating Hypergraph Grammars*. Springer.

TALP (2009). http://www.lsi.upc.edu/ nlp/freeling/.

Volk, M. (2004). *Introduction to Natural Language Processing,*. Course CMSC 723 / LING 645 in the Stockholm University, Sweden.

Weaver, W. (1955). *Translation, Machine Translation of Languages: Fourteen Essays*.

Zollmann, A. and Venugopal, A. (2006). Syntax augmented machine translation via chart parsing. In *Procceedings of the Workshop on Statistic Machine Translation*. HLT/NAACL.