RuDriCo2 - a faster disambiguator and segmentation modifier

Cláudio Diniz, Nuno Mamede, João D. Pereira

IST – Instituto Superior Técnico
L^{2}F – Spoken Language Systems Laboratory – INESC ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{Cdiniz,Nuno.Mamede,Joao}@inesc-id.pt

Abstract. Currently, L^2F's NLP chain has a bottleneck. Module RuDriCo (Rule Driven Converter) is substantially slower than the remaining modules of the chain. RuDriCo is a rule-based morphological disambiguator with the possibility to change segmentation (join or split tokens). This paper describes the changes made to the system to improve its performance by using the concept of layers and also by reducing the number of variables contained in the rules. It also describes the changes in rule syntax, such as the addition of new operators and contexts, which makes the rules more expressive.

Resumo. Actualmente, a cadeia de PLN do L^2F tem um módulo que é substancialmente mais lento que os outros, o RuDriCo. O RuDriCo é um desambiguador morfológico baseado em regras que também permite alterar a segmentação de texto. Este trabalho descreve os melhoramentos realizados, nomeadamente a introdução de novos operadores, a introdução do conceito de camada e a redução do número de variáveis usadas na especificação das regras.

1 Introduction

Natural Language Processing (NLP) is one of the most important Artificial Intelligence research areas. Many of the systems developed in this area, such as dialog systems or spelling correction systems, use a set of modules responsible for processing text. Usually such systems are organized in a pipeline and are referred to as *NLP chain.* Currently, the L^{2}F[1] research group uses a NLP chain (see Figure 1) to identify and classify Named Entities, extract semantic relations between those entities, to mention only a few. RuDriCo is one of the modules of the L^{2}F NLP chain.

The L^{2}F NLP chain is organized as follows. The first module receives the text to process and tokenizes it, defining the segments that compose the text. Palavroso [Medeiros, 1995] is a morphological tagger that receives the result of this segmentation as input and associates all possible part-of-speech (POS) tags

[1] Spoken Language Systems Laboratory of INESC-ID Lisboa.

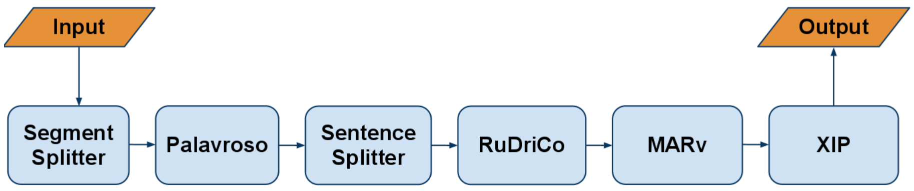


Fig. 1. L^{2}F's NLP chain

to each segment. The next module groups the segments into sentences. The next module to apply is RuDriCo [Pardal, 2007]. This module is a rule-based morphological disambiguator and it also makes segmentation changes to the input, like joining segments (compound words). MARv [Ribeiro et al., 2003], a stochastic morphological disambiguator, receives the result of RuDriCo and it selects the best POS tag to each segment. Finally, the last module to apply is XIP [Xerox, 2003] which is responsible for the syntactic analysis.

Disambiguation systems based on rules, also known as systems with linguistic knowledge [Màrquez and Padró, 1997], are the target of this study. The rules used in these systems are written by linguists. The rules consider the context of each word, and depending on the context make their disambiguation. This kind of methodology leaves some ambiguities unresolved, but is still common that current systems have an accuracy rate around 99%[2].

The input of RuDriCo is a set of rules and the text to process. Input text is in XML format and consists in a set of sentences where each sentence has one or more segments. The segments represent words that are constituted by a surface (`word`) and one or more annotations (`class`). An annotation is composed by a lemma (`root`) and a set of attribute-value pairs. The attribute-value pairs represent the properties of each annotation, e.g. the category of a word. For example, Figure 2 represents an ambiguous segment containing the word *partido*: it has one surface and three annotations.

RuDriCo has two types of rules: disambiguation and segmentation rules. The former ones allow the system to choose the correct category of a word by considering the surrounding context. Segmentation rules change the segmentation and can be divided into contraction and expansion rules. Contraction rules convert two or more segments into a single one. Expansion rules transform a segment into at least two segments. An example of an expansion rule is to transform the segment "Na" into two segments "Em" and "a". An example of a contraction rule is to turn segments "Coreia", "do" and "Sul" into a single segment "Coreia do Sul".

In the original RuDriCo, all types of rules share the same syntax:

```
antecedent --> consequent .
```

where the `antecedent` defines the conditions that must exist to perform the action specified in the `consequent`. In other words, RuDriCo tries to pair the

[2] The hit rate does not take into account the words that are not disambiguated.

```
<sentence>
  <word name="partido">
   <class root="partido">
     <id atrib="CAT" value="nou"/>
     <id atrib="SCT" value="com"/>
     <id atrib="NUM" value="s"/>
     <id atrib="GEN" value="m"/>
     </class>
   <class root="partido">
     <id atrib="CAT" value="adj"/>
     <id atrib="NUM" value="s"/>
     <id atrib="GEN" value="m"/>
   </class>
   <class root="partir">
     <id atrib="CAT" value="ver"/>
     <id atrib="MOD" value="par"/>
     <id atrib="NUM" value="s"/>
     <id atrib="GEN" value="m"/>
   </class>
  </word>
</sentence>
```

Fig. 2. The word "partido" represented in XML

`antecedent` with a sequence of segments from the XML input, and when it succeeds, that sequence of segments is replaced by the segments described in the `consequent`. The segment syntax is as follows:

```
'surface' ['lemma', 'prop_1'/'value1', 'prop_2'/'value2' ... ]
```

where `surface` and `lemma` are obligatory.

Figure 3 contains an example of a contraction rule that transforms the segments "Coreia", "do", and "Sul", in one segment with a single annotation.

```
'coreia' [L1,'CAT'/C1]
'do' [L2,'CAT'/C2]
'sul' [L3,'CAT'/C3]
-->
'Coreia do Sul' ['Coreia do Sul',
'CAT'/'nou','GEN'/'f','NUM'/'s'].
```

Fig. 3. Rule to join segments "Coreia", "do" and "Sul"

The RuDriCo main algorithm (see Figure 4) processes each sentence, segment by segment. A sentence is declared processed when the algorithm cannot apply any rule to it. When a rule is applied to a set of segments, the sentence is processed again to see if there is a new rule that can be applied. The Agenda algorithm (step 4) applies rules to input segments and is not explained since it falls out of the scope of this paper.

RuDriCo has 3 main disadvantages: (i) it is not sufficiently expressive: it does not have neither the "not" nor the "or" operator; (ii) it is not sufficiently

```
1: FOR EACH sentence S in text DO
2:    FOR each segment I in S DO
3:       agenda(I)
4:       IF (agenda(I) has applied a rule) THEN
5:          I = first segment of S
6:          GOTO 3:    /*first segment*/
7:       ELSE
8:          GOTO 2:    /*next segment*/
9:    ENDFOR
10: ENDFOR
```

Fig. 4. RuDriCo simplified algorithm.

efficient: it is the slowest module of the NLP chain; and (iii) it enters in infinite recursion whenever the antecedent of one rule matches the consequent of another rule, and the consequent of the first rule matches the antecedent of the second rule.

The goal of this work is to develop RuDriCo2, a faster RuDriCo with a more expressive and user-friendly syntax, and that avoids infinite recursion. This paper describes the improvements introduced in RuDriCo to implement RuDriCo2.

2 State of the Art

The morphological disambiguators can be classified according to the methodology that is used to solve the problem. [Cole et al., 1995] classifies these systems in two types:
- based on rules, where the knowledge (rules) is manually coded;
- stochastic, where the knowledge is automatically extracted from a previously manually annotated corpora.

Other authors classify these systems differently. For example, [Schmid, 1994b], [Schmid, 1994a] and [Schulze et al., 1994] classify these type of systems using different categories, based on neural networks. In this document, we will just consider Cole's classification. L^2F's NLP chain uses both types of morphological disambiguators: RuDriCo is based on rules and MARv is stochastic. Some of the most well known-rule based systems are:
- Computational Grammar Coder (CGC) [Klein and Simmons, 1963],
- TAGGIT [Greene and Rubin, 1962],
- EngCG [Voutilainen, 1995b] [Voutilainen, 1995a],
- Brill Tagger [Brill, 1992],
- XIP [Xerox, 2003],
- RuDriCo [Pardal, 2007].

CGC is a morphological analyzer and a disambiguator. It begins by addressing some exceptions which the morphological analyzer cannot deal, with a lexicon of 1500 words. After the morphological analyzer, the rule-based disambiguation starts with about 500 rules. TAGGIT is based on the CGC and uses a larger vocabulary.

EngCG is not only a disambiguator, but it also performs some extra tasks such as the segmentation of the input text. The task sequence is the following: (i) segmentation; (ii) morphological analysis; (iii) morphological disambiguation; (iv) find other syntax tags; and (v) finite-state syntactic disambiguation. The morphological disambiguation task is seen as a set of rules. Each rule specifies one or more contexts where a label is false. A tag will be removed if a pattern is established. If a word has a single tag, the word is not ambiguous. This system leaves 3-7% of ambiguous words but their accuracy rate is 99.7%.

The system described in [Brill, 1992] is a morphological analyzer, but when it assigns tags to words, the context is analyzed. This system uses automatically learned rules to associate tags with the input text words. One of the drawbacks of rule-based systems is the need of human experts and linguists for the complex and time-consuming task of writing rules, but [Brill, 1992] shows that this effort can be reduced. The system begins by assigning the most likely tag to each word ignoring the context. Then it performs the learning task, which considers eight types of predefined rules. The system instantiates them and chooses the rules that have a lower error rate. After the rules are chosen, they are applied to the text. The author claims that this system can get better results if some rules are manually written.

The last system here considered is the XIP system [Xerox, 2003], which includes modules to perform morphological disambiguation, syntactic analysis and changes to the hierarchy of segments. Section 2.1 compares XIP and RuDriCo.

2.1 Comparing RuDriCo and XIP

In RuDriCo, the input data is a list of segments, but in XIP, it is a hierarchy of nodes. XIP has disambiguation rules, but it does not have contraction or expansion rules. XIP chunking rules include the following two types: sequence rules and immediate dominance rules. The sequence rules do something similar to a contraction, grouping several nodes into a new node that is added to a tree hierarchy. The difference between immediate dominance and sequence rules is that immediate dominance rules can not represent any order between the antecedent nodes. XIP still has two other types of rules that are not mentioned here since they fall out of the scope of this paper.

Table 1 summarizes the features of XIP and RuDriCo. As it can be seen, XIP does not have contraction rules, having, however sequence rules that change the hierarchical structure instead of the segmentation.

The syntax of a disambiguation rule in XIP is the following:

```
1> noun,verb = |det| noun |verb|
```

The number at the beginning of a rule is the rule layer. The rules are applied according to the layers they belong, starting with the rules of the layer with the lowest number. The rules which do not have a layer are placed in the higher

Features	RuDriCo	XIP
Disambiguation rules	x	x
Contraction rules	x	
Expansion rules	x	
Chunking rules		x
or operator		x
not operator		x

Table 1. Features of RuDriCo and XIP systems

priority layer, the layer number zero. The rule antecedent (`noun,verb`) indicates that there must be a segment with two annotations, a noun and a verb. The two sections between pipes (`|det|` and `|verb|`) are the contexts of the rule, the left context and the right context. The contexts mean that the segment that matches with the antecedent has to have a `det` before and a `verb` after. The rule consequent is the part between the contexts (`noun`), and it indicates which category should be chosen from the antecedent. In this example, the word is disambiguated to noun. This rule can be written in RuDriCo's syntax as the rule in Figure 5.

```
S0[L0,'CAT'/'det']
S1[L11,'CAT'/'nou'][L12,'CAT'/'ver']
S2[L2,'CAT'/'ver']
 -->
S0*
S1[L11,'CAT'/'nou']+
S2* .
```

Fig. 5. Disambiguation rule

Comparing the syntax of both systems, one concludes that XIP's rules are much more compact than RuDriCo's rules. In RuDriCo the lemma and the surface are always present in each item, but in XIP the surface and the lemma can be omitted. Rules do not always need to use the lemma, nor a surface, as the rule presented above. In RuDriCo, the way to ignore the lemma and the surface is by using variables (`S0`, `S1`, `S2`, `L0`, `L11`, `L2`, and `L12` in this example). When a rule does not want to change a segment surface, a variable must be used in the antecedent and the `*` operator on the consequent. This is a disadvantage compared to XIP, because the use of variables requires more computation and the rules are more complex. In conclusion, the syntax of RuDriCo is more complex than the syntax of XIP, less compact and less expressive (RuDriCo does not have logical operators).

3 Layers

In RuDriCo, all rules are stored into a single file and are considered at the same time by the rule matching algorithm. The rules are tested in the order they appear in this file. As an example, consider that the rules are organized as follows: first the expansion rules, then the contraction rules and at the end of the file the disambiguation rules. Then, expansion rules have an higher priority than any other type of rule, because they are placed at the beginning of the rule file.

Instead of loading a file with all the rules, RuDriCo2 loads a file with the filenames of the files that contain the rules. The layers of the rules are relative to the file they belong to. All layers of the first file have priority over the layers of the following files, regardless of their numbers. The number that represents the layer is only used to sort layers on that file.

To support the concept of layers, the rule processing algorithm, presented in Figure 4, was extended with a new cycle that goes through all the layers. This new cycle was added between steps 1 and 2.

Although adding complexity to the algorithm, the agenda algorithm solely runs with the rules of one layer at a time. The performance of the RuDriCo algorithm improves when the gain in the Agenda algorithm is larger than the loss of having an additional cycle.

Layers can also be used to solve the problem of recursion between rules. If the rules that generate recursion are placed in distinct layers then the recursion is avoided.

4 Syntax

Because the syntax of RuDriCo is not expressive enough to express linguistic knowledge, RuDriCo2 has an expanded syntax. The syntax is based on RuDriCo's original syntax, and the changes were made incrementally. The changes to the syntax of RuDriCo are:

- node description (Section 4.1);
- contexts (Section 4.2);
- new operators (Section 4.3);

4.1 Node description

In RuDriCo, when an item is described in a rule, the surface and the lemma are always present, even when their values are irrelevant. For instance, the rule specified in Figure 5 uses three variables (`L0`, `L12` and `L2`) that are not used in the consequent. The use of variables for this purpose can be avoided if the lemma and the surface become explicit attribute-value pairs. So, the properties "lemma" and "surface" are introduced.

The second change in the syntax to describe the attribute-value pairs consists in the absence of quotes (′) around the property names. Figure 6 contains an example of the rule presented in Figure 5 in the new syntax. The surface property can only be used once in each item and the lemma can only be used once in each annotation (an annotation is represented between brackets). Notice that the rules can be represented on a more compact way in the new version.

```
[surface=S0,CAT='det'],
[CAT='nou'][CAT='ver'],
[surface=S2,CAT='ver']
-->
S0*
[CAT='nou']+
S2* .
```

Fig. 6. Disambiguation rule with new syntax

4.2 Contexts

Many of disambiguation rules use variables to simulate contexts, such as the rule shown in Figure 6. To avoid the use of variables for this purpose and to simplify the writing of the rules, contexts (composed by items) were introduced in the antecedent:

$$| \text{ left context } | \text{ Item}_1 \text{ Item}_2 \text{ ... Item}_N \text{ } | \text{ right context } |$$

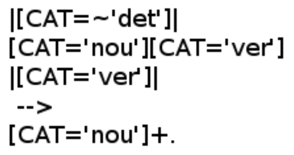
Figure 7 contains the rule presented in Figure 6 rewritten with the new syntax. The use of the contexts turns the rules less extensive since there is no need to use variables to simulate contexts.

```
|[CAT='det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
-->
[CAT='nou']+.
```

Fig. 7. Disambiguation rule with contexts

4.3 New Operators

In RuDriCo the negation operator has not been implemented, although it can be simulated by rule replication (when the negation is applied to properties with a finite set of possible values). For example, when the left context of the rule presented in Figure 7 is a token with any category except determinant, in RuDriCo it is necessary to spell out as many rules as the number of categories, except the one related to the determinant category. In RuDriCo2, the negation



```
|[CAT=~'det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

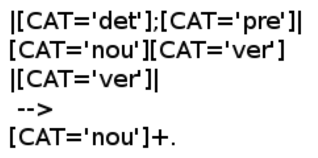
Fig. 8. Example of operator negation

operator (∼) has been introduced and it allows for the specification of this type of condition with a single rule as shown in Figure 8.

The lack of a disjunction operator is a similar problem. If it is necessary to make a disjunction between two values of a property, two rules have to be written, one for each value. For instance, imagine that in the rule of Figure 7, it is desired that the left context is a determinant or a preposition. In RuDriCo, two almost identical rules had to be written (see Figure 9). But in RuDriCo2, with the disjunction operator (;), this situation can be solved with a single rule, as it is shown in Figure 10.

```
|[CAT='det']|                  |[CAT='pre']|
[CAT='nou'][CAT='ver']         [CAT='nou'][CAT='ver']
|[CAT='ver']|                  |[CAT='ver']|
 -->                            -->
[CAT='nou']+.                  [CAT='nou']+.
```

Fig. 9. Two rules to make a disjunction



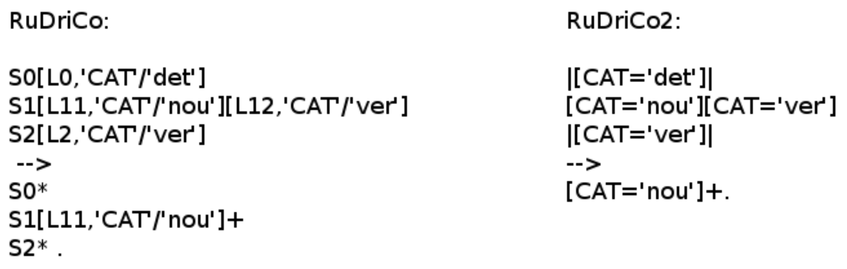
```
|[CAT='det'];[CAT='pre']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
 -->
[CAT='nou']+.
```

Fig. 10. Example of disjunction

5 Evaluation

The evaluation of the syntax may be subjective, but Figure 11 shows that the same rule can be written in a more compact way. In segmentation rules, the number of characters in the new RuDriCo is 16% smaller than in the original RuDriCo. In disambiguation rules, the number of characters is 76.1% smaller than on the original. The improvement is much higher in disambiguation rules since they use more contexts and the disjunction operator.

The performance can be measured running the original RuDriCo and RuDriCo2 with the same input (set of rules and text input files). The performance of XIP was not compared with RuDriCo2's because it is difficult to convert the rules from one system to the other, and because some of the rules are not convertible since both systems have different expressive sintaxes.



```
RuDriCo:

S0[L0,'CAT'/'det']
S1[L11,'CAT'/'nou'][L12,'CAT'/'ver']
S2[L2,'CAT'/'ver']
 -->
S0*
S1[L11,'CAT'/'nou']+
S2* .
```

```
RuDriCo2:

|[CAT='det']|
[CAT='nou'][CAT='ver']
|[CAT='ver']|
-->
[CAT='nou']+.
```

Fig. 11. Comparision between RuDriCo's and RuDriCo2's syntax

To test the performance, a set of 3096 rules was used with a set of text input files from CETEMPúblico[3], each one with a different size. The smallest file has only one sentence and the largest one has 50.000 sentences. Since changes were made incrementally, we have also evaluated an intermediate system, the RuDriCo with layers to assess the impact resulting from the introduction of layers.

The first performance evaluation aimed at discovering the optimal number of rules per layer. This study was done right after the implementation of layers. Since the disambiguation rules must remain on a single layer, only the other rules are used in this evaluation. The remaining rules (2330 rules) are divided into layers of equal size in order to find the optimal size of the layers. The tests were performed using an input text file with 1000 sentences and the results are shown in Table 2.

On one hand, when all rules (2330 rules) are kept in a single layer, which is the behavior of the original RuDriCo, the new RuDriCo spends 15.232 CPU seconds to process all the 1000 sentences. On the other hand, if there are 2330 layers, one rule per layer, then the system takes much longer to process them because the complexity of the algorithm for rule application depends on the number of layers. Results show that the use of layers improves the performance if each layer contains more than 32 rules. For the present set of rules and structure of the algorithm the optimal number of rules per layer is 167.

To evaluate the impact of the new syntax, the performance is measured comparing RuDriCo (the original), with RuDriCo with layers and the RuDriCo with

[3] Corpus with electronic extracts from the Público newspaper.

Rules/Layer	1	2	4	8	16	32	73	146	156	167	180	292	583	1165	2330
Time (s)	146	75.0	40.5	23.2	15.8	9.1	7.7	6.9	6.7	6.1	6.9	7.8	8.7	14.9	15.2

Table 2. Optimal number of rules per layer study

all features described in this paper, RuDriCo2[4]. The comparison is presented in Table 3.

The two smaller input files present only a small improvement because the system spends more time loading the rules than processing the input. For a text file with 1000 sentences, RuDriCo with layers needs 38,6% less time, and RuDriCo2 only needs 19.6% of the original time. To conclude, the new RuDriCo2 is about five times faster than RuDriCo in most cases.

Sentences per file	RuDriCo	RuDriCo with layers	RuDriCo2	% time of RuDriCo2 (comparing to RuDriCo)
1	0.15	0.19	0.11	73.3 %
10	0.20	0.74	0.18	91.8 %
100	8.33	3.36	1.69	20.0 %
500	38.00	15.37	7.83	20.1 %
1000	78.00	30.70	15.29	19.6 %
5000	392.00	152.19	76.80	19.6 %
10000	782.75	301.50	154.12	19.7 %
50000	can't process	1546.70	791.00	-

Table 3. Performance comparision

6 Future work and Conclusion

The experiments made show that RuDriCo2 is five times faster than RuDriCo. However, its efficiency can still be improved. The task that RuDriCo2 has to perform more times is the comparison between items of the rules and segments from the text input. This can be optimized using arrays of bits to represent the segments and the text input restrictions. So to test if an item pairs with a segment, the system will only have to perform a logic operation. The representation of items and segments in arrays of bits is the next scheduled improvement.

RuDriCo takes part of the L2F NLP chain, used to process text and it is the bottleneck. Besides this performance problem, RuDriCo does not support an expressive rule specification language. Is this paper we showed the changes performed in RuDriCo to address these issues. The layers and contexts make RuDriCo2 about five times faster than the original RuDriCo for the current set

[4] The original rules were automatically converted to the new syntax and the results are the same.

of rules. The addition of disjunction and negation operators makes the syntax of RuDriCo2 more expressive than the RuDriCo. The addition of contexts and the new node description allow RuDriCo2 rules to become more compact and easier to write. To conclude, RuDriCo2 is a significant improvement over the original module.

References

[Brill, 1992] Brill, E. (1992). A simple rule-based part of speech tagger. In *Proc. of the third conference on Applied natural language processing*, pages 152–155, Morristown, NJ, USA. Association for Computational Linguistics.

[Cole et al., 1995] Cole, R. A., Mariani, J., Uszkoreit, H., Zaenen, A., and Zue, V. (1995). Survey of the State of the Art in Human Language Technology, Center for Spoken Language Understanding CSLU, Carnegie Mellon University, Pittsburgh, PA.

[Greene and Rubin, 1962] Greene, B. B. and Rubin, G. M. (1962). Automatic Grammatical Tagging of English. Technical Report, Brown University, Providence, RI.

[Klein and Simmons, 1963] Klein, S. and Simmons, R. F. (1963). A Computational Approach to Grammatical Coding of English Words. In *Journal of the Association for Computational MAchinery (10)*, pages 334–347.

[Medeiros, 1995] Medeiros, J. C. (1995). Processamento Morfológico e Correcção Ortográfica do Português. Master's thesis, IST - Univ. Técnica de Lisboa, Portugal.

[Màrquez and Padró, 1997] Màrquez, L. and Padró, L. (1997). A Flexible POS Tagger Using an Automaticalluy Acquired Language Model. In *Proc. of the 35th Annual Metting of the Association for Computational Linguistics*, pages 238–245, Madrid.

[Pardal, 2007] Pardal, J. (2007). Manual do Utilizador do RuDriCo. Technical report, Instituto Superior Técnico - Universidade Técnica de Lisboa, Portugal.

[Ribeiro et al., 2003] Ribeiro, R., Mamede, N. J., and Trancoso, I. (2003). *Computational Proc. of the Portuguese Language: 6th Intern. Workshop, PROPOR 2003, Faro, Portugal, June 26-27, 2003*, volume 2721, Using Morphossyntactic Information in TTS Systems: Comparing Strategies for European Portuguese. Springer.

[Schmid, 1994a] Schmid, H. (1994a). Part-of-Speech Tagging with Neural Networks. In *Proc. of the 15th Inter. Conf. on Computational Linguistics*, Kyoto, Japão.

[Schmid, 1994b] Schmid, H. (1994b). Probabilistic Part-of-Speech Tagging using Decision Trees. In *Proceedings of the 15th International Conference on new methods in language processing*, Manchester, Reino Unido.

[Schulze et al., 1994] Schulze, B. M., Heid, U., Schmid, H., Schiller, A., Rooth, M., Grefenstette, G., Gaschler, J., Zaenen, A., and Teufel, S. (1994). Decide. MLAP-Project 93-19 D-1b I, STR and RXRC.

[Voutilainen, 1995a] Voutilainen, A. (1995a). *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*, chapter Morphological Disambiguation. Mouton de Gruyter.

[Voutilainen, 1995b] Voutilainen, A. (1995b). A systax-based par-of-speech analyser. In *Proceedings of 7th Conference of the European Chapter of The Association for Computational Linguistics*, Dublin.

[Xerox, 2003] Xerox (2003). Xerox Incremental parser – Reference Guide.