TÉCNICO LISBOA

# Suffix Identification in Portuguese using Transducers

**Hugo Miguel Correia de Almeida**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisors: Prof. Doutor Nuno João Neves Mamede
Prof. Doutor Jorge Manuel Evangelista Baptista

### Examination Committee

Chairperson: Prof. Doutor João Emílio Segurado Pavão Martins
Supervisor: Prof. Doutor Nuno João Neves Mamede
Member of the Committee: Prof. Doutor João Carlos Serrenho Dias Pereira

**November 2016**

**Abstract**

STRING is a Natural Language Processing (NLP) chain developed at $L^2F$/INESC-ID Lisboa, and it is currently capable of identifying both prefixed and non-affixed (base) words. This MsC dissertation tackles the problem of automatic identification of suffixed words (e.g. 'cat + inho', 'little cat') in texts as well as some improvements in category guessing for unknown words.

A Suffixed Words Generator was developed, functioning as a new submodule of LexMan, the transducer-based morphological analyzer of STRING. The architecture here developed is based on a Two-Level Morphology approach, which involves combining a lexicon of lemmas and the set of corresponding flexional paradigms, which, together generate suffixed words.

To date, seven of the most productive suffixes in Portuguese have been described (diminutive suffixes *-inho* and *-ito*, superlatives *-íssimo* and *-érrimo*, adverbial suffix *-mente*, adjectival suffix *-vel* and corresponding nominal suffix '-bilidade').

This work also tackles the correct guessing of grammatical category for unknown words, that is, words that do not exist in the lexicon. The existing Guesser module was modified to specifically address two types of unknown words: name-name compound words and diacritical errors.

The performance of both solutions was then evaluated and compared to the original performance of LexMan. The new Suffixed Word Generator module only adds a small static overhead to the transducer operations, barely having any impact in the rate of words processed by second. This small loss in performance is attenuated by the capability to identify almost 500,000 new words. The modified Guesser achieved an accuracy of 87%.

**Keywords:** Natural Language Processing; Part-of-Speech Tagging; Suffixation; Transducers; Morphological Analysis.

**Resumo**

A STRING é uma cadeia de Processamento de Língua Natural desenvolvida pelo $L^2F$/INESC-ID Lisboa, capaz de indentificar palavras prefixadas e palavras base (sem afixos). Esta dissertação resolve o problema da identificação automática de palavras sufixadas (ex. 'gatinho'='gato'+'inho') em textos e oferece também alguns melhoramentos na categorização de palavras desconhecidas.

Foi desenvolvido um Gerador de Palavras Sufixação, que funciona como um novo submódulo do LexMan, o anlisador morfológico da STRING baseado em transdutores. A arquitetura deste submódulo é baseada numa abordagem de Mofologia em Dois Níveis, que envolve combinar um léxico de lemas com um conjunto de paradigmas flexionais e, em conjunto, gerar palavras sufixadas.

Até agora foram desenvolvidos paradigmas para sete dos sufixos mais produtivos no Português: os sufixos diminutivos *-inho* e *-ito*, os superlativos *-íssimo* e *-érrimo*, sufixo adverbial *-mente*, o sufixo adjetival *-vel* e o correspondente sufixo nominal *-bilidade*.

Este trabalho também resolve também parte do problema de atribuição de categoria gramatical a palavras desconhecidas, isto é, palavras que não se encontram no léxico. O Adivinhador do LexMan foi modificado para tratar especificamente de dois casos: compostos nome-nome e palavras com erros em acentos.

O desempenho das duas soluções foi avaliada e comparada com os resultados originais do LexMan. O novo módulo de geração de palavras sufixadas produziu apenas um incremento de tempo constante nas operações do transdutor, que quase não tem impacto no número de palavras processadas por segundo, e o sistema é agora capaz de identificar mais de 500,000 palavras sufixadas. O Adivinhador modificado obteve uma precisão de 87%.

**Palavras-Chave:** Processamento de Língua Natural; Anotação Morfosintáctica; Sufixação; Transdutores; Análise Morfológica.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Natural Language Processing (NLP) is a sub-field of Artificial Intelligence that aims to enhance and study the interaction and communication between computers and humans. Machine translation, speech recognition and name entity recognition are some of the applications of NLP.[1]

For many NLP tasks, the lexicon is key since many processing steps rely on an accurate part-of-speech (POS) tagging, as well as adequate lexical resources with a high granularity and broad lexical coverage. A particular case in lexical analysis is *derivation*. This is one of devices languages rely on in order to create new lexical items from pre-existing elements, as well as conveying additional meaning by regularly modifying base words. Derivation can be considered a (limited) recursive process as it applies to both non-derived elements (base form) and other words already derived from some base form. This is achieved by adding an affix to a base word. Affixes are bound morphemes that change in a regular way the meaning (eventually, the POS) of the base. While, as previously mentioned, this process is recursive, several complex restrictions apply to this operation, restrictions on when and where an affix can be added. The most common types of affixes are: *prefixes*, added to the beginning of the base; *suffixes*, at the end of the base; *infixes*, inside the base; and *circumfixes*, that is, discontinuous bound morphemes, added simultaneously at the beginning and the end of the base word.

Derivation by suffixation can not only change the meaning of the base word but also its POS. For example, from a direct-transitive apassivable verb *lavar* ('to wash') one can regularly derive adjectives by adding the suffix *-vel*, forming *lavavél* ('washable'). For historical reasons, some forms had their meaning (and syntax) evolve over time so that the current word cannot be regularly derived from its original base form, such as *amável/amar* ('kind'/'to love') even if such base form still exists in the current state of the language. In this work, only productive, that is, regularly derived words are targeted.

The correct identification of derived words is an important feature in the lexical analysis of many NLP systems, as it allows the generation of all possible derived word forms for a given base lexicon. Additionally, derived words need to be associated to their adequate POS, in order to allow for the parsing stage, as well as the lexical (syntatic) information on the base form of the derived word, and so produce the correct syntatic and semantic processing of text.

[1] Adapted from the wikipedia entry for Natural Language Processing: `http://en.wikipedia.org/wiki/Natural_language_processing` [Online; accessed 21-December-2015]

Furthermore, unknown words are abundant in real texts and constitute a relevant issue for NLP systems. These out-of.vocabulary words (OOV) originate from typographical errors, foreign words, or entirely new words. Possessing a string mechanism to try and guess what these unknown words may be not only it enhances the accuracy of POS tagging but it also improves the overall performance of the system, enabling better semantic analysis in later stages of the chain.

This, in broad strokes, is the challenge this work addresses.

## 1.1 Context

The issue of processing out-of-vocabulary, regularly derived, suffixed words in Portuguese is not a new one, and it would be out of the scope of this thesis to produce a comprehensive review of previous work on the topic, hence, just the briefest overview is provided here. A more in-depth review of various systems dealing with suffixation is handled in Chapter 2.

Concerning derivational morphology, in the context of the transfer-based machine-translation (MT) system Eurotra, [2] is probably the first work on the topic. In the early nineties, the construction of large-scale lexical resources required dealing with the phenomena of derivation and several trends can be associated to the development of the following systems: (i) *Palavroso* [18] at INESC-ID Lisboa; (ii) *Palavras* [4]; (iii) *jSpell.pm* [34]; and (iv) *Digrama* [8, 25], which later lead to the Label-Lex-sw lexicon [9].

The availability of user-friendly finite-state tools like *Intex* [31] and later *Unitex* [24] lead to the adaption of existing resources to these platforms [20], including Brazilian Portuguese [22], and the development of several morphological analysis' modules and experimentations in Portuguese derivational morphology [3, 19], a trend that is still pursued today [21] within the NooJ platform [32]. These early works lead to the Morpholimpics joint evaluation contest in 2003 [30], coordinated by Linguateca[2]. More recent work, in different approaches, can also be found in [28] and [33], the former built over the lexicon of [9]. Evaluation on the specific topic of suffixal derivation is scarce, but some data is provided by [3] and [19].

This project aims at processing derived, suffixed words within an existing NLP processing chain, STRING. This system will be briefly described now.

## 1.2 STRING

STRING [14] is a *STatistical and Rule-based Natural lanGuage processing chain* for the Portuguese language. Its architecture can be observed in Figure 1. The first module of the chain is a morphological analyzer (LexMan [35]). This module is responsible for splitting the input text into segments (sentences and tokens) and attributing morphological tags to text segments. A token is the minimal unit of text, which includes not only words but also punctuation, number words, symbols, etc. Words often correspond to lexical units, but some are multiword units, formed by several words such as *lua cheia* ('full moon') or *lua de mel* ('honey moon').

[2] http://www.linguateca.pt/Morfolimpiadas/

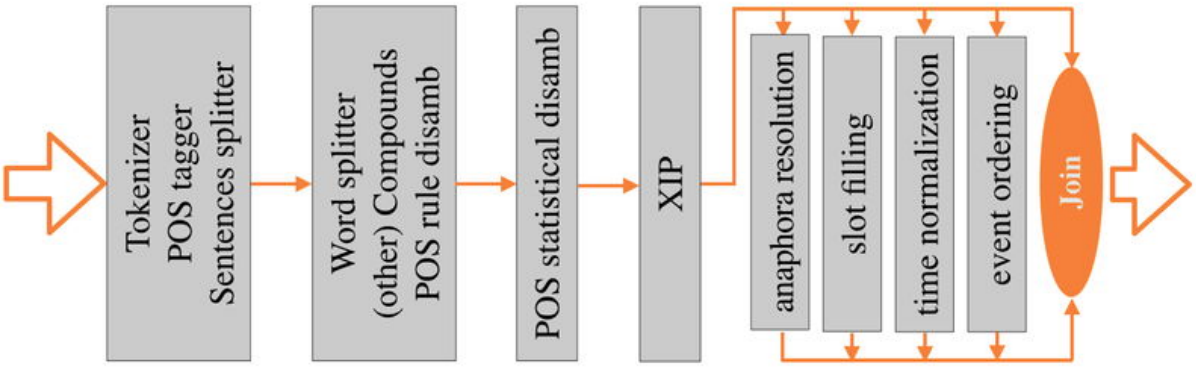


Figure 1.1: STRING's Architecture

It is also LexMan that generates the dictionaries of the system, using a two-level morphology [12] approach, that is, combining a lexicon of lemmas with a set of inflection paradigms that allow the generation, for a given lemma, of all the inflected forms that are associated with it. LexMan performs this task in two separate components, one for verbs and another for all non-verb lemmas, due to technical reasons, more specifically, in order to treat verbs with clitic pronouns.

Words are generated by applying to a base form (stem) the inflection rules described in the inflection paradigms. Hence, to form the plural noun *escadas*, an *-s* is added to the stem of the lemma *escada* (in this case, the stem is identical to the lemma).

Some affixes (currently, only prefixes) are also dealt with using a similar strategy. The current derivation module uses a set of rules, stating which prefixes can be added to different types of base words (i.e. their POS and initial characters) and performs the necessary morphosyntactic adjustments (inserting or removing characters) to adequately generate the derived (prefixed) word.

After morphological analysis, the input passes to a rule-based disambiguator (RuDriCo [7]). This module tries to resolve any conflicting tags produce by the previous module, altering both tags and word segmentation as necessary. For example, resolves contracted words into their component lexical units (e.g. *nas* into *em + as*).

After the previous step, some ambiguity may still exist, derived from RuDriCo not possessing all the necessary rules to fully perform disambiguation. For this task, a statistical ambiguity resolver is used (MARv [26]). This module is based on (second order) hidden Markovian models (HMM) and it uses Viterbi's algorithm [11].

Next, there is XIP [36] (Xerox Incremental Parser), which is the module responsible for constructing minimal syntactic constituents (chunks) and establishing the dependency relations between them. Each chunk has several features which are used to describe its properties. For example, *Pedro* constitutes a proper masculine noun but it also constitutes a named entity (NE) representing a human male individual. All those features are represented by a tag on the chunk. This means that XIP is capable not only of performing syntactic analysis but also of extracting relations between chunks and provide additional meaning to each chunk.

At the end of the processing chain a set of modules can optionally be used which perform: anaphora

resolution [15] (expressions whose meaning and reference depends on other expressions), slot filling [6], time expressions normalization [16] and event ordering [5].

## 1.3 Goal

Currently, LexMan supports the identification of prefixed (and some suffixed) words. This is done by applying derivation rules to base words and then attributing their correct, new morphological category, which can later be used for syntactic analysis.

The purpose of this work is to extend the functionality LexMan so that it is able to automatically identify unknown, but properly derived, suffixed words. For this, a new LexMan submodule will be developed, which, for a given word, identifies and produces all of its suffixed forms (currently only for the most productive suffixes, described ahead, are processed). For example, for the word *escada* the module would produce: *escadinha* and *escadita*. Due to the large number of suffixes in Portuguese this work will only focus on seven of the most productive suffixes: *-inho*, *-ito*; *-íssimo*, *-érrimo*; *-ável*; and *-bilidade*, *-mente*.

This work will also deal with words regularly derived from a compound base, that is, compound words formed by juxtaposition, such as *pés-de-galinha*, 'crow's feet', where some of the elements can undergo regular derivational processes, like diminutive *pezinhos-de-galinha* in as much as ordinary simple words (*pé/pezinho*).

This work will also try to enhance the accuracy of LexMan's Guesser module.

## 1.4 Estimating the relevance of the problem

There are several words for which LexMan cannot deterministically attribute a morphological tag because they are not in the dictionary. For these words, one or several tags are guessed based on the word form of the unknown token. Here we estimate the relevance of this problem.

Using the CETEMPúblico corpus [27] (180 million words of corpus), a list of unknown words was obtained after processing it throuh LexMan. This constitues the list of *guessed words*. This list contains words that were guessed and the number of times they appeared in the corpus: a total of 523,529 words which appeared a 9,981,287 times in the corpus. This list was then filtered using regular expressions. The regular expressions used correspond to the seven productive suffixes, previously mentioned. In Portuguese not all words matching a given ending (*-inho*, for example) are actually suffixed words, e.g. *ninho* ('nest'), is not a suffixed word but a form of a noun. Next is an excerpt of the output of the filtering for the *-ito* suffix.

```
saquito 2
saquitos 2
sardinhita 1
sardinhitas 1
```

| Suffix | Unique Words | Count in the Corpus |
|---|---|---|
| *-ável* | 781 | 4,498 |
| *-bilidade* | 471 | 3,188 |
| *-íssimo* | 657 | 3,235 |
| *-érrimo* | 38 | 72 |
| *-inho* | 5191 | 143,715 |
| *-ito* | 3100 | 48,369 |
| *-mente* | 2048 | 5,729 |

Table 1.1: Number of possible suffixed words in CETEMPúblico's guessed words

| Suffix | Count of Non-Suffixed words in the Corpus | Count of Suffixed words in the Corpus | Percentage of Suffixed words |
|---|---|---|---|
| *-ável* | 1,149 | 991 | 86.0% |
| *-bilidade* | 774 | 730 | 94.3% |
| *-íssimo* | 790 | 775 | 98.1% |
| *-érrimo* | 72 | 59 | 81.9% |
| *-inho* | 990 | 938 | 94.7% |
| *-ito* | 467 | 83 | 17.7% |
| *-mente* | 186 | 40 | 21.5% |

Table 1.2: Percentage of suffixed words in a 100 word sample

```
satidfeitos 1
satifeita 1
satifeitas 1
satifeito 6
```

The output contains (as the original list) the word filtered and the number of times it appeared in the corpus. It is possible to see that from those words only four are actually suffixed words (*saquito* and *saquitos*, from *saco*, 'bag'; *sardinhita* and *sardinhitas*, from *sardinha*, 'sardin'), the others being typographical errors (different forms of the adjective *satisfeito*, 'satisfied').

Table 1 shows the breakdown of the list of guessed words by ending (suffix) and it features three columns: the first has the suffix by which the guessed words were filtered; the second column shows the number of unique words found for that particular word termination; and the third column the number of times these words appeared in the corpus.

Since the amount of filtered words was over 400 words for most suffix terminations, an estimation was made. Out of a 100 word sample, chosen at random, the number of times these words appeared in the corpus was counted and, then, the suffixed words among these 100 were manually identified and the number of times they appeared on the corpus computed. This allows us to estimate the percentage of suffixed words for the whole list of filtered words. The results are shown in Table 2.

Using data from Table 1 and Table 2 (number of times the words appear in the corpus and the percentage of suffixed words relative to the number of words, respectively) it is now possible to estimate how many suffixed words (for the seven productive suffixes this work will focus on) exist in the Corpus. The results are shown in Table 3.

As previously mentioned, the list of Guessed Words for the CETEMPúblico corpus had a total of 523,529 words which appeared a 9,981,287 times in the Corpus. Adding the estimated number of

| Suffix | Count of Non-Suffixed Words in the Corpus | Percentage of Suffixed words | Extrapolated Number of Suffixed Words |
|---|---|---|---|
| *-ável* | 4,498 | 86% | 3,868 |
| *-bilidade* | 3,188 | 94.3% | 2,996 |
| *-íssimo* | 3,235 | 98.1% | 3,170 |
| *-érrimo* | 72 | 81.9% | 59 |
| *-inho* | 143,715 | 94.7% | 136,098 |
| *-ito* | 48,369 | 17.7% | 8,561 |
| *-mente* | 5,729 | 21.5% | 1,231 |

Table 1.3: Extrapolated number of suffixed words in CETEMPúblico guessed words

suffixed words for each of the productive suffixes a total of 155,982 is reached, which represents 1.5% of the Corpus. It is important to note that this percentage might seem low, but that is mostly due to the nature of the words found in this list, it contains not only words that are correctly written but are not in the LexMan's dictionary but also words with typographical errors, the latter being the larger percentage.

In addition to correctly tag more words, the addition of suffixes will also improve syntactic and semantic analysis in later stages of the STRING processing chain. An example of this would be the suffixed word *reizinho* 'little king'. Currently, the word is being guessed by the *Guesser* module in LexMan as a form of the verb *reizinhar*. Since this guessed form for the word was a verb and not a noun, derived from *rei*, not only the POS disambiguation of the remaining words in the sentence may be severely hindered, but also the syntactic analysis of that sentence would be seriously foiled. Furthermore, even if a noun tag was attributed to the unknown form, it would not receive any semantic or syntactic features the parser associates to the base form in the lexicon (i.e. non-guessed words), *rei*. Thus, the word *reizinho* would not be considered to designate a title, given to a human entity. Therefore, all processing steps depending on that information are ignored (e.g. named entity recognition [13] [23] [29]).

## 1.5 Structure of the dissertation

This disseration is organized as follows:

Chapter 2 - describes STRING, the system which the work of this project was developed and other systems with similar purposes;

Chapter 3 - describes the of the new module developed, the Suffixed Word Generator;

Chapter 4 - presents the methodology, evaluation and results regarding the Suffixed Word Generator;

Chapter 5 - describes the modifications made to the Guesser, a submodule of LexMan in order to enhance its guessing capability;

Chapter 6 - defines how the modified Guesser submodule was evaluated and corresponding results;

Chapter 7 - provides conclusion remarks regarding the work developed and how it may be improved still.

# Chapter 2

# State of the Art

In this section, three morphological parsers, their solutions, and how they process and identify affixed words are described, analyzed and compared.

Section 2.1 describes each module of LexMan, STRING's morphological analyzer. Section 2.2 describes Hermit Crab, a morphological parser and generator for generative phonology and morphology. Section 2.3 describes PALMORF, which is the morphological analyzer for PALAVRAS. Section 2.4 describes PALAVROSO a morphological analyzer for the Portuguese language. Finally, section 2.5 provides a comparison between the systems described.

Section 2.6 focuses on spell checking techniques, which may also be a possible solution to identify suffixed words.

## 2.1 LexMan

LexMan (Lexical Morphological analyzer) is one of STRING's modules. It is responsible for the morphological analysis and tokenization. This process consists in splitting the input into tokens (or part-of-speech) and giving them their POS along with a morphological tag. LexMan currently makes use of 12 tag categories and 11 fields, achieving a tagset with high granularity. The process of POS-tagging is split in six modules: word generator, transducer generator, morphological parser, guesser, normalizer and sentence splitter. LexMan's architecture can be seen in figure 2.

### 2.1.1 Text Normalizer

Before any morphological parsing can be done the input text needs to be in a form all other modules in the chain can use. This is called *normalization*. This process is achieved by producing a transducer for the input text and at the same time performing any substitutions necessary.

Normalization is a necessary step for several different reasons. A character in the input text might not correspond to a character of the Portuguese alphabet, thus it must be replaced by a special symbol (*UNKNOWNCHAR*). There are also several ways to represent spaces, tabulations and newlines, but this module replaces all of them into a specific form.

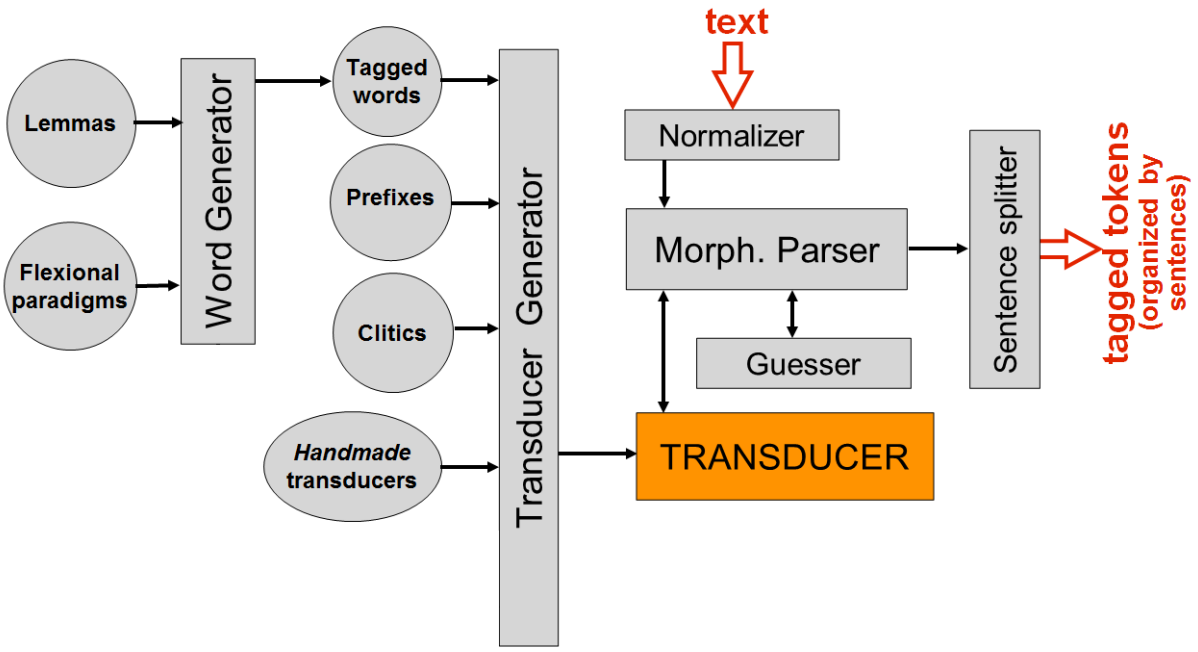


Figure 2.1: LexMan Architecture

It is important in this step that the substitutions be done character by character (as opposed to several characters to one, or one to several). By maintaining the same number of characters as the input text, each character retains its original index in the input text, which is important for some tasks (like adding marks in the original text).

### 2.1.2 Word Generator

Word generation in LexMan is done using a dictionary (or lexicon) of lemmas and a set of inflection paradigms. The file that represents the dictionary of lemmas has the following structure:

```
<lemma> ([(category) (, subcategory)* ]+ (paradigm)+ <stem>)+
```

The lemma represents the base form of the word, typically the form one would look for in a normal dictionary. The lemma is described in the lexicon with its POS (or morphological) category and its subcategories, if any of such apply. Each lemma is also given a set of *stems*, that is, the actual form(s) used as base for the inflection. For each lemma, one or more paradigms are used to generate all of its inflected forms. The information on how the stem is modified to generate these inflected forms is provided in the inflection paradigm. This has the following structure:

```
paradigm-identifier (<word example>)?
    rule-1
    rule-2
    ...
    rule-n
$
```

In turn, each rule shows the following structure:

```
tagset weight <to remove><to add>
```

The paradigm identifier is used in the dictionary file to indicate which paradigm is to be used; an example is usually provided; then follows the tagset change (if any), the weight, and the stem transformation, which charaters are discarded and which are added, respectively, if any. For example, the dictionary entry for the word *escada* would look like this:

```
<escada>      [Nc] FlN11   <escada>
```

As it can be seen, the lemma *escada* is tagged as a common noun (Nc) and its inflected forms are generated from the stem *escada* using the paradigm inflection rule with the identifier *FlN11*. This particular inflection paradigm is the following:

```
FlN11  <insígnia>
    ...sfn.==  0   <><>
    ...pfn.==  0   <><s>
$
```

In this particular example, after the inflection paradigm identifier, an example is provided; the rules indicate that no change is make to the lexical entry POS; the feminine (*f*) singular (*s*) normal degree (*n*) form is produced without neither removing nor adding any character to the stem; while the plural-feminine normal degree (*pfn*) form is obtained by simply adding an *-s* to the end of the stem; the '0' code indicates the stem to be used (by default, there is only one, numbered '0', but there can be more).

The words resulting from applying paradigms to lemmas are then stored in a file, according to their morphological category. The words generated using the lemma and inflection paradigm above would be stored in the file for nouns. The structure of one such file is the following:

```
surface lemma POS-tag weight
```

Where *surface* is the generated word, followed by its lemma, its morphological tags and its weight. During word generation, it is possible that the same word form is generated twice, through different lemma and paradigm combinations. The weight is used to define priorities (discarding the ones with higher weight) in the dictionary because the same *surface* can be generated with different POS.

For *escada*, the entries in the generated nouns file would be the following:

```
escada escada Nc...sfn.== 0
escadas escada Nc...pfn.== 0
```

As verbs need to be complemented with clitic information on generated words the lexicon is first split between verbs and non-verbs. Verbs are then split further into 17 sub-categories, according to which kind of clitics and tenses they can (or not) have. Non-verb words are split into 4 categories: adjectives, adverbs, nouns and other (pronoun, article, adverb, preposition, conjunction, numeral, interjection, punctuation and symbol).

### 2.1.3 Transducer Generator

This module is responsible for constructing a transducer with the words generated by the previous module. For each subcategory of verbs and non-verbs, a transducer is built and, then, using the operations available in FSM[1] and OpenFST [1] libraries, these transducers are composed, to form the final transducer.

The first step in this task is to analyze the generated words and to create the files where all their morphological information is stored, along with the transducers' symbol tables.

The second step is the construction of the transducers themselves. Each generated word is traversed and added to the transducer, so that it links with the initial and final state. Each character of a word is a transition to the next state and the output the character itself. There are also several *epsilon* transitions, which are used to generate additional information regarding the word being matched, including its category, lemma and morphological tagset.

### 2.1.4 Clitic Pronouns

It is during this step that the transducers for clitic pronouns are built, one for each category of clitic. A verb form can have between zero and two clitic pronouns. A clitic initial state may be the verb termination or the termination of another clitic, so that the transducers thus built are slightly different from the ones for verbs and non-verbs [35].

### 2.1.5 Prefixes

LexMan is currently capable of generating prefixed words as well as some suffixed words (manually added to the dictionary). The method is similar to the one used for word generation, that is, by combining lemmas and paradigms. The lemmas' file for prefixes has the following structure:

```
<prefix> [(category)+] paradigm
```

The first column refers to the prefix being defined, the second is a list of categories to which the prefix may be added and, finally, the last column indicates the paradigm to be used. The entry for the *trans-* prefix looks like this:

```
<trans>   [A,N,V,ADV]  Pref33
```

As it can be seen in the example, this prefix can be added to adjectives, nouns, verbs and adjectives. The way this is done is specified by the prefix paradigm with the identifier *Pref33*.

The file for prefix paradigms has the following structure:

```
paradigm  (<example>)?
   rule-1
   rule-2
```

[1] AT&T FSM Tools not available.

| Base Restriction | Generated Prefixes |
|---|---|
| s | tran |
| * | trans-, trans |

Table 2.1: Prefixes Generated from *trans* lemma.

```
   ...
   rule-n
$
```

And the structure for rules is the following:

```
<base-restriction> (cost)? <#car1><add-prefix>Hifen
```

The first column identifies a restriction in the first letter of the word to be prefixed; then comes the cost of applying this rule (if it is not present it defaults to 0), the number of characters to remove from the end of the prefix, the characters to add at the end of the prefix, and the hyphenation rule, that is, the hyphen may be mandatory, nonexistent or optional (*H, S, O* tags respectively).

Still using the prefix *trans-* as an example, the corresponding entry for Pref33 is the following:

```
Pref33  <transiberiano>
<s>          <1><>S
<*>          <0><>O
$
```

In this case, if the word to be prefixed starts with an *s*, the prefix loses its last character (becoming *tran*), gains none and the hyphen is nonexistent. The asterisk in the second line signifies that for all other words the prefix loses no letters at the end, gains none and hyphenation is optional. From the example word we can see that for the word *siberiano* the prefix loses its last letter before being attached, forming *transiberiano*.

The result of this generation process is split according to the category to which the prefixes may be added to. For each category and for each base restriction a file is generated, containing the prefix and its cost. The prefixes generated for the *trans* lemma can be seen in Table 2.1. As noted before, this lemma can be added to nouns, verbs, adjectives and adverbs, so for each of these categories the *tran* prefix will be added to the file of prefixes which base restriction is *s*, and both *trans-* and *trans* will be added to these same files with no base restriction (that is, the word to be prefixed can start with any character, represented by the asterisk).

Following the same strategy that was used for clitics, each of these files is used to build a transducer. The transducers first transition is *epsilon* and the output a special tag, denoting that it is a prefix start, followed by a transition for each of the characters in the prefix itself and another *epsilon* transition to the final state. Figure 2.2 shows the transducer built from some of the generated prefixes that can be added to adjectives with base restriction *r*.

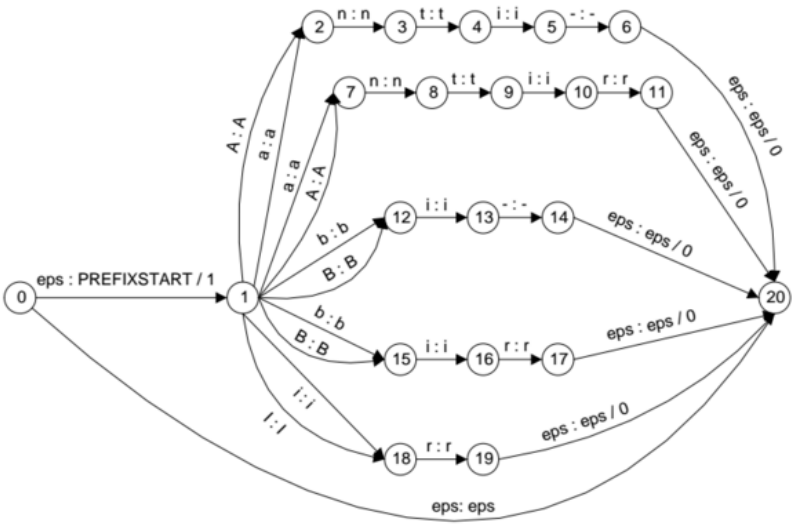


Figure 2.2: Example of transducer built from generated prefix file (from [35]: Fig. 4.27)

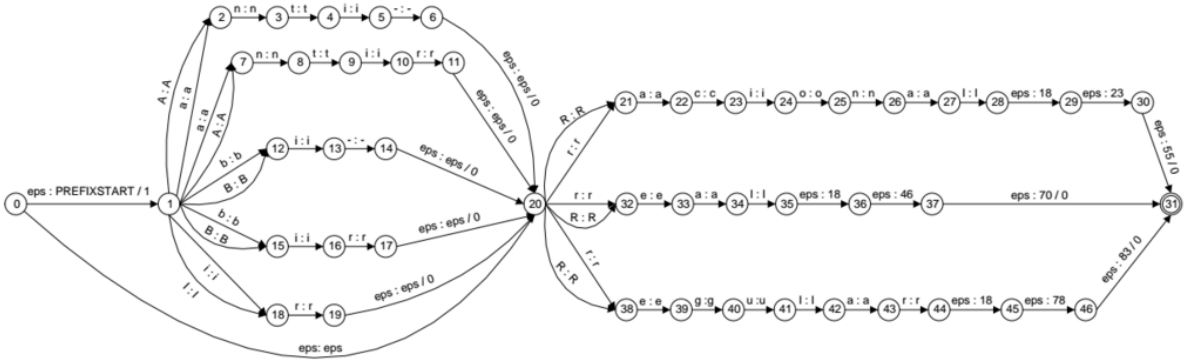


Figure 2.3: Part of a transducer that identifies some prefixed adjectives starting with letter *r* (from [35]: Fig. 4.28)

The prefix transducers are then composed with the transducers of the words that respect the prefix's base restriction, forming the final transducer. Figure 2.3 shows the transducer from Figure 3 composed with three adjectives that satisfy the prefix's base restriction *r*.

### 2.1.6 Morphological Parser

This module is responsible for segmenting the input and attributing morphological tags to each of these segments. The input is a transducer of the input text created by the Normalizer.

This input transducer composed with the final transducer (dictionary) produced by the Transducer Generator.

The result of the composition is then analyzed. If the composition is not empty, then the path with the lowest weight is chosen and the segment is classified with that morphological tag. If the composition is empty, the system verifies if the segment contains only capital letters. If so, then all letters but the first are transformed to lower case and the composition process is repeated. If the composition is not empty, the morphological tag is attributed as before. However, if the composition is empty for a second time,

the system considers that that particular text segment cannot be classified using the current transducer. Then the segment is written to a file to create a log of segments that the transducer cannot classify. Words that cannot be classified are passed to the next module of LexMan, the *Guesser*.

### 2.1.7 Sentence Splitter

After all segments have been given a morphological tag another analysis is performed, with the objective of splitting the text into individual sentences.

The output is split by sentence index and within each sentence by word index. Then the word itself is displayed (between '|' characters). Each word then has the index span it occupies in that given sentence and finally the morphological information, which includes base lemma, POS and morphological tags.

### 2.1.8 Guesser

This module is responsible for trying to guess the morphological tag for words that do not exist in the dictionary and thus could not be tagged using it.

For this, the word termination is analyzed, seeking similarities with known words, and from these results a morphological tag is proposed.

## 2.2 Hermit Crab

Hermit Crab [17] is a morphological parser and generator for generative phonology and morphology. This means that Hermit Crab is both capable of for a given stem, generating all derived words but also finding all possible stems for a given derived word.

Hermit Crab is written in Prolog and Microsoft C and serves mainly as a linguistic tool for language analysis (mostly for English, but it also supports other language-specific rules). The system allows the user to define his/her own phonology and morphology rules or redefine existing ones.

The system is composed by dictionaries, which contain words, stems and roots, along with their morphological and syntactic properties. Hermit Crab then applies morphological rules to the entries in these dictionaries to generate new words.

The system treats affixation as a sequence of steps and not as simple concatenations. The system supports prefixes, suffixes, circumfixes, infixes, among others. A simple suffix rule would be represented as follows:

```
[X1]N -> [1 ut]V
```

This means that this rules takes as input a noun, whose content is represented by X1 and attaches the suffix (*ut*) and the resulting word would be a verb. For the infix *-n-* which attaches after the first consonant plus vowel of a stem, the rule would be as follow:

```
[C1 V2 X3]N -> [1 2 n 3]V
```

where C and V represent consonant and vowel, respectively, and X anything else, in this case the rest of the stem. For languages in which the stem is not parseable in such way (consonants and vowels) this same rule would be applied as a prefix instead. This is possible because Hermit Crab allows the definition of sub-rules for a given morphological rule.

In Hermit Crab each stem has a set of morphosyntactic features, such as tense, person, number, etc. These may be modified by morphological rules. When a morphological rule is applied to a stem, the affix's morphological features override those of the stem, modifying it. Morphological rules in Hermit Crab also support required fields, which means that for a rule to be applied the stem must possess certain morphological features (such as being singular).

## 2.3 PALAVRAS

PALMORF [4] is the module responsible for morphological analysis in PALAVRAS [4], a Portuguese language text analyzer. Grammatical rules are formulated in the Constraint Grammar formalism. The system is composed by several modules and is capable of performing morphological analysis and disambiguation, syntactic and semantic analysis.

PALMORF takes plain text as input and outputs an analyzed file as output where both words and sentences have been split. In this file each word is tagged for its class, inflexion, derivation. Ambiguous words are given multiple tags.

The PALMORF module can be decomposed into two different submodules: the Preprocessor and the Morphological Analyzer. PALMORF allows both analysis of a single word and whole sentences. In the first case, the Preprocessor module is entirely skipped.

The Preprocessor module is responsible for transforming input text into a format the system can analyze. Firstly, it marks with a special sign all the text that does not represent actual words. Secondly, the Preprocessor splits words with line feeds. Words in this context are alphanumeric strings separated by blank spaces, hyphens, non-abbreviation-punctuation, line feeds or tabs. Thirdly, several manually entered polylexical (or multiword) are identified. A polylexical unit is a group of words that would otherwise be treated as individual words (e.g. '*em vez de*'). These polylexical words are treated as ordinary words by the Contraint Grammar rules.

The Morphological Analyzer uses several databases to perform its task: a grammatical lexicon, inflexion endings, suffix and prefix lexicons.

The grammatical lexicon is ordered alphabetically. This allows a word to be searched using a binary technique. To search for a word the lexicon is split in half and decide in which of the halves the word is. It takes only 17 steps to search the whole lexicon but usually only 5 or 6 are needed.

The inflexion terminations are ordered backwards (from the end to the start), the combination rules, base conditions and tagging information attached in successive fields. Look-ups are thus done backwards, starting at the end of the word, where each field controls the next. For example, in *comeis*, *-s* is looked up first and then *-is* (in a list also containing *-as*, *-es*, *-os*, etc.) and finally *-eis*. The entry for *-eis* then provides the tagging information for this inflexion.

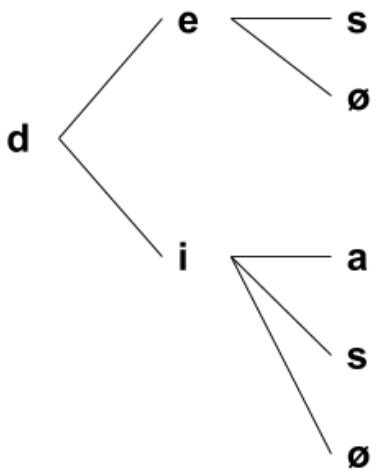


Figure 2.4: Prefix Pointer Tree Searching

| Suffix Form | Base Condition | Word Class Condition | Combination Rules | Output (derivation class and semantics) | Suffix Word Class | Suffix Combination Rules |
|---|---|---|---|---|---|---|
| *-ista* | V | asnb | | DERS -ista | smf | |
| *-ico* | VTP | sn | | DERS -ico [ATTR] | adj | |
| *-ótico* | ose | sn | | DERS -ico [ATTR] | adj | R |

Table 2.2: Example of PALAVRAS Suffix Lexicon

The prefix and suffix lexicons are stored as alphabetical pointer trees, with the suffixes reversed. Each pointer represents a character of the prefix or suffix and offers the path (a pointer) to the next character, the last pointer giving access to the combination rules, base condition and tagging information of the chosen affix. Figure 2.4 shows part of a pointer tree for prefixes starting with the letter *d*.

The lexicon for suffixes is a table featuring 7 columns and where each line represents a suffix. Table 2.2 shows part of such a lexicon.

The first column represents the suffix itself and is what the suffix module cuts off the word it receives as input.

Base conditions can either be a string or contain orthographic combination information. In case of a string, this is added to the word before it is searched in a dictionary. In case of combination information, as the *V* code in the first lone of the table, it means that the suffix connects with a root word by replacing its last vowel, thus, several vowels are tried when searching the dictionary for a root word. Word class condition is a list of morphological categories of root words to which the suffix may apply. For example, in the table, we see that *-ico* can be attached to nouns(*s*) and names(*n*) (i.e. proper nouns).

Combination rules represent prefix restrictions, that is, for a suffix to apply one of the given prefixes must be also present in the word.

Output is the string produced by the analysis. It contains the derivative stem (code *DERS*), the

suffixes and may be followed by some semantic markers.

Suffix word class is the suffix's own morphological category. This category is transferred to the root word. This is necessary for words with multiple suffixes, the outermost suffixes determines the final word class.

The last column, Suffix Combination Rules also restricts when the suffix may be attached, that is, for a suffix to apply another suffix must also be present in the word or the word may possess no other suffix (*R* code in the table, which means the suffix connects only to root words).

## 2.4 PALAVROSO

PALAVROSO [18] is a morphological analyzer for the Portuguese language. This system makes use of both morphological rules and dictionaries to perform its task, however, these two are independent and thus the system can still provide a result without one or the other.

When a word is submitted to be processed, the system applies several rules which produce hypotheses. These hypotheses are then verified in the dictionary. The system can work on several modes providing more or less information, outputting all hypotheses or only those confirmed by the dictionary. Morphological rules for this system have been divided into four categories: closed classes, adverbs, names/adjectives, and verbs. Likewise the entries in the dictionary have been split into the same categories.

Like many other language processing systems, PALAVROSO has a preprocessing step in which it standardizes the input text in a way it can be analyzed. The system also contains a post-processing step, in which the user can specify the amount of information outputted by PALAVROSO, or even specify a whole new tagset to be used.

At the core of the system there are six modules, each responsible for processing a specific kind of words: closed class words, names and adjectives, verbs, adverbs, verbs with clitics and compound words. A word is analyzed by each module (if applicable) in the previously mentioned order, and at the end the result of all modules is organized and outputted.

Each module contains a set of rules, specific to the classes it processes. The general idea of the process is to apply rules to the inputted word until a root word (one that can be in the dictionary) is found. For example, for the word *utilíssimas*, the word would go through three steps: number, gender and finally comparison.

Number processing is based on a table that contains a column for the plural termination for words, a column for their singular form termination and a column that classifies the input type. A few examples of entries can be seen in Table 2.3.

Using the table, the word *utilíssimas* would fall under the first line and thus producing the word *utilíssima*. As the last line of the table indicates, it is also possible to process words such as *lápis* which has the same form for single and plural, thus their number being Invariant (I in the table).

Likewise, gender processing is also based on a table but with two columns: the first for masculine termination, the second for feminine termination (which may be the same for some words). Table 2.4

| Plural | Singular | Number |
|---|---|---|
| as | a | P |
| os | o | P |
| teis | til | P |
| is | is | I |

Table 2.3: Example of number processing in PALAVROSO

| Masculine | Feminine |
|---|---|
| or | ora |
| o | a |
| . | ã |
| eiro | . |
| ista | ista |

Table 2.4: Example of gender processing in PALAVROSO

shows an example of such table.

Entries with a dot serve only to identify terminations which are characteristic to one of the genders. For example *ã* is the characteristic of feminine words (‘irmã’, ‘capitã’). Following the rules on the table we can say that the word *utilíssima* is the feminine form of the word *utilíssimo*.

Finally, degree comparison processing is also done using rules, which are expressed in a table. The table contains the following columns: termination inflexion, word termination, comparison, check for diacritic and punctuate, Table 2.5.

As it can be seen in this table, if a word ends with *-inha* then it is the diminutive of a word that is formed by replacing *-inha* with *a*. The *Check Diacritic* flag indicates that the suffix cannot be applied to words with diacritics (as *rápido*). The last column indicates that if the modification of the word does not form a word existing in the dictionary then diacritics should be tried. For example, the word *monogaminho* is the diminutive of *monógamo* but the application of the rule would form *monogamo*. Several graphical punctuation are tried, until the word is found in the dictionary.

While PALAVROSO does not explicitly identify suffixed words, it identifies degree comparison, which are formed through suffix based rules.

## 2.5 Comparison

In this section a comparison is made between the systems previously described. For this comparison we will focus on the following aspects: the language processed by the system, programming language(s) it was written in and whether it supports suffixes or not. The results can be seen in Table 2.6.

| Inflexion Termination | Word Termination | Comparison | Check for Diacritics | Try Diacritics |
|---|---|---|---|---|
| inha | a | Diminutive | 1 | 1 |
| ona | a | Augmentative | 0 | 1 |

Table 2.5: Example of degree comparison processing in PALAVROSO

| System | Main Processed Language | Programming Language(s) | Suffixes |
|---|---|---|---|
| LexMan [35] | Portuguese | Java, C++, [2] | No |
| Hermit Crab [17] | English | Prolog, Microsoft C | Yes |
| PALAVRAS [4] | Portuguese | C | Yes |
| PALAVROSO [18] | Portuguese | C | Yes |

Table 2.6: Comparison between the described systems.

# Chapter 3

# Suffixed Word Generation

This chapter presents the solution developed to identify suffixed words within the LexMan module of STRING. The chapter starts with a description of the new module. Then, it focuses on the description of the data necessary for the task, on how the new module integrates with LexMan's other modules and on how the special cases of compound words and words with diacritics are handled.

## 3.1 Architecture

Figure 3.1 shows the new architecture of LexMan. The new module, the Suffixed Word Generator, generates suffixed words using the same two-level morphology approach as the Word Generator. For suffixes, this is done automatically through an algorithm. The module takes as input all the words generated by the Word Generator and several files describing the suffixation paradigms. The process consists of the steps presented below.

#### Simple Words and Compound Words

The Word Generator outputs both simple and compound words. For simple words, zero or more characters are removed from the end of the word and the suffix is added. For compound words, the suffixation process is not always allowed, and it depends on the type of compound. The suffixation process applies only to an element of the compound, either the first or the last, depending on which kind of compound is being suffixed. For example, in *chapéu-de-chuva* ('umbrella'), only the first element *chapéu* is suffixed, forming *chapeuzinho-de-chuva*. However, for other types of compounds such might not be the case such as in *pequeno-almoço* ('breakfast'), it is the second element that is suffixed, forming *pequeno-almocinho*. More information regarding the suffixation of compound words will be detailed further below.

#### Parsing

Figure 3.2 describes the internal structure of the Suffixed Word Generator module.

The files containing both simple and compound words, are processed and parsed by the Suffixed Words Generator module. Each line of these files corresponds to a word to be suffixed. The files are

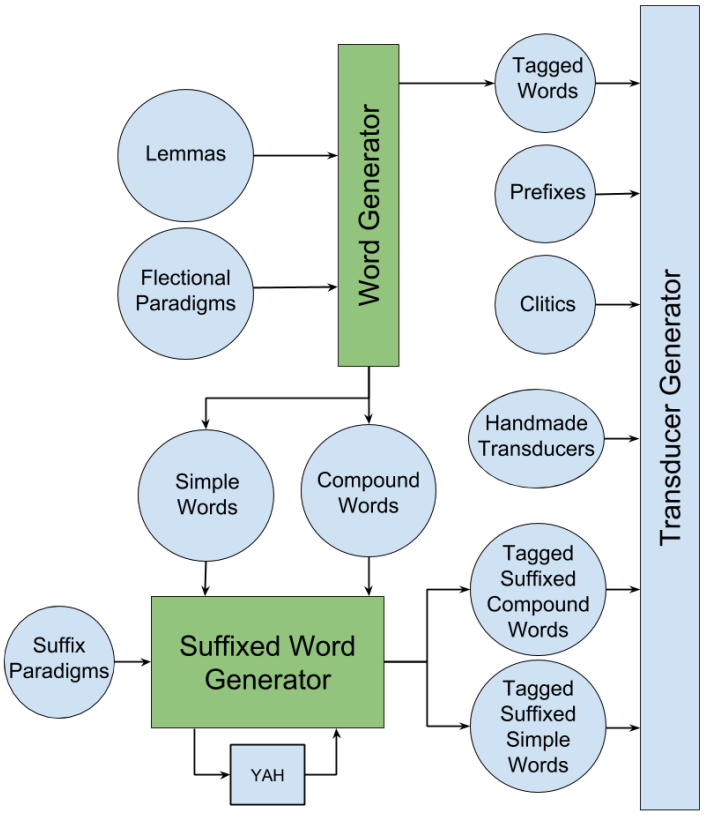


Figure 3.1: Internal Architecture of the new module

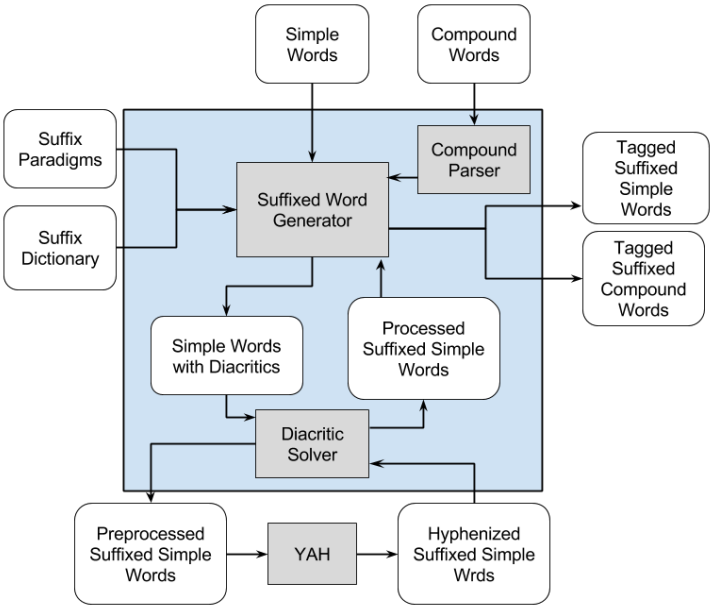


Figure 3.2: New Architecture of LexMan

then given as input to the rest of the module. The syntax of the input file is different for simple and compound words.

```
(word lemma tagset weight)*
```

For simple words, each line contains the word to be processed, its lemma, its part-of-speech (pos) and its weight. A concrete example can be seen below.

```
fuselagem       fuselagem       Nc...sfn.==     0
fuselagens      fuselagem       Nc...pfn.==     0
fusionista      fusionista      Nc...s=n.==     0
fusionistas     fusionista      Nc...p=n.==     0
```

**Example 1.** Simple Words File

For compound words, each line contains the word to be processed, its lemma, the type of the compound, the word of the compound that is suffixed, the part-of-speech of the compound word, and the weight.

```
(compound-word       lemma   compound-type   word-to-suffix  pos     weight)*
```

An example can be seen below:

```
filho único     filho único     Comp1   filho   Nc...smn.==     0
filha única     filho único     Comp1   filha   Nc...sfn.==     0
filhos únicos   filho único     Comp1   filhos  Nc...pmn.==     0
filhas únicas   filho único     Comp1   filhas  Nc...pfn.==     0
```

**Example 2.** Compound Words File

#### Paradigm Loading

This module first loads the Suffix Dictionary file, which associates each suffix to the morphological classes it applies to, and the files containing the suffix paradigms. It then loads the files containing the suffix paradigms specified by the Suffix Dictionary file. The suffix dictionary file specifies the suffixes that should be considered. Its syntax is presented as follows:

```
<suffix1>       [(category-restriction)*]       file1
...
<suffixN>       [(category-restriction)*]       fileN
```

Each line refers to a suffix to be treated by the module. The first column indicates which suffix the line refers to; the second column states the grammatical categories to which the suffix may be applied to; and, finally, the paradigm file containing the rules to generate the suffixed words. Example 3 shows an example of one such file.

```
<inho>          [A,Nc]  SUFinho.pdg
<ito>           [A,Nc]  SUFito.pdg
<íssimo>        [A]     SUFissimo.pdg
<érrimo>        [A]     SUFerrimo.pdg
<mente>         [A]     SUFmente.pdg
```

**Example 3.** Suffix Dictionary Syntax

The suffix paradigm file specifies for a given suffix how the suffixed words are formed. The file has two main zones: *exceptions* and *suffixation rules*. The first zone states three exception types:

```
file-exception:
[(file)*]

lemma-ending-exceptions:
[(-ending)*]

lemma-exceptions:
[(lemma)*]
```

It is necessary to deal with exceptions because the grammatical category alone, specified in the suffix dictionary file, is not the only restricting factor when it comes to word suffixation and other types of exceptions are considered for a given suffix: (i) *file* exceptions, for all stems coming from a given lexicon file[1], the entire file is ignored. As Example 4 shows certain word types are not suffixed, such as *gentilics*, or foreign words, which do not follow the same suffixation rules; (ii) *lemma ending* exceptions, for all stems with a given termination (e.g. *-âneo* termination for the suffix *-inho*) are ignored because they never have that given suffix; and (iii) and *lemma* exceptions, for specified lemma to be excluded from the suffixation process. A concrete example can be seen in Example 4.

```
file-exceptions:
[gentilicos.dic, foreign.dic]

lemma-ending-exceptions:
[-âneo, -cromo]

lemma-exceptions:
[alto, animal, ano]
```

**Example 4.** Suffix Paradigm Exceptions

[1] In LexMan, the lexicon is distributed by several files, some of them corresponding to particular subsets (e.g. *colour* nouns and adjectives, nouns designating *animals* or *plants*; *gentilic* adjectives and nouns, nouns designating natural languages (e.g. Latim 'Latin'); *foreign* words, etc...).

The second part of the paradigm file contains the *suffixation rules*, whose syntax is the following:

```
tag-restriction (<remove><add> weight new-tag [(exceptions)*])+
```

To process a word, its morphosyntactic category (or PoS) is first matched against all the class restrictions of all the suffixes. For example, the word 'gato' (*cat*) being a common noun will match with *-inho* and *-ito* restrictions (adjectives and nouns) but not *-íssimo* which only accepts adjectives. The suffixation process then begins for each suffix with which the word was successfully matched.

The first column restricts which values the morphological tag of the inputted word may have. For example, certain rules may only apply to a specific combination of gender and number. In this tag restriction, the '*' character represents any value in the inputted word tag. The second column represents which characters are removed from the end of the inputted word and which characters are added to form the suffixed word. The third column specifies the weight for each particular rule. The fourth column specifies the tag of the suffixed word produced by that rule. In this new tag, the '*' character represents the value in the original tag of the inputted word. Finally, the fifth column is a list of words to which that specific rule does not apply. The following example shows two such rules, the first generating only a suffixed word and the second generating two suffixed words:

```
**...smn.** <io><iozinho>     0 **...**d.x* []
**...smn.** <ico><icozinho>   0 **...**d.x* []
            <ico><iquinho>    0 **...**d.x* []
```

**Example 5.** Suffixation Rules

The first rule adds the suffix *-inho* to words whose PoS is defined in the Suffix Dictionary without changing them (first '**'), with the morphologic feature-values corresponding to the singular-masculine-normal degree (`smn`); it matches the *-io* ending (e.g. *tio* 'uncle') adds the suffix along with a linking consonant *-z-*, and changes the morphological values from normal to diminutive (`d`), while indicating this word is derived by suffixation (`x`). The second rule is similar, but it applies to *-ico* ending words; the suffixation can be achieved, either by using the linking consonant *-z-*, or by directly attaching the suffix to the root, along with the orthographic-morphotactic change of <c> into <qu> (e.g. *rico* 'rich'), thus forming both *riquinho* and *ricozinho*.

During the loading phase the integrity of the file is also verified to ensure that not only it contains both the exception and rule zone but also that both have a correct syntax. The paradigms rules are checked to ensure they have the correct number of parameters and that their type is correct. Rules with duplicate restrictions (in both PoS and letter removal) are also verified. If any of these cases if found then an error message is produced and the suffixation process halted.

#### Suffixation Process

To process a word, its morphological class is first matched against all the class restrictions of all the suffixes (as specified in the Suffix Dictionary). The suffixation process then begins for each suffix with which the word was successfully matched.

The first step is *exception processing*. The inputted word is matched for all the exception types described above. If a word matches any of these exceptions that suffix is not applied. The second step is the *rule matching*. Both the word's morphological tag and termination are matched against the rules restrictions. If they both match, then the specified characters are removed from the end of the word and the specified characters (corresponding to the suffix) are then added to the end. The new (suffixed) word is then given a new morphological tag and weight.

It is possible that several rules in the Suffix Paradigm file may apply. In this case, only the most specific is used. A rule is considered more specific than another one if it removes more characters from the end of the base word. Two rules can never have the same specificity, such case is verified in the paradigm loading phase and the process halted if such occurs.

#### Diacritic Removal

As explained in the previous section, the suffixation rules in the paradigm files produce suffixed words by matching the inputted word with a series of restrictions and then removing certain characters from the end of the words and adding others to form the suffixed form. Words that contain diacritical marks (before the suffixation process) are flagged for diacritical removal.

In Portuguese, it is possible that after the suffixation process the derived words may gain one or more additional syllables. For example, the suffixed word *rapidinho* is formed from the base word *rápido*. The suffixed word must then lose the graphical diacritic of the base form. This happens because in Portuguese the location of the stressed syllable (among other factors) determines whether a word has a diacritic or not. Words stressed in the third-from-last syllable are always marked with acute accent (and more rarely with circumflex accent ).

Additionally, in Portuguese, words derived through suffixation only gain graphical diacritics if the diacritic is already part of the suffix (for example, *lento* and *lentíssimo*), and thus it is automatically added through the rules described above.

To determine whether a diacritic should be removed or not after the suffixation process, words with diacritics are processed by an hyphenator (YAH) [10].

This module takes a word and hyphenizes it by syllables (split by the '=' character). Words with diacritics in any but the last three syllables have such diacritics removed [2]. For the suffixed word *rápidinho*, the word generated by the suffix paradigm, YAH would output the following hyphenization:

```
rá=pi=din=ho
```

The word would thus lose its diacritic to form the correct form of the suffixed word, *rapidinho*.

To deal with these issues, all suffixed words generated that contain diacritics are outputted to two different files, one containing the generated, unmodified words and the other containing the generated word but preprocessed for the hyphenization process.

Finally, words in LexMan may contain regular expressions in them, to express an optional character (e.g. *ac*`?`*to* 'act') or a choice between several characters in a given position (e.g. *o*`[iu]`*ro* 'gold'). YAH,

[2] There are some exceptions to this rule, envolving other diacritics than accute accents, e.g. *órgão*/*órgãozinho*

| Compound Type | Formation | Suffixation | Example |
|---|---|---|---|
| Comp1 | Noun-Adjective | 1st Element | *batatinha doce* |
| Comp2 | Noun-de-Noun | 1st Element | *luazinha de mel* |
| Comp3 | Noun-Noun | 1st Element | *peixinho-lua* |
| Comp4 | ?-Noun | 2nd Element | *mini-mercadozinho* |
| Comp5 | ?-Adjective | 2nd Element | *mal-humoradinha* |
| Comp6 | Verb-Noun | None | *guarda-loiça* |
| Comp7 | Adjective-Noun | 2nd Element | *pequeno-almocinho* |
| Comp8 | Adjective-Adjective | None | *lateral-esquerdo* |
| Comp9 | Noun-Noun | 1st Element | *decretinhos-lei* |

Table 3.1: Compound Types

however, does not allow the use of regular expressions and this words must first be preprocessed, having their regular expressions removed. After YAH processes the word, regular expressions are reintroduced and the diacritic marks removed as necessary. This is done by keeping the original words stored and then traversing them character by character. When a regular expression is found on the original word then that same regular expression is introduced in the new, processed word.

The YAH module then processes the words and outputs the result to another file. This file is then parsed by the Suffixed Words Generator. It parses the hyphenized words and determines whether the diacritic is to be removed or not according to the rules described previously. The words then have their regular expressions reintroduced in the word, by reading the file containing the unmodified words. If a regular expression happens to represent a choice between a form with diacritic and one without (e.g. [ée]) and the word had its diacritic removed then the regular expression is removed altogether, and only the form without diacritic remains.

Each word processed in this manner is stored so that for a given original word with diacritics the result (with or without diacritics) can be later accessed.

#### Compound Words

Compound words are processed last. They use the same Suffix Paradigm files described above. However, in this case, it is not the whole word that will be suffixed, but just one of its elements. For example, in the compound word *chapéu-de-chuva* ('umbrella'), only the first noun *chapéu* ('hat') is suffixed, forming *chapeuzinho-de-chuva*. The element receiving the suffix depends on the type of compound. Table 3.1 shows the different types of compounds used in LexMan, their internal structure, the suffixation rule and an example.

For each compound word, the element that can be suffixed goes through the suffixation process, being matched with restrictions and exceptions, exactly as if it were a regular simple word. All of its derived forms are generated and, if they happen to have diacritics, the corresponding diacritic treatment is checked directly. This is possible because all diacritic modifications are stored (both the original and the processed word), thus allowing for a direct search for the corresponding processed version of a word containing a diacritic. The generated suffixed word is then attached to the rest of the compound (the other elements), forming a suffixed compound word. For example, for the compound word *peixe-lua* (*moonfish*), the first element *peixe* is the suffixed element. The suffixed forms would be gener-

ated *peixinho*, *peixito* and then attached to the second element, forming the suffixed compound words *peixinho-lua* and *peixito-lua*.

#### Duplicates

Both suffixed simple and compound words are then merged into a single file. This file is checked for duplicates, which are removed. Each word is then merged with the other, non-suffixed words, generated by the Word Generator, according to their grammatical category. During this phase a special module was developed to check for duplicate forms.

As previously mentioned, in the original architecture of LexMan some suffix paradigms were manually associated with some lemmas and thus some suffixed words were generated by the Word Generator module. To check for duplicates, a special module had to be created. This was necessary because, in this case, detecting duplicates was not a trivial task.

Certain words generated by the Word Generator did not contain regular expressions while the ones generated by the Suffixed Words Generator did. An example of such would be *fraccionável* ('fractionable'), which can also be written as *fracionável*. The word was already being generated but did not contain the regular expression for the optional letter *c*, which the Suffixed Word Generator included (to form 'frac?cionável'). For obvious reasons, simple string matching techniques do not find both words to be the same. Thus, the new module, to decided whether a word is duplicated or not, first expands regular expressions into its forms. If any of the forms is already generated, then it is considered a duplicate and the word is not added.

#### Transducer Generation

Finally, the files containing all the generated words (now both suffixed and non-suffixed words) are then used to generate part of the final transducer, used to analyze inputted text.

## 3.2 Modifications to the Word Generator

Several changes have been made to the Word Generator module in order to improve its performance. In addition to the files previously generated, it now outputs an additional file containing only compound words. This file contains not only the compound word, its lemma, tagset and weight, but also the word of the compound that can be suffixed. This was necessary for the suffixation process as compound words required both additional information and separate file as compound words are not processed at the same time as simple words.

Some specific types of compounds were found to generate erroneous words. types 1 and 7 are formed by combining several simple words. Comp7 is composed by an adjective and a noun, in this order (e.g. *alta fidelidade*) while compound type Comp1 involves a noun and an adjective (e.g. *batata doce*). The compound is formed by combining the several forms of both adjective and noun, that is, the masculine singular, feminine singular, masculine plural and feminine plural forms of each adjective

and noun. Compounds 1 and 7 were producing incorrect words such as *mau pagadora* and *rico-mulher* where there is no agreement between the adjective and the noun. When generating the compound forms, these two compound types must guarantee that each element of the compound agree in gender with the other. That is, that if an element is feminine, the other either is feminine or gender-unmarked. The two previous examples violate this rule because one element is of masculine gender and the other feminine.

Compound 1, 7 and 8 were also violating another rule. When attributing a morphological tag, only a word of the compound matters, which is the element that specifies the gender and number of the whole word. For compounds 1, 7 and 8 it is the second element that specifies such values. However, words like *accionista maioritário* and *grandes superfícies* were having their morphological tags attributed from the first element ('accionista' and 'grandes' which are both non marked for gender) when it should be the second ('maioritário' and 'superfícies).

The Word Generator was violating the two rules previously described and thus wrongly generating words. Both cases were thus corrected, the first by verifying whether the gender of both elements match, and the second by attributing the morphological tag from the correct element of the compound.

# Chapter 4

# Evaluation

A new module has been added to the LexMan's architecture, allowing for the generation and identification of some types of suffixed words. This chapter describes the impact of this new module in LexMan's performance, both in time and memory usage and in accuracy.

## 4.1 Transducer Generation

Table 4.1 measures the impact of the changes introduced in LexMan on the size of the transducers built before and after the changes to LexMan.

| | **Before** | **After** | **Difference** |
|---|---|---|---|
| Transducer Generation (s) | 635 | 639 | 0.6% |
| Size (MB) | 214.9 | 294.0 | 36.8% |
| Number of States | 6,432,441 | 8,944,098 | 39.0% |
| Number of Arcs | 8,605,947 | 11,666,048 | 35.5% |

Table 4.1: Generation time, size of the transducer and number of states

As can be seen by comparing results in the table, the generation time increased by an average of 4 seconds. The transducer size increased 80 megabytes (36%). Likewise, the number of states and arcs of the transducer also increased 39% and 35.5%, respectively.

Therefore, the increase on the size of the transducer had only a minimal impact in the time required by the system to generate it.

## 4.2 Testing Corpus and Evaluation Scenarios

This section describes in more detail the data used for the evaluation and the results of the evaluation of each environment for time and memory.

The corpus used for the evaluation is part of the CETEMPúblico corpus [27]. Seven files were use for the testing, ranging from 1 sentence up to 10,000 sentences. These files were used as input for the STRING chain, its output stored so that they can be verified, in order to determine whether the difference

| File Name | Number of Sentences | Number of Words | Size (KB) |
|---|---|---|---|
| Parte08-1.txt | 1 | 32 | 0.21 |
| Parte08-10.txt | 10 | 380 | 2.22 |
| Parte08-100.txt | 100 | 2388 | 14.61 |
| Parte08-500.txt | 500 | 11.189 | 69.51 |
| Parte08-1000.txt | 1000 | 22.403 | 140.17 |
| Parte08-5000.txt | 5000 | 109.902 | 686.74 |
| Parte08-10000.txt | 10000 | 219.530 | 1368.86 |

Table 4.2: Characteristics of the files used in the performance evaluation

| | Before Suffixes | Before Suffixes | After Suffixes | After Suffixes | |
|---|---|---|---|---|---|
| File Name | Time (s) | Time (ms/w) | Time (s) | Time (ms/w) | Time Difference |
| Parte08-1.txt | 4.66 | 1.45 | 6.37 | 1.9 | 36.6% |
| Parte08-10.txt | 4.83 | 0.17 | 6.52 | 0.17 | 34.9% |
| Parte08-100.txt | 5.81 | 0.02 | 7.58 | 0.03 | 30.4% |
| Parte08-500.txt | 10.6 | 0.009 | 12.25 | 0.01 | 15.5% |
| Parte08-1000.txt | 16.68 | 0.007 | 18.30 | 0.008 | 9.6% |
| Parte08-5000.txt | 69.29 | 0.006 | 72.18 | 0.006 | 4.1% |
| Parte08-10000.txt | 135.7 | 0.006 | 139.98 | 0.006 | 3.1% |

Table 4.3: Evaluation results for both scenarios

would be due to an error or an improvement of the new module and their characteristics can be seen in table 4.2. For this evaluation two scenarios were set:

**Before Suffixes** - this scenario constitutes the original LexMan prior to the alterations introduced by the new module developed here.

**After Suffixes** - this scenario includes all the alterations made to LexMan, including the new module developed, the Suffix Words Generator.

## 4.3 Time Evaluation

Using the seven evaluation files described previously the time necessary to process each of them was tested.The number of words is used to measure the amount of time used to process each word, in milliseconds per word (ms/w). Each file was processed 4 times, an average of all run times was then calculated for each file. Table 4.3 shows the average time (4 runs) necessary to process each file.

Analysis of the results shown on table allows one to conclude that the time is very weakly related with input size.

To further study the impact of additional words in the processing time, we took the new LexMan lexicon of inflected words (with the new suffixed forms) and progressively added increments of 100,000 new suffixed artificial words (for example, *rapidixpto*). These artificial words were mostly adjectives with a dummy suffixed at a time. The new, artificially enlarged, lexicons were added to the system and the same files described in Table 4.3 (but only those with 1 sentence and up to 1000 sentences) were processed anew. Results are shown in Figure 4.1.

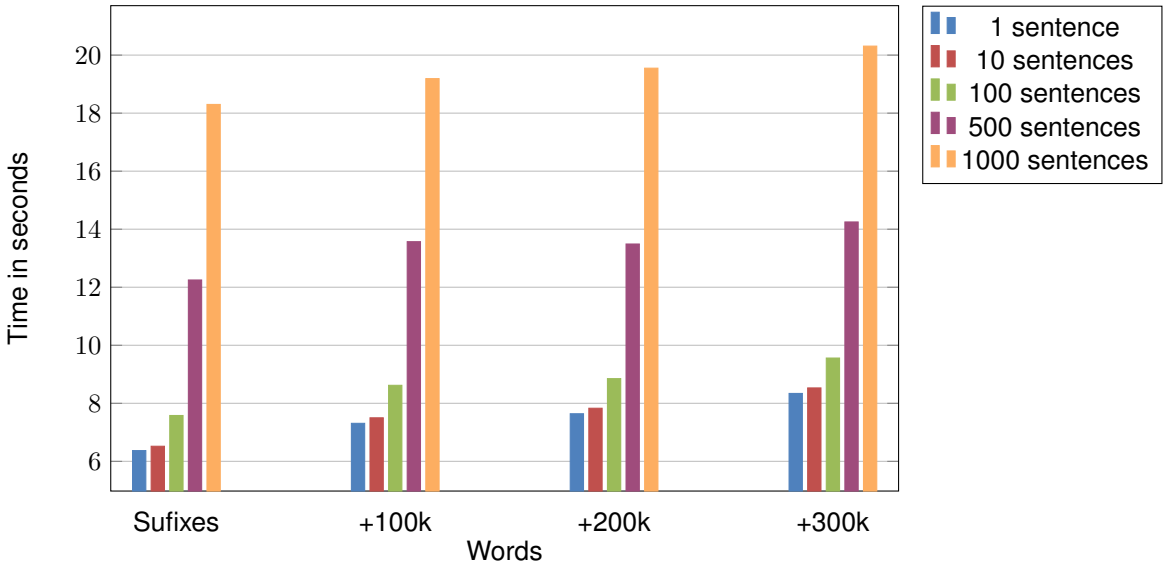


Figure 4.1: Evaluation results for 100k increments of words

As it is possible to see in the graph, the impact of each increment of 100,000 words on the time required for processing them is constant, seeming not to depend on the size of the input. This confirms the results of Table 4.3.

## 4.4 Memory

This sections compares the memory used by the system before and after the changes. This comparison is made using the two scenarios described in the previous section: before and after the introduction of the suffix processing module. The unit of the table is megabyte (MB) and the results are shown in table 4.4, using megabytes (MB) as the measuring unit.

| File Name | Memory Usage before Suffixes (MB) | Memory Usage after Suffixes (MB) |
|---|---|---|
| Parte08-1.txt | 550.3 | 267.5 |
| Parte08-10.txt | 592.2 | 269.5 |
| Parte08-100.txt | 635.7 | 1005.1 |
| Parte08-500.txt | 968.4 | 1221.7 |
| Parte08-1000.txt | 1239.3 | 1499.4 |
| Parte08-5000.txt | 3950.1 | 4207.6 |
| Parte08-10000.txt | 6497.1 | 7026.7 |

Table 4.4: Memory Usage before and after Suffix changes

The values in table 4.4 were registered during the composition operation, that is, when the transducer for the input corpus and the transducer of the tokenizer are being composed. The Table shows an increase in the memory usage as the size of the input grows for both scenarios. It also shows that for most cases the *After Suffixes* scenario requires more memory than *Before Suffixes* scenario. These results stem from the fact that with the introduced changes the transducer has grown, and thus, besides having to load more information into the system, the composition also results in more data.

## 4.5 Accuracy

To measure the accuracy of the new module an excerpt of CETEMPúblico was used. This excerpt contained 414,132 sentences and was split into 208 files, each containing 2,000 sentences. The result of processing the 208 files was first filtered, removing all words not classified as suffixes, then duplicate words were also removed. The remaining words were then classified as *Correct* or *Incorrect* according to the following criteria: if the Suffixed Word Generator generates a form that is i) of the same category and ii) has the same form as another already in the lexicon, then the generated form is wrong and, thus, is to be categorized as *Incorrect*; iii) if the generated form does not exist in the lexicon; or if the generated form exists but belongs to another grammatical category; or if it exists in the lexicon and in the same category but the other form is a Proper Noun, then it is to be considered *Correct*.

As such, if the suffixed form *mantinha* was tagged as both a suffixed common noun (from the noun 'manta') and a verb (from the verb 'manter') then the suffixed form would be considered *Correct* as both form belongs to different grammatical categories. However, if the suffixed form *galinha* was tagged as both suffixed common noun (from the noun 'galo') and a non-suffixed common noun *galinha*, then the suffixed form would be considered *Incorrect*.

The output produced a total of 917 unique words tagged as suffixed. After manual evaluation there were 58 words considered as *Incorrect*, thus achieving 93% of accuracy.

These errors stemmed from several sources. Some were problems in the suffix paradigms, where rules were not as granular as they needed to be. For example, *largíssima* being generated from base adjectival forms ending in *-go* instead of *larguíssima*. Another error was detected in the formation of adverbs and adjectives. When a derived word originates a word in another category it should have as its lemma the derived form. For example, when deriving *aprovadoramente* from *aprovador* the lemma should also be *aprovadoramente*, however the lemma attributed to these cases was the original word, constituting an error. The same error was found in the generation of adjectives from verbs. For example, *inutilizável* from the verb *inutilizar* should have as its lemma *inutilizável*.

These errors were then corrected and subsequent processing of the same input produced no errors.

## 4.6 New Words

In total, the new module introduces 529,232 new (suffixed) words to the chain. Of these new words, 246,316 are adjectives, 282,916 are nouns; 209,824 are compound words and 319,408 are simple words.

An important thing to note is that not all of the introduced suffixed words are likely to ever appear in a text. A side effect of automatic generation versus manual generation is that there is few to no control over the output. Thus, there are many generated words that while they exist and are valid suffixed words in Portuguese, they aren't commonly used, if ever.

# Chapter 5

# Guesser Improvements

The Guesser is a submodule of the LexMan analyzer. It is responsible for trying to guess the morphological tag for words that do not exist in the system's dictionary and thus could not be tagged. This chapter describes the first describes the changes made to it necessary to support the modifications made to the Guesser submodule and then the enhancements made to the Guesser submodule itself to improve its capability to guess lemma and PoS of unknown words. Typically, unknown words can be of two types: truly unknown words (new, but correctly formed words) or they can be known words with typographical errors. The modifications tackle both these categories.

## 5.1 General Architecture

The Guesser is a submodule of LexMan analyzer. The analyzer is responsible for taking a text input and analyzing it. The process can be seen in figure 5.1. The circles represent data, the rectangles represent submodules. Rectangles in green represent submodules that were modified and the thick green arrows represent new data flow.

The process starts with the inputted plain text. The input goes through the tokenizer, which, among

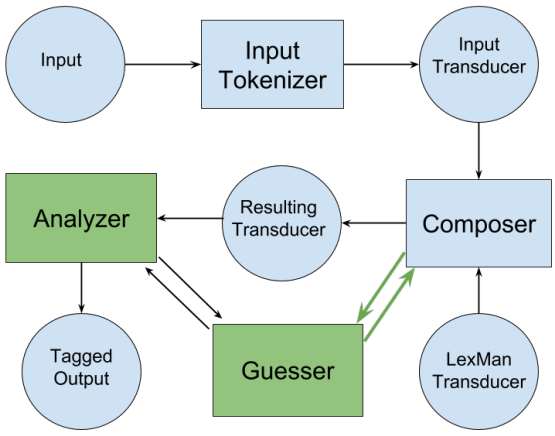


Figure 5.1: LexMan analyzer process

other things, transforms the input into a corresponding transducer. This transducer is then composed with the final transducer, produced by the LexMan generator. The resulting transducer is the result of the composition of the previous two. The Analyzer module then traverses this resulting transducer looking for words. At this point, for each word, two things can happen: either a word was in the LexMan's transducer, in which case the resulting transducer contains the lexical information regarding it (lemma and PoS tagging); or the word was not in the LexMan's transducer, in which case the Analyzer asks the Guesser to make a guess about that word.

The Guesser takes the unknown word and tries to make an educated guess regarding its lemma and PoS. For this, the guesser looks for specific terminations or for special diacritics in the unknown word form and, then, it can either directly produce a result (lemma and PoS) or generate alternative guesses. These new guesses may need to be transformed into a transducer and composed with the LexMan transducer again as the green arrows show. The resulting composition is analyzed again and if any of the generated alternatives is found, then the lemma and PoS of the alternative are returned from the Guesser to the Analyzer.

The LexMan Analyzer had to be modified to support the Guesser changes. Previously, for an unknown word, the Guesser would immediately return the guessed PoS and lemma to the Analyzer. Now, however, the guessing process may include transducer compositions. Thus it, is necessary to delay the resulting guesses. As such, all unknown words are now processed at the end and all at once, and then their respective guesses returned in bulk to the Analyzer.

## 5.2 Enhancements

This section describes the modifications made to the Guesser submodule, with the objective of improving its capability of guessing morphological information about unknown words. Figure 5.2 shows the new Guesser internal architecture.

Instead of one submodule responsible for producing guesses, the Guesser has now four (the first three being new). An unknown word passes through each of the submodules in order (from left to right, as the arrows show). If a given submodule produces alternatives then subsequent submodules are ignored. The last submodule, the Regular Guesser, always produces a guess.

### 5.2.1 Direct Substitution

The first submodule deals with the most common typographical errors. The module enables common errors to be substituted by their correct PoS tagging, entries can be added or removed as necessary. The Direct Substitution submodule first reads information from a file where all the substitutions are defined. The file currently has 11 words and their corresponding correct PoS tag and has the follow syntax:

```
word1-with-error        ([correct-lema]tags)+
...
word2-with-error        ([correct-lema]tags)+
```

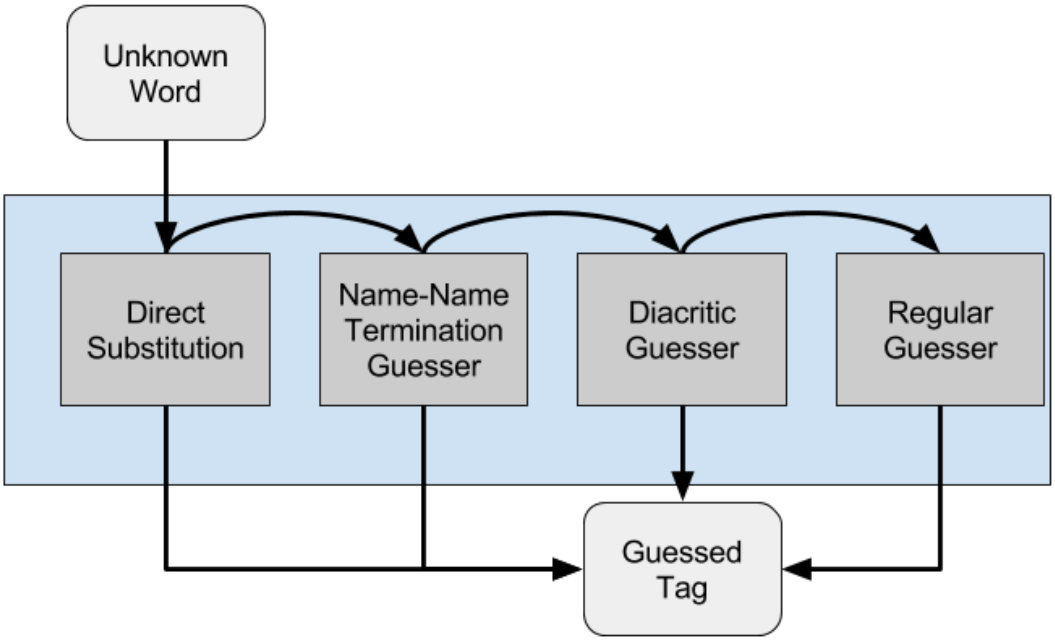


Figure 5.2: Guesser internal architecture

Each line of the file specifies a typographical error to treat and then all of its correct lemmas and tags.

```
hà      [haver]V.ip3s=..=v
Àfrica  [África]Np...sfn.=v [áfrica]Nc...sfn.=v [áfrico]A....sfn.=v
```

**Example 6.** Direct substitution file

The example above shows an example of common mistakes found in texts, mostly stemming from the wrong type of diacritic being used. The first example is the typing spelled word *hà* instead of its correct form *há*, which is a form of the verb *haver*. The second being the wrongly spelled word *Àfrica* instead of *África*. In front of each of the words are all of the morphological tags corresponding to the word's correct form.

### 5.2.2 Noun-Noun Compounds

Another of the enhancements made to the Guesser submodule deals with a particular case of name-name compound words that are not automatically generated. In this particular compound only the first element's morphological category matters for the compound's PoS tag. Examples of such compound word would be words terminated in *-alvo*, *-chave*, *-mãe*, *-mestre*, *-padrão*, such as *apoio-chave* ('key support'), *jogador-chave* ('key player'); *língua-alvo* ('target language'), *mercado-alvo* ('target market'); *escrivão-mestre* ('scribe master'), *sargento-mestre* ('master-sergeant'); *língua-mãe* ('mother language'), *rainha-mãe* ('queen mother') and *operário-padrão* ('standard worker'), *desvio-padrão* ('median deviation'). If we were to classify the word *língua-mãe*, it would be classified as a noun, singular and feminine

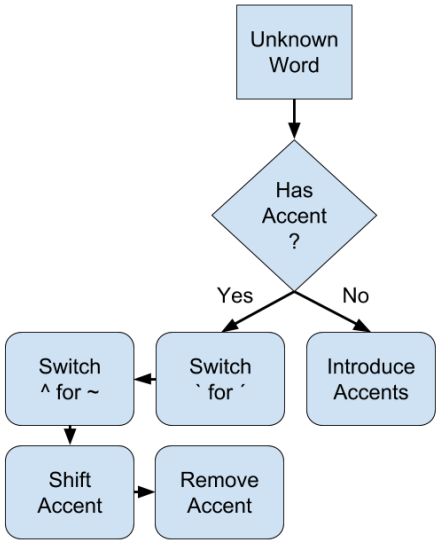


Figure 5.3: Diacritic alternative generation algorithm

because that is the morphological classification of the first element *língua*. Some of these compounds have already a long-standing lexicographic tradition (*rainha-mãe*, *desvio-padrão*), while others are real neologisms, still in the process of being coined and disseminated.

To achieve this, the Guesser submodule had to be modified. One of its subroutines verifies if the word being guessed has one of the ending elements mentioned above. These ending elements are specified in a single text file that the subroutine reads. If there is a match, then the system needs to verify if the first element of the compound is an actual word and a noun. For this, the system writes the first element to a file. This file is then converted to a transducer, which is composed with the LexMan's transducer and the result is analyzed. It is important to note that for performance issues (further detailed below) this file is only converted to a transducer and composed with LexMan's transducer once for each text being processed.

If the word is in the dictionary and a noun, then those morphological tags are attributed to the unknown, compound word. The single difference is that the unknown (but now guessed) compound word will be tagged as *not in dictionary*, since it was guessed.

### 5.2.3 Diacritic Correction

The third enhancement made to the Guesser submodule tries to deal with incorrectly placed diacritics, which are a common cause of unknown words such as *ràpido*, instead of *rápido* ('quick') or *rápidamente* instead of *rapidamente* ('quickly').

To achieve this, the Guesser submodule has yet another routine, created to deal with diacritics (or the lack of them). The general algorithm can be seen in figure 5.3.

If a word has a diacritic then the subroutine finds where (which letter in the word), then, generates possible alternatives. These alternatives are generated as follow: i) if the word has a grave accent then it is replaced by an acute accent (e.g. 'ràpido' becomes 'rápido'); ii) then, if the word has a circumflex

| Vowel with Diacritic | Alternatives |
|---|---|
| a | à á ã â a |
| e | é ê e |
| i | í i |
| o | ó õ ô o |
| u | ú u |

Table 5.1: Diacritic alternatives

accent, it is switched by a tilde accent (e.g. ‘câo’ becomes ‘cão’); iii) then, the accent is shifted through all of the word’s vowels (e.g. ‘acessivél’ becomes ‘ácessivel’, ‘acéssivel’ and ‘acessível’); iv) then, the accent is switched by all other accents that can occur in that particular vowel (e.g. ‘câo’ becomes ‘cào’, ‘cáo’ and ‘cão’); and, finally, v) the accent is removed (e.g. ‘rápidamente’ becomes ‘rapidamente’).

Possible alternatives are generated from a static list, which can be seen in Table 5.1. This table includes all the diacritic alternatives to that particular vowel in Portuguese, as well as the possibility that the diacritic may have been inserted by mistake.

As such, the word *ràpidamente* would produce the alternatives: *rápidamente*, *rapídamente*, *rapidámente*, *rapidaménte*, *rapidamenté*, *rãpidamente*, *râpidamente* and ra*pidamente*. The subroutine then writes all the generated alternatives to a file (the same in which noun-noun compound words have been written). This file is transformed into a transducer, and, as before, it is composed with LexMan’s final transducer, the result being analyzed then. If any of the alternatives is in the dictionary, then the Guesser returns that word’s morphological tags (including the ’not in dictionary’ tag).

#### Default Guess

It is possible that the unknown word produces no result for the three submodules, in which case a default guess is made. This guess is made by taking into consideration the word’s form (mostly its ending). For example, Words ending in *-o* are considered to be of the masculine gender while those ending in *-a* are considered feminine; words ending in *-mente* are considered adverbs. This last submodule always produces a guess.

#### Alternative Generation

As described above, two of the submodules (noun-noun compound guesser and diacritic guesser) constituting the Guesser produce alternatives that need to be checked in LexMan’s transducer. For this, such alternatives are all written to the same file and this file is only converted to a transducer and composed with LexMan’s final transducer when the whole input has been processed. This is done last and only once because composition operations with transducers are complex.

For the unknown words *jogador-chave* and *ràpidamente*, the file would have the contents in example 7.

```
jogador # rápidamente . ràpidamente . rapidàmente . rápidamente . rãpidamente .
râpidamente . rapidamente
```

**Example 7.** Alternatives file example

# Chapter 6

# Guesser Evaluation

The Guesser submodule is part of the LexMan's analyzer and, as such, it is important that trying to classify unknown words does not slow down the whole process. This chapter describes the evaluation and correspondent results made to test the impact of the enhancements made to the Guesser submodule.

## 6.1 Methodology

Modification to the guesser submodule can impact LexMan's analyzer in two ways: time and accuracy, that is, the time it takes for the system to process a given input and the percentage of correct results it produces for said input. As such, both factors were evaluated separately.

For this evaluation, an excerpt of the CETEMPúblico corpus was used (the same used to evaluate the accuray of the Suffixed Word Generator module). This corpus excerpt contained 414,132 sentences and has been split into 208 files, each one containing 2,000 sentences

## 6.2 Time

For this test both versions (original and new) of the Guesser submodule processed the excerpt of CETEMPúblico containing 208 files. The results (measured in minutes) are shown in table 6.1.

As the table shows there was an increase in the processing time of the corpus from 114 to 146 minutes, an increase of 28%. This increase is expected as the modified Guesser does a much more complex task than the original one, and in most of the cases the new Guesser requires a second transducer composition to verify the guesses, which is a time-intensive operation. This, coupled with the fact that the corpus is split between 208 files, means that we are making 416 transducer operations, instead of just 208.

| **Original Guesser** | **Modified Guesser** |
|---|---|
| 114 minutes | 146 minutes |

Table 6.1: Guesser time evaluation

| | |
|---|---|
| Unique Guessed Words | 48,125 |
| Unique Differences | 481 |
| Correct Guesses | 422 |
| Accuracy | 87% |

Table 6.2: Guesser accuracy

## 6.3 Accuracy

To measure the accuracy difference between both versions of the Guesser submodule, both processed the same corpus. The output of both was first filtered for guessed words and then duplicates removed. The differences between outputs were then manually analyzed. Differences were categorized as either correct, or incorrect, depending on whether the difference represents a correct guess or not. For example, if the unknown word *emprésario* was being guessed as a form of the verb *emprésariar* (does not exist) and is now being guessed as a form of the noun *empresário* ('businessman'), then the difference is considered correct as it represents an improvement.

The results are shown in Table 6.2.

As the table shows, of the 48,125 unique guessed words, there were 481 unique different guesses between the original and the modified Guesser, roughly 1%, 422 of which were considered correct, thus, corresponding to an improvement of 87%.

Further analysis of the outputs shows that some of the incorrect guesses stem from foreign words (mostly French) some being written in a Portuguese style, in which cases, it is very difficult to make an accurate guess. For example, the word *décalage* was guessed as the prefixed form of the noun *lage* (*déca-* 'ten' + *lage* 'cobblestone'), which is, obviously, incorrect. Some other errors stem from words that were missing in the lexicon, for example, *ciclope*, which should have been spelled as *cíclope* ('cyclops') but, instead, was guessed as a prefixed form of *pé* (*ciclo-* 'cycle' + *pé* 'foot').

Words with multiple typographical errors are also impossible to correct with the current system, however, some produce erroneous results. The word *retrógado* (as opposed to 'retrogrado') has two errors but since the system only deals with diacritics it will never find the correct form. However, in cases such as the one above it finds other possible words, which exist but are not the correct guess. For the above example the guessed form was *retrogado*, a prefixed form of *gado* ('cattle') formed from *retro+gado*.

Some differences in accentuation between European Portuguese and Brazilian Portuguese also produce some errors. The word *camelô*, unknown to the system was guessed as the word *camelo* ('camel'), when in Brazilian Portuguese it actually means *vendedor ambulante* ('peddler').

An important difference to consider between the modified and original Guesser is that when guessing unknown words, the original Guesser would produce guesses for all the grammatical categories. For example, for the unknown word *autómona* (opposed to the adjective *autónoma*) previously was guessed as a form of the adjective *autóme*, a form of the adjective *autómo* and a form of the noun *autómona*. While all these guessed forms were incorrectly guessed (none of them exist), some of them had been given the correct part-of-speech (adjective). With the original Guesser, prefixes were ignored but PoS tags were sometimes correct even if the word did not exist. Now, with then ew Guesser, the morphologi-

cal analysis, includng prefixes, yields some incorrectly guessed forms, and their PoS are often incorrect, which may later have some impact in syntactic and semantic analysis.

# Chapter 7

# Conclusions

This dissertation tackled the problem of automatically identifying suffixed words in Portuguese. This work was integrated in LexMan [35], the morphological analyze of STRING [14]. Some enhancements and corrections were also done in other parts of LexMan such as fixing the incorrect generation of certain compound words.

In order to identify suffixed words, a new module was developed within LexMan, the Suffixed Words Generator. This module is based on a Two-Level Morphology approach, making use of a lexicon (LexMan's non-suffixed simple and compound words and a lexicon of productive suffixes) and a set of rules to generate suffixed words. This is currently done for the seven most productive suffixes.

This new module introduced over 500,000 new suffixed words to LexMan's lexicon. The module barely increased the processing time of the system, showing only a 3.1% time increase for 10,000 sentences, when compared to the original system. The number of words processed per millisecond remained roughly the same. Using an excerpt of the CETEMPúblico corpus containing 414,132 sentences, the system correctly tagged 859 out of 917 unique suffixed words, achieving an accuracy of 93%. Many of the errors stemmed from problems in both the suffix paradigms and the Suffixed Word Generator and have since been corrected, further increasing the accuracy for the same input.

This dissertation also tackled the problem of attributing a morphological tag to unknown words. This work was also done in the LexMan framework. To perform this task, the existing Guesser submodule was modified to treat three particular cases of unknown words: common unknown words, which the Guesser now automatically identifies and replaces with the correct tag; noun-noun compound unknown words, in which the Guesser verifies if the unknown word has a specific termination and if so, it verifies if the first element of the compound if valid and; diacritical errors in unknown words, for which the Guesser tests different variations of the diacritic position, as well as different diacritics in th same position, trying to find a possible, valid alternative to the unknown word form.

This solution was compared with the original Guesser by processing an excerpt of CETEMPúblico. The outputs were compared and differences concerning unknown words manually analyzed and categorized. Of these 481 differences the new Guesser correctly guessed the morphological tags of 422 of these differences, achieving an accuracy of 87%.

While this work introduced many improvements to the LexMan system there are still some enhancements to be made. More suffixed words can be generated by developing new Suffix Paradigms for other (less productive) suffixes (e.g. augmentative *-ão*, adjective nominal suffixes *-ista*, *ólogo*, *-or*, etc.).

For the Guesser submodule, there are still many strategies that can be used to further improve its accuracy, such as inserting, replacing or removing characters from unknown words (possibly characters assigned to keys ergonomically closer to each other in the Portuguese keyboard). To limit the generation of diacritic alternatives one could use an hyphenator yet again(such as YAH), to discard alternatives that are not possible in the Portuguese language or even trying to look at the context of the word to decide between two alternatives (e.g. between *crítica* 'criticism' and *critíca* 'critic'). A possibility would be to look at the surrounding words, to decide between alternatives.

Another possibility would be to use a threshold when making guesses, that is, using several methods to generate alternatives and then evaluate the alternatives generated in order to use them only if the system level of certainty is above a certain percentage value.

# Bibliography


[1] Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, Volume 4783 of *Lecture Notes in Computer Science*, pp. 11–23. Springer. `http://www.openfst.org`.

[2] Ananiadou, S., A. Ralli, and A. Villalva (1991). The Treatment of Derivational Morphology in a Multilingual Transfer-based MT System (Eurotra). *Language Research 27*(4), 627–638.

[3] Baptista, J. (2004). Suppletive morphology: How far can you go? *paLavra 12*, 149–163.

[4] Bick, E. (2000). *The parsing system PALAVRAS: Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Arhus: University of Arhus.

[5] Cabrita, V. (2011). Identificar, ordenar e relacionar eventos. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[6] Carapinha, F. (2013). Extração automática de conteúdos documentais. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[7] Diniz, C., N. Mamede, and J. Pereira (2010). Rudrico2 - a faster disambiguator and segmentation modifier. In *II Simpósio de Informática (INForum)*, pp. 573–584.

[8] Eleutério, S., E. Ranchhod, H. Freire, and J. Baptista (1995). A system of electronic dictionaries of Portuguese. *Lingvisticae Investigationes 19*(1), 57–82.

[9] Eleutério, S., E. Ranchhod, C. Mota, and P. Carvalho (2003). Dicionários Electrónicos do Português. Características e Aplicações. In *Actas del VIII Simposio Internacional de Comunicación Social*, Santiago de Cuba, 2003, pp. 636–642.

[10] Figueirinha, P. (2013). Syntactic REAP - exercises on word formation. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[11] Forney, G.D., J. (1973). The viterbi algorithm. *Proceedings of the IEEE 61*(3), 268–278.

[12] Koskenniemi, K. (1983). Two-level model for morphological analysis. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*, pp. 683–685.

[13] Loureiro, J. (2007). Reconhecimento de entidades mencionadas (obra, valor, relações de parentesco e tempo) e normalização de expressões temporais. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[14] Mamede, N., J. Baptista, C. Diniz, and V. Cabarrão (2012). STRING: A Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese. In *International Conference on Computational Processing of Portuguese (PROPOR'2012)*, Volume PROPOR'2012 Demo session.

[15] Marques, J. (2013). Anaphora resolution in Portuguese. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[16] Maurício, A. (2011). Identificação, classificação e normalização de expressões temporais. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[17] Maxwell, M. (1996). Two theories of morphology, one implementation. In *SIL's General CARLA Conference*, pp. 203–230.

[18] Medeiros, J., R. Marques, and D. Santos (1993). Português quantitativo. *Actas do Encontro da Associação Portuguesa de Linguística 1*, 33–38.

[19] Mota, C. (2000). Analysis of Derivational Morphology by Finite-State Transducers. In A. Dister (Ed.), *Revue Informatique et Statistique dans les Sciences Humaines*, Volume 36, Liège, Belgium, pp. 273–287. Université de Liège.

[20] Mota, C. (2003). A Renewed Portuguese Module for Intex 4.3x. In M. Silberztein and S. Koeva (Eds.), *Proceedings of the 6th Intex Workshop*, Sofia, Bulgaria, May 28-30, 2003.

[21] Mota, C., P. Carvalho, and A. Barreiro. (2016). Port4NooJ v3. 0: Integrated Linguistic Resources for Portuguese NLP. In *10th Intl. Conference on Language Resources and Evaluation (LREC 2016)*, pp. 1264–1269.

[22] Muniz, M., M. Nunes, and E. Laporte (2005). Unitex-PB, a set of flexible language resources for Brazilian Portuguese. In *Workshop on Technology on Information and Human Language (TIL)*, pp. 2059–2068.

[23] Oliveira, D. (2010). Extraction and classification of named entities. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[24] Paumier, S. (2003). *De la reconnaissance de formes linguistiques à l'analyse syntaxique*. Thèse de doctorat, Université de Marne-la-Vallée, Paris.

[25] Ranchhod, E., C. Mota, and J. Baptista (1999). A Computational Lexicon of Portuguese for Automatic Text Parsing. In *Proceedings of SIGLEX99: Standardizing Lexical Resources, 37th Annual Meeting of the ACL*, pp. 74–80. Association for Computational Linguistics.

[26] Ribeiro, R. (2003). Anotação morfossintáctica desambiguada do português. Master's thesis.

[27] Rocha, P. and D. Santos (2000). CETEMPúblico: Um corpus de grandes dimensões de linguagem jornalística portuguesa. In *V Encontro para o processamento computacional da língua portuguesa escrita e falada (PROPOR'2000)*, Volume PROPOR'2000, pp. 131–140.

[28] Rodrigues, R., H. Oliveira, and P. Gomes (2014). LemPORT: a high-accuracy cross-platform lemmatizer for Portuguese. In *SLATE 2014*, Volume 38. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[29] Romão, L. (2007). Reconhecimento de entidades mencionadas em língua portuguesa: Locais, pessoas, organizações e acontecimentos. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[30] Santos, D., L. Costa, and P. Rocha (2003). Cooperatively evaluating Portuguese morphology. In *International Workshop on Computational Processing of the Portuguese Language (PROPOR 2013)*, pp. 259–266. Springer.

[31] Silberztein, M. (1993). *Dictionnaires électroniques et analyse automatique de textes: le système INTEX*. Paris: Masson.

[32] Silberztein, M. (2016). *Formalizing Natural Languages: The NooJ Approach*. John Wiley & Sons.

[33] Silva, J. (2007). Shallow processing of Portuguese: From sentence chunking to nominal lemmatization. Master's thesis, Universidade de Lisboa - Faculdade de Ciências, Lisboa.

[34] Simões, A. and J. Almeida (2001). jspell.pm – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas do XVII Encontro da Associação Portuguesa de Linguística*, pp. 485–495.

[35] Vicente, A. (2013). LexMan: um segmentador e analisador morfológico com transdutores. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa.

[36] Xerox (2003). *Xerox Incremental Parsing - Reference Guide*. Xerox.