Suffix Identification in Portuguese using Transducers

Hugo Almeida[1,2], Nuno Mamede[1,2], and Jorge Baptista[2,3]

[1] Universidade de Lisboa - Instituto Superior Técnico
[2] L^{2}F - Spoken Language Systems Laboratory - INESC-ID Lisboa
[3] Universidade do Algarve - Faculdade de Ciências Humanas e Sociais
hugo.almeida@l2f.inesc-id.pt
nuno.mamede@inesc-id.pt
jbaptis@ualg.pt

Abstract. This paper[1] addresses the challenge of automatically identifying suffixed words not previously encoded in the lexicon of a Natural Language Processing system, STRING, and correctly tagging them with their PoS and lemma, as well as all relevant linguistic information. To date, five of the most productive suffixes in Portuguese have been described. The performance of this solution was then evaluated.

Keywords: natural language processing, transducers, suffixation, derivation morphological analysis

1 Introduction

For many Natural Language Processing (NLP) tasks, the lexicon is key. Many processing steps rely on an accurate part-of-speech (PoS) tagging, requiring adequate lexical resources with a high granularity and broad lexical coverage. A particular challenge in lexical analysis is *derivation*, a linguistic device through which languages create new lexical items from pre-existing lexical units. Derivation is a recursive process that applies to both non-derived (or *base* form) and derived forms. 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 they are attached to. Complex restrictions may apply in these processes. This work focuses on *suffixes*, which are morphemes added at the end of the base. Derivation by suffixation can not only change the meaning of base word but also its PoS. For example, the adjective *rápido* 'quick' can yield the adverb *rapidamente* 'quickly' by adding the adverbializer suffix *-mente* '–ly'.

In Portuguese, the correct identification of derived words is an important feature in the lexical analysis of many NLP systems. Since derivation is a often a productive phenomenon, no lexicon could be comprehensive enough to include all possible, regularly derived word forms for a given base vocabulary, and a general

[1] This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

mechanism is required to automatically recognize them. Additionally, derived words need to be associated to their adequate PoS, in order to allow for the parsing stage, as well as to the lexical (syntactic-semantic) information available for the base form of the derived word, and so enable the correct syntactic and semantic processing of text. This, in broad strokes, is the challenge that 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 paper to produce a comprehensive review of previous work on the topic, hence, just the briefest overview is provided here. Concerning derivational morphology, in the context of the transfer-based machine-translation (MT) system Eurotra, [1] 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* [2] at INESC-ID Lisboa; (ii) *Palavras* [3]; (iii) *jSpell.pm* [4]; and (iv) *Digrama* [5, 6], which later lead to the Label-Lex-sw lexicon [7].

The availability of user-friendly finite-state tools like *Intex* [8] and later *Unitex* [9] lead to the adaption of existing resources to these platforms [10], including Brazilian Portuguese [11], and the development of several morphological analysis' modules and experimentations in Portuguese derivational morphology [12, 13], a trend that is still pursued today [14] within the NooJ platform [15]. These early works lead to the Morpholimpics joint evaluation contest in 2003 [16], coordinated by Linguateca[2]. More recent work, in different approaches, can also be found in [17] and [18], the later built over the lexicon of [7]. Evaluation on the specific topic of suffixal derivation is scarce, but some data is provided by [12] and [13].

1.2 Goal

Currently, LexMan [19], the lexical analyser of the STRING system [20], supports the identification of prefixed words. This is done by applying derivation rules to base words and then attributing their correct PoS, which can later be used for syntactic analysis. Unlike prefixes, which could be concatenated with the base word without much change in its inflection nor altering their PoS, suffixes are attached to the root of the base, and the inflection values of the derived word must be found at the word ending. For example, for the word *escadas* 'stairs', the module would produce the diminutive forms with suffixes *-inho* and *-ito*: *escadinhas* and *escaditas*. Notice that the plural-feminine morphemes *-as* appear after the suffix.

The purpose of this work is, thus, to extend the morphologic analysis functionality of LexMan, in order to automatically identify out-of-vocabulary, but

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

regularly derived, suffixed words. For this, a new LexMan submodule was developed, which, for a given word, identifies and produces all of its suffixed forms. Currently, and in spite of the large number of suffixes in Portuguese, only for the most productive suffixes were considered, namely: diminutive suffixes *-inho* and *-ito*; superlatives *-íssimo* and *-érrimo*; adjectivalizer *-vel* '-ble' and corresponding nominalizer *-bilidade* '-bility'; and adverbializer *-mente* '-ly'. 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 element can undergo regular derivational processes, like the diminutive *pezinhos-de-galinha*, in as much as ordinary simple words (*pé*/*pezinho*).

2 STRING Architecture

STRING [20] is a *STatistical and Rule-based Natural lanGuage processing chain* for the Portuguese language[3]. The system has a pipeline structure and is composed of several modules. The first module of the chain is a morphological analyzer, LexMan [19]. This module is responsible for splitting the input text into segments (sentences and tokens) and for attributing morphological tags to tokens. LexMan also generates the dictionaries of the system, using a two-level morphology [21] approach, that is, by combining a lexicon of lemmas with a set of inflection paradigms that for allow the generation, for a given lemma, of all the inflected forms that are associated with it. Words are generated by applying to a base form (called *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 their initial characters and performs the necessary morphosyntactic adjustments (inserting or removing characters) to adequately generate the derived, prefixed, word.

3 Suffixed Word Generation

This section describes how the new module, the Suffixed Word Generator, was developed and integrated into LexMan to tackle the problem of recognizing out-of-vocabulary and regularly derived suffixed words. Figure 2 shows the new architecture of LexMan, while Figure 3 describes the internal structure of the new module.

This new module generates suffixed words using the same two-level morphology approach as the Word Generator. However, instead of manually tagging each word in the LexMan base dictionary with its correct suffixation rule, this is done automatically through a process described below. 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.

[3] https://string.l2f.inesc-id.pt

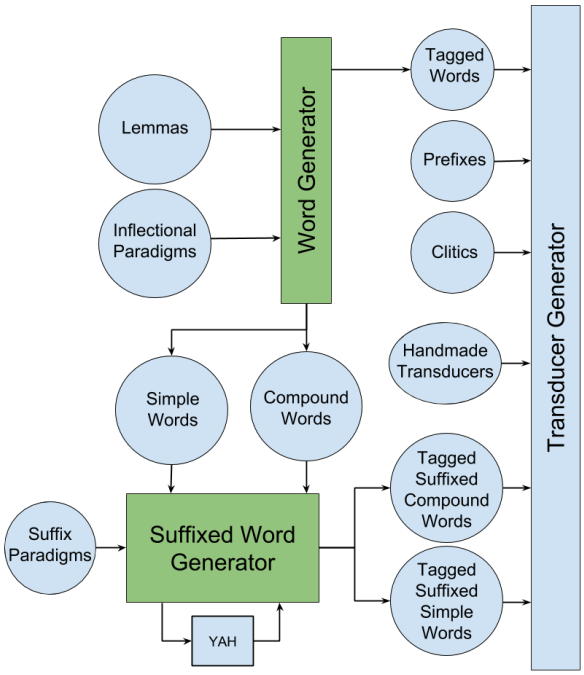

Fig. 1. The new LexMan architecture

Parsing The files, containing both simple and compound words, are processed and parsed by the module. Each line of these files corresponds to a word that is to be suffixed. The files are then given as input to the rest of the module. The syntax of the input file is different for simple and compound words:

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

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

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*. The *Suffix Dictionary* file specifies the suffixes that should be considered. Its syntax is presented below:

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

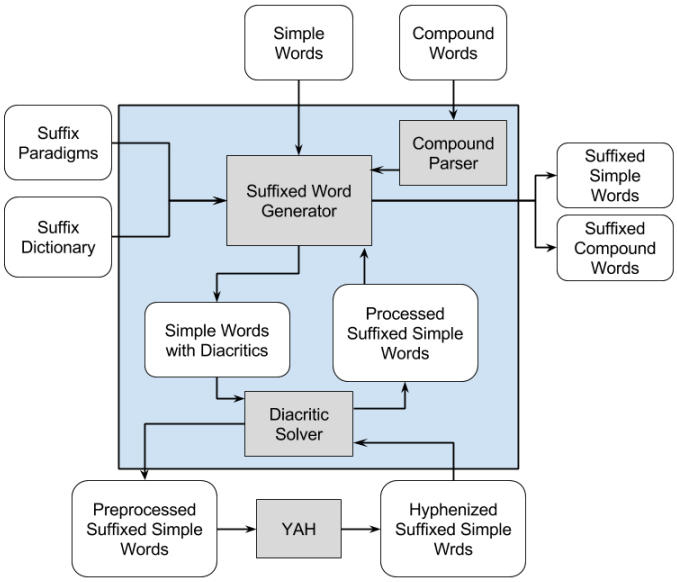

Fig. 2. The inernalstructure of the Suffixed Word Generator module

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 is indicated.

A *Suffix Paradigm* file specifies, for a given suffix, how the suffixed words are formed. This 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, so other types of exceptions must be considered for a given suffix or suffixation rule: (i) *file* exceptions, for all stems coming from a given lexicon file, the entire file is ignored (eg. gentilic adjectives and nouns for diminutive suffixes); (ii) *lemma ending* exceptions, for all stems with a given termination (e.g. *-âneo* termination for the diminutive suffix *-inho*) are

ignored; and (iii) and *lemma* exceptions, for specified lemmata to be excluded from the suffixation process.

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. It is possible that a word PoS matches with no suffixes (and thus cannot be suffixed); or that is matches one, several, or even all suffixes. 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, delimited by the square brackets, 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 generated two suffixed words:

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

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') and adds the suffix along with a linking consonant *-z-*, changing the morphological values from normal to diminutive (`d`), while indicating that this word has been derived by suffixation (`x`). The second rule is similar, but it applies to *-ico* ending words(eg. *rico* 'rich'); the suffixation can be achieved, either by using the linking consonant *-z-*, or by directly attaching the suffix to the root ofthe base, along with the orthographic-morphotactic change of <c> into <qu>.

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, then

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. For example, the word *amigo*, 'friend', matches with the two following rules since it ends in *–o* but also ends with *–igo*. In this case, only the most specific rule is used, *i.e.* the one that removes more characters from the end of the base word. Otherwise, as the input string matches with the first rule, the nonexistent word **amiginho* would then be inadequately generated.

```
**...smn.** <o><inho>          0 **...**d.x* []
**...smn.** <igo><iguinho>     0 **...**d.x* []
```

Diacritic Removal 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 adjetive *rápido* 'quick'. The suffixed word must then lose the graphical diacritic <á> of its base form.

This module takes a word and, with the help of the YAH hyphenator [22], it hyphenates the word, dividing it by its syllables using the '=' character. Words with the acute sign <´> in any of the last three syllables must have such diacritic removed (other diacritics than acute are kept, e.g. *órgão* 'organ'/*orgãozinho*). Thus, for the suffixed word `rápidinho` as generated by the suffix paradigm, the YAH would output the following hyphenated form `rá=pi=din=ho`; the word would then lose its diacritic to produce the correct form of the suffixed word, *rapidinho*.

Finally, words in LexMan may contain regular expressions in them, to express an optional character (e.g. *a*`c?`*to* 'act') or a choice between several characters in a given position (e.g. *o*`[iu]`*ro* 'gold'). YAH, however, does not allow the use of regular expressions and these words must first be preprocessed, and have their regular expressions removed. After YAH processes the word, regular expressions are reintroduced and the diacritic marks removed as necessary.

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 is suffixed, but just one of its elements. For example, in the compound word *chapéu-de-chuva* 'umbrella', lit. 'hat-of-rain', only the first noun *chapéu* 'hat' is suffixed, forming *chapeuzinho-de-chuva*.

The element receiving the suffix depends on the type of compound. Table 1 shows the different types of compounds used in LexMan, their internal structure, the suffixation rule and an example. If a compound word can be suffixed, the

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	*aninho-luz*

Table 1. Compound Types

respective element goes through the normal suffixation process, the only difference being that it does not need to pass through YAH again, since all diacritic removals have been stored in a map and thus the final form of the word can be checked without additional processing time.

Transducer Generation Both simple and compound suffixed words are then merged into a single file. This file is checked for duplicates, which are removed if present. The file is then used to generate part of the final transducer, used to analyze inputted text. Figure 4 shows an example of a transducer that can recognize both the simple word *gato* 'cat' and one of its suffixed form, *gatinho*.

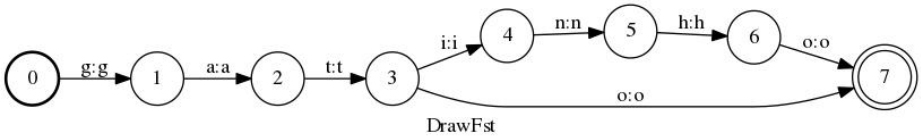

Fig. 3. Transducer for simple and suffixed word.

4 Evaluation

Several parameters were evaluated to measure the impact of the changes described above: processing time, memory usage, transducer size and generation time. To measure processing time and memory usage of the system along its development, several, different-sized files have been previously built, ranging from 1 to 10,000 sentences. These tests were extracted from one of the parts of the CETEMPúblico corpus and their details can be seen on Table 2.

The baseline for this evaluation is the performance of LexMan before any of the changes described above was implemented, which will be referred as *Before*

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 2. Files used in the performance evaluation

Suffixes. The comparison scenario includes all the changes described above and will be referred as *After Suffixes*. Table 3 shows the evaluation results for both scenarios. Each file was processed 4 times, an average of all run times was then calculated for each file. Results shows the average time necessary to process each file (in seconds), as well as the millisecond/word ratio. Analysis of the table permits to conclude that the changes add a constant time overhead, not depending on the amount of words processed.

File Name	Before Suffixes Time (s)	Before Suffixes Time (ms/w)	After Suffixes Time (s)	After Suffixes 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 3. Evaluation results for both scenarios

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 artificially suffixed words. These artificial words were formed by adding 'dummy' suffixes to a list of 100,000 simple adjective words. For example, adding *–xpto* to simple adjective words such as *bonito* 'pretty' to form the artificial sufixed word *bonitoxpto*. For the subsequent increments of 100,000 this dummy termination was replaced by another, such as *–otpx*. The new, artificially enlarged, lexicons were then added to the system and the same files described in Table 3 (but only those with 1 sentence and up to 1000 sentences) were processed anew. Results are shown in Table 4.

File Name	**Suffixes** Time (s)	**+100k Words** Time (s)	**+200k Words** Time (s)	**+300k Words** Time (s)
Parte08-1.txt	6.37	7.31	7.64	8.34
Parte08-10.txt	6.52	7.50	7.83	8.53
Parte08-100.txt	7.58	8.62	8.85	9.56
Parte08-500.txt	12.25	13.57	13.49	14.25
Parte08-1000.txt	18.30	19.19	19.55	20.31

Table 4. Evaluation results for 100k increments of words

As we can see, 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 the results of Table 3.

We now turn to the memory required to process the previously mentioned files, comparing the two scenarios. Results in Table 5 are shown in megabytes (MB). The values in thos table were registered during the composition operation, that is, when the transducer for the input lexicon and the transducer of the tokenizer are being composed. Table 5 shows an increase in the memory usage as the size of the input grows for both environments. It also shows that, for most cases, the *After Suffixes* scenario requires more memory than *Before Suffixes* scenario, as expected.

File Name	**Memory Usage (MB) Before Suffixes**	**Memory Usage (MB) After Suffixes**
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 5. Memory Usage before and after Suffix changes

Table 6 measures the impact of the changes introduced in LexMan on the size of the transducers built before and after the changes to LexMan. As it can be seen by comparing results in the table, the generation time increased by an average of 4 seconds and the transducer size increased 80 megabytes, (36%). Likewise, the number of states and arcs of the transducer also increased 39% and 35.5% respectively. To deal with these five suffixes, 292 rules were created. This set of rules is, however, not yet complete as there are still some words that do not match with any rule. In total, the new module introduces 529,232 new (suffixed) words in the system's lexicon. Of these new words, 246,316 are

adjectives, 282,916 are nouns; 209,824 are compound words and 319,408 are simple words. The evaluation of the linguistic adequacy of these new entries and their impact in the performance of the STRING system will be the topic of another paper.

	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 6. Generation time, size of the transducer, number of states and arcs

5 Conclusions

This work addressed the problem of identifying out-of-vocabulary suffixed words in Portuguese texts using transducers. The results obtained in comparison with the baseline (before suffixes) were very positive, over 500,000 new words were added to their lexicon costing only an average of 2 seconds in text processing, and with an acceptable increase in memory usage. There is still room for improvement, naturally. It is possible to increase the number of identified suffixed words by extending the described paradigms to other suffixes than those five this work focused on).

References

1. S. Ananiadou, A. Ralli, and A. Villalva. The Treatment of Derivational Morphology in a Multilingual Transfer-based MT System (Eurotra). *Language Research*, 27(4), 1991.
2. J. Medeiros, R. Marques, and D. Santos. Português quantitativo. *Actas do Encontro da Associação Portuguesa de Linguística*, 1:33–38, 1993.
3. E. Bick. *The parsing system PALAVRAS: Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. University of Arhus, Arhus, 2000.
4. A. Simões and J. Almeida. jspell.pm – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas do Encontro da Associação Portuguesa de Linguística*, pages 485–495, 2001.
5. S. Eleutério, E. Ranchhod, H. Freire, and J. Baptista. A system of electronic dictionaries of Portuguese. *Lingvisticae Investigationes*, 19(1):57–82, 1995.
6. E. Ranchhod, C. Mota, and J. Baptista. A Computational Lexicon of Portuguese for Automatic Text Parsing. In *Proceedings of SIGLEX99: Standardizing Lexical Resources, 37th Annual Meeting of the ACL*, pages 74–80. Association for Computational Linguistics, 1999.

7. S. Eleutério, E. Ranchhod, C. Mota, and P. Carvalho. Dicionários Electrónicos do Português. Características e Aplicações. In *Actas del VIII Simposio Internacional de Comunicación Social*, pages 636–642, 2003.
8. M. Silberztein. *Dictionnaires électroniques et analyse automatique de textes: le système INTEX*. Masson, Paris, 1993.
9. S. Paumier. *De la reconnaissance de formes linguistiques à l'analyse syntaxique*. Thèse de doctorat, Université de Marne-la-Vallée, Paris, 2003.
10. C. Mota. A Renewed Portuguese Module for Intex 4.3x. In Max Silberztein and Svetla Koeva, editors, *Proceedings of the 6th Intex Workshop*, Sofia, Bulgaria, May 28-30, 2003, 2003.
11. M. Muniz, M.G. Nunes, and E. Laporte. Unitex-PB, a set of flexible language resources for Brazilian Portuguese. In *Workshop on Technology on Information and Human Language (TIL)*, pages 2059–2068, 2005.
12. C. Mota. Analysis of Derivational Morphology by Finite-State Transducers. In Anne Dister, editor, *Revue Informatique et Statistique dans les Sciences Humaines*, volume 36, pages 273–287, Liège, Belgium, 2000. Université de Liège.
13. J. Baptista. Suppletive morphology: How far can you go? *paLavra*, 12:149–163, 2004.
14. Cristina Mota, Paula Carvalho, and Anabela Barreiro. Port4NooJ v3. 0: Integrated Linguistic Resources for Portuguese NLP. In *10th Intl. Conference on Language Resources and Evaluation (LREC 2016)*, pages 1264–1269, 2016.
15. M. Silberztein. *Formalizing Natural Languages: The NooJ Approach*. John Wiley & Sons, 2016.
16. D. Santos, L. Costa, and P. Rocha. Cooperatively evaluating Portuguese morphology. In *International Workshop on Computational Processing of the Portuguese Language (PROPOR 2013)*, pages 259–266. Springer, 2003.
17. J. Silva. Shallow processing of Portuguese: From sentence chunking to nominal lemmatization. Master's thesis, Universidade de Lisboa - Faculdade de Ciências, Lisboa, 2007.
18. R. Rodrigues, H. Oliveira, and P Gomes. LemPORT: a high-accuracy cross-platform lemmatizer for Portuguese. In *SLATE 2014*, volume 38. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.
19. A. Vicente. Lexman: um segmentador e analisador morfológico com transdutores. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, June 2013.
20. N. Mamede, J. Baptista, C. Diniz, and V. Cabarrão. 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, April 2012.
21. K. Koskenniemi. Two-level model for morphological analysis. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*, pages 683–685, 1983.
22. P. Figueirinha. Sintactic REAP - exercises on word formation. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, October 2013.