Finite Automata and Efficient Lexicon Implementation

Tomasz Kowaltowski, Cláudio L. Lucchesi and Jorge Stolfi

Relatório Técnico IC–98-2

Janeiro de 1998

Finite Automata and Efficient Lexicon Implementation

Tomasz Kowaltowski, Cláudio L. Lucchesi and Jorge Stolfi

Abstract

We describe a general technique for the encoding of lexical functions — such as lexical classification, gender and number marking, inflections and conjugations — using minimized acyclic finite-state automata. This technique has been used to store a Portuguese lexicon with over 2 million entries in about 1 megabyte. Unlike general file compression schemes, this representation allows random access to the stored data. Moreover it allows the lexical functions and their inverses to be computed at negligible cost. The technique can be easily adapted to practically any language or lexical classification scheme, and this task does not require any knowledge of the programs or data structures.

1 Introduction

A minimized acyclic finite automaton provides an efficient technique for storing and retrieving a finite set of strings over a finite alphabet. The most obvious usage, within the domain of natural language processing, is the representation of the vocabulary of a language, without any additional information.

The finite automaton is a very compact data structure: for instance, we were able to store a Portuguese vocabulary of over 200,000 words (including inflected verbs, nouns, and adjectives) in 124 kilobytes of storage [14]. Similar storage efficiencies were observed for other languages [12]. In spite of the high compression rate, the data can still be accessed in random order. Moreover, since the entire vocabulary can be kept in memory, even in a small computer, the queries are very fast (a few instructions per letter).

The finite automaton representation compares well in speed and size against the bit-vector hashing method used in the classic Unix `spell` tool [3]. In addition, the finite automaton is exact, and does not require ad-hoc language-specific affix-stripping procedures. Finally, besides simple inclusion tests, the automaton allows also various kinds of word enumeration and pattern searching operations. This last feature is essential for many processing applications, such as spelling correctors and advisors [14], analysis and debugging of vocabularies [11], text compression [17], and word games [1].

*Institute of Computing, University of Campinas, Caixa Postal 6176, 13083-970 Campinas, SP. The research described in this paper was partially supported by Itautec-Philco S. A., by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), and by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Advanced natural language applications generally require more information about the language's lexicon than simply the set of all valid words. Typically, for each word it is necessary to record also its syntactic function (noun, verb, etc.) and other grammatical attributes such as number, gender, tense, declension, etc. Some applications also need to know the connections between lexically related words—for instance, that the infinitive of *had* is *have*, and that the plural of *ox* is *oxen*.

There is an extensive literature dealing with the efficient construction, representation and processing of lexicons [2, 4, 5, 6, 7, 9, 10, 13, 18]. Here we describe a general technique, based on acyclic finite automata, for representing a grammatically-tagged lexicon. In our approach, all the lexical information for a word is encoded as one or more strings, which are then represented by a standard wordlist automaton. With a well-designed encoding, this extra information has a modest impact on the size of the automaton, and can be recovered quite efficiently by standard word lookup and enumeration algorithms.

Compared to other lexicon representations, this technique has the advantage of requiring only one simple data structure—the finite automaton. The technique is also fairly flexible and easy to use: to encode a different language, or a different set of lexical attributes, it suffices to change the string encoding—a task that does not require knowledge of the automaton's internal structure, or sophisticated programming skills.

We have recently used this technique to store a large grammatically-tagged Portuguese lexicon [15], with over two million entries, in less than 1.1 MBytes. From this data structure we can extract the syntactic attributes of any word, and enumerate all its related forms, basically for the same cost as checking whether the word is in the vocabulary.

2 Encoding and retrieval of lexical information

The grammatical attributes of a word can be stored in a wordlist automaton by encoding them as a distinctive string attached to the word. To illustrate our technique, consider the problem of identifying inflected verbs. In modern English, a verb can have up to eight distinct affirmative forms:

infinitive	`be`	`love`
1st person singular, present	`am`	`love`
2nd person singular, present	`are`	`love`
3rd person singular, present	`is`	`loves`
present participle	`being`	`loving`
1st person singular, past	`was`	`loved`
2nd person singular, past	`were`	`loved`
past participle	`been`	`loved`

These inflected verb forms and their tense and person data could be encoded as the strings

`be:vi`	`love:vi`
`am:va1`	`love:va1`
`are:va2`	`love:va2`
`is:va3`	`loves:va3`
`being:vap`	`loving:vap`
`was:vb1`	`loved:vb1`
`were:vb2`	`loved:vb2`
`been:vbp`	`loved:vbp`

(We could reduce the attribute codes to one letter instead of three but the savings would be minimal, since the states that represent those codes are usually shared among thousands of words.)

In this particular encoding we use '`v`' to denote a verb; '`a`', '`b`', and '`i`' to denote present, past, and infinitive; digits '`1`' to '`3`' to encode the subject person; and '`p`' for participles.

Other persons and tenses regularly coincide with one of these forms, and therefore do not need separate entries or marks in the lexicon. In particular, it is understood that '`:vi`' denotes 'imperative' as well as 'infinitive'; '`:va2`' and '`:vb2`' include all plural persons; and '`:vb1`' includes the third person singular past. (A practical lexicon should include also negative verb inflections, such as `isn't` and `cannot`; but the above will suffice for the sake of example.)

Analogous encodings can be devised for other word classes; for example, '`:ns`' and '`:np`' for singular and plural nouns, '`:p3s`' for a 3rd person singular pronoun, and so on.

It should be noticed that any other English verb will use the same grammatical codes '`:vi`', '`:va2`', ..., '`:vbp`', so that their inclusion in the minimized automaton will have negligible effect on its size. As a matter of fact, regular verbs ending in '*e*' will share the longer suffixes '`e:va1`', '`e:va2`', '`es:va3`', ..., '`ing:vbp`'. Therefore, most verbal forms will still belong to large factorable subsets (i.e, collections of prefixes that admit exactly the same suffixes) of the string-encoded lexicon, meaning that the corresponding wordlist automaton will still be quite compact.

2.1 Word classification queries

Once we have built the finite automaton for all words of the lexicon, encoded as above, we can easily and efficiently answer queries like *What is the grammatical class of the word '`were`'?* We need only enumerate all suffixes that may follow the prefix '`were:`' in the encoded lexicon—namely, '`vb2`'. This operation is trivial to code and costs only a few instructions for each letter in the word plus each letter of the answer [14].

For the word '`love`', the answer would be the four strings '`vi`', '`va1`', '`va2`' and '`ns`'. As this example shows, words with multiple grammatical functions require no special care during the automaton construction and the string encoding provides a compact and uniform representation for multiple answers.

2.2 Canonical-form queries

For some applications, knowing the class and inflection of each word is not enough; we may also need to answer queries like *What is the infinitive form of the word '`were`'?* We can still use the automaton to answer such queries if we include the infinitive form of each word as one of the grammatical attributes of its inflected forms. Thus, for instance, the word '*were*' would be entered in the automaton as the string '`were:vb2:be`'.

Unfortunately, this simple scheme would lead to a very large automaton, because paradigms would no longer be factorable. The encoding strings '`loving:vap:love`' and '`working:vap:work`' will no longer be able to share the states of the '`ing:vap`' suffix; and the same will happen to most other verb forms.

This problem can be solved through a more careful encoding of the standard form. Specifically, we encode the word '*loving*' as '`loving:vap:3e`'; the string '`3e`' means that in order to get the standard form we must remove three letters from the end of the given form '`loving`' and append to it the string '`e`'. With this convention we get:

```
be:vi:0          love:vi:0
am:va1:2be       love:va1:0
are:va2:3be      love:va2:0
is:va3:2be       loves:va3:1
being:vap:3      loving:vap:3e
was:vb1:3be      loved:vb1:1
were:vb2:4be     loved:vb2:1
been:vbp:2       loved:vbp:1
```

It is obvious now that almost all verbs ending in '*e*' will have encodings with the same suffixes, so that their factoring will take place. The same technique can be used to connect other classes of inflected words to their standard (dictionary) forms. So, for example, a plural noun could be coded '`automata:np:1on`', and a comparative adjective as '`fancier:ac:3y`'.

2.3 Inflection queries

The techniques described so far allow us to efficiently analyze an inflected word and map it to its standard form. They are not sufficient however to answer text synthesis queries like *What is the first person past form of the verb* 'to be'? To find the desired word '*was*' we would have to search through the whole automaton, enumerating all strings whose grammatical code is '`:vb1`' and checking whether their standard form evaluates to '`be`'. This operation would be very inefficient and its speed would be proportional to the size of the lexicon. To speed up this type of queries, we add to the automaton another set of "reverse" strings which will provide the desired information. For the verbs '*to be*' and '*to love*', these strings would be:

```
be>vi:0          love>vi:0
be>va1:2am       love>va1:0
be>va2:2are      love>va2:0
be>va3:2is       love>va3:0s
be>vap:0ing      love>vap:1ing
be>vb1:2was      love>vb1:0d
be>vb2:2were     love>vb2:0d
be>vbp:0en       love>vbp:0d
```

Thus, for instance, the string '`love>vap:1ing`' says that to obtain the present participle ('`>vap`') of '*love*' we should delete the last letter ('`:1`') and append '`ing`' to what is left. With these strings we can produce a specific inflection of a standard form (or enumerate all its inflections) at the cost of a few instructions per letter.

In conclusion, a verb with all its inflected forms is represented by 16 strings (8 for "analytic" queries and 8 for "synthetic" queries) stored in the automaton. Even though this description looks rather formidable, it should be noticed that most verbs would contribute very little to the growth of the minimized automaton; e. g., the addition of the verb *to move* to the automaton which contains already the strings describing the verb *to love* would create only one additional state. It should be stressed that this factoring happens automatically as part of the minimization process. It is not necessary to identify the "regular" and "irregular" inflections. All words of a certain grammatical class are encoded by the same simple rules and are stored and retrieved by the same simple algorithms.

2.4 Other lexical functions

The general idea of encoding functions as strings could be used to encode other relationships between words, e.g. between '*love*' (noun), '*love*' (verb), '*lover*', '*lovingly*', '*non-lover*', '*lovely*', etc. All that is required is an encoding of each relation as a suffix, in such a way that the the resulting set of strings still contains large factorable subsets.

In particular, the delete-and-append encoding we used to indicate English inflections can be used for most other Indo-European languages, where inflections generally affect the ending of the word. Analogous encodings can be devised for other language families.

3 Case study: A Portuguese lexicon

The ideas introduced in a simplified way in the previous section were applied to a fairly large lexicon for Brazilian Portuguese. The main characteristic relevant to our discussion is that Portuguese has, as other Romance languages, a very elaborate verb system, two genders and singular and plural forms for nouns and adjectives, and many irregularities and exceptions. In what follows, we describe some of the grammatical information recorded in the lexicon and the particular encodings we used.

3.1 Direct verb encodings

A Portuguese verb can be conjugated into 70 combinations of mood, tense, person and number, not counting compound tenses. We encode the corresponding inflected forms as explained in section 2. (Although only 59 of these forms can be distinct, we decided to enter them all, in order to make life simpler for the user. Again, this redundancy has negligible effect on the automaton's size.) For example, the string

`levantaste:1eac2:3r`

describes the word *levantaste* (*you raised*) as being a verb form ('`e`'), indicative mood ('`a`'), perfect past tense ('`c`') and second person singular ('`2`'). The infinitive form is *levantar* ('`3r`'). The initial digit ('`1`') means that this is a form of the first meaning of the word *levantar* – there might be other meanings. The corresponding reverse entry for this form is:

`levantar>eac2:1ste`

Consequently each verb produces the total of 140 direct and reverse encodings.[1]

3.2 Contractions

An additional complication in encoding Portuguese verbs is the existence of a large number of verb-pronoun contractions. Most verb tenses admit only *enclitic* contractions, where the pronoun is appended to the verb. The future tenses admit only *mesoclitic* contractions—a peculiarity of Portuguese—where the pronoun is actually inserted in the middle of the verbal suffix. In both cases, the pronoun is delimited by hyphens. For instance, *levantávamo-los* (*we raised them*) results from *levantávamos+os*, and *levantá-la-iam* (*they would raise her*) results from *levantariam+a*. Strictly speaking, in most cases these forms could be produced algorithmically from their component words. However, the contraction often entails non-trivial phonetic and orthographic changes to both components. Moreover, there are many irregular contractions, like *fi-lo* (*I did it*) resulting from *fiz+o*, which would have to be dealt separately. Therefore, we decided to explicitly represent all contracted forms in the automaton. The word *levantávamo-los* is encoded by

`levantávamo-los:1r(eab4:9ar,l2:o):9ar-o`

which denotes an enclitic form (`r`) with the first component being a verb (`e`) in indicative mode (`a`), imperfect past tense (`b`), first person plural (`4`), whose infinitive form is *levantar* (`9ar`), i. e. *levantávamos*. The second component is a pronoun (`l`) in its masculine plural form (`2`) whose standard form is *o*, i. e. *os*. (The last field `9ar-o` represents the non-existing standard or canonical form `levantar-o`, whose purpose is to provide an entry point for the reverse entries (see below).) Similarly, the mesoclitic form *levantá-la-iam* is encoded by

`levantá-la-iam:1s(eaf6:8ar,l3:o):8ar-o`

[1] For the sake of clarity, we are omitting some unessential details of the encoding we actually used. For instance, we used '`::`' instead of '`>`' to mark reverse entries, and '`#`' as an explicit end-of-string marker.

whereas *levantar-vos-iam* (*they would raise all of you*) would be

`levantar-vos-iam:1s(eaf6:8,l2:te):7te`

Portuguese also allows a number of triple contractions, consisting of a verb and two oblique pronouns, such as *dai-lhos* = *dai+lhe+os* (*give them to him*) and *mostrar-te-la-ei* = *mostrarei+te+a* (*I will show her to you*). These triple contractions were not included in our lexicon, since they are rarely used in Brazilian Portuguese, but they could have been handled in the same way as the double forms.

Finally, Portuguese also has a couple hundred non-verbal contractions—chiefly prepositions joined with articles, pronouns, or adverbs—such as *prum* = *para+um* (*for one*), *nesta* = *em+esta* (*in this*), *comigo* = *com+mim* (*with me*), *donde* = *de+onde* (*from where*), etc. For syntactic analysis, it is important that these words be replaced by their isolated components. Therefore, we used for them the same encoding method as for clitic verbs, e.g.

`nesta:1q(k:5em,l3:este):1e`

where '`q`' denotes a non-verbal contraction, '`l3`' a feminine singular pronoun, and '`k`' a preposition.

For each of these "analytic" entries, we also included a reverse "synthetic" entry, that allows us to quickly find a contracted form with desired verbal and pronominal inflections. Formally, all these inflected forms are viewed as deriving from a fictitious "canonical contraction," consisting of the infinitive form of the verb and the masculine singular pronoun, without any phonetic or graphic changes. For example, all conjugations of the verb *levantar*, contracted with any of the 3rd person pronouns *o*, *a*, *os*, *as*, are viewed as inflections of the fictitious form `levantar-o`. In particular, access to the inflected forms *levantávamo-los* and *levantá-la-iam* is provided by the strings

```
levantar-o>r(eab4,l2):4ávamo-los
levantar-o>s(eaf6,l3):4á-la-iam
```

The inclusion of all contracted forms adds more than a thousand entries for each verb. It should be noticed again, however, that regular verbs will have encodings with the same suffixes so that these entries will be factored, and will have relatively little effect on the automaton's size.

3.3 Invariant grammatical attributes

The entries described so far record only the general syntactic class of a word, and those attributes that are relevant to its inflection: gender, number, and magnitude for nouns and adjectives, tense and person for verbs, etc.

Syntactic analysis often requires other *invariant* properties, that apply to all inflections of the same word. In particular, the users of our Portuguese lexicon needed to know whether a given verb was *transitive direct* (admitted a direct object) and/or *transitive indirect* (admitted one or mere indirect objects). In the latter case, they also needed to know which prepositions are commonly used to introduce the indirect object(s). We encoded this information by a third type of string. For instance, the encoding

```
levantar@e:012(a,contra,de,em,para,sobre)
```

means that *levantar* as a verb ('`e`'), which can be intransitive ('`0`'), transitive direct ('`1`') or indirect ('`2`'), which can be followed by one of the prepositions *a*, *contra*, *de*, *em*, *para* or *sobre*.

3.4 Gender and number

In Portuguese, as in other Romance languages, many nouns can be inflected according to grammatical gender (*masculine* or *feminine*) and number (*singular* or *plural*): *gato/gata/gatos/gatas* (*cat/cats*). The same is true for most noun modifiers (adjectives, articles, and demonstratives), which must agree in gender and number with the base noun: *o gato branco/a gata branca/os gatos brancos/as gatas brancas* (*the white cat/the white cats*).

We represent those inflections by string encodings analogous to those of verbal forms. The words *gato* and *alto* (*tall*) produce the following entries:

```
gato:c1n:0        gato>c1n:0
gatos:c2n:1       gato>c2n:0s
gata:c3n:1o       gato>c3n:1a
gatas:c4n:2o      gato>c4n:1as

alto:b1n:0        alto>b1n:0
altos:b2n:1       alto>b2n:0s
alta:b3n:1o       alto>b3n:1a
altas:b4n:2o      alto>b4n:1as
```

Here the class code '`c`' identifies a noun, and '`b`' an adjective. The following digit '`1`' to '`4`' denotes the number/gender combination, and the '`n`' means these entries have *neutral* magnitude (see the next subsection).

3.5 Augmentatives and diminutives

Portuguese nouns can also be inflected in many semi-regular ways to qualify them with various attributes such as size, quality, endearment, etc. Thus, from the neutral term *gato* (*cat*) one gets the diminutive form *gatinho* (*small cat*, *kitten*), the augmentative forms *gatão* and *gatarrão* (*big cat*); the pejorative form *gataço* (*big bad cat*); and a few more.

Similarly, adjectives can be inflected to indicate intensity or degree: from *vermelho* (*red*) one can get the superlative *vermelhíssimo* (*very red*), the diminutive *vermelhinho* (*thoroughly red*, *pretty red*), *vermelhusco* (*reddish*), *vermelhão* (*vivid red*), etc.

These *magnitude* attributes are orthogonal to gender and number. In the Portuguese lexicon we built, the magnitude variants were considered additional inflections of the neutral word, and coded as follows:

```
gatinho:c1p:4o          gato>c1p:1inho
gatinhos:c2p:5o         gato>c2p:1inhos
gatinha:c3p:4o          gato>c3p:1inha
gatinhas:c4p:5o         gato>c4p:1inhas

gatão:c1o:2o            gato>c1o:1ão
gatões:c2o:3o           gato>c2o:1ões
gatona:c3o:2            gato>c3o:0na
gatonas:c4o:3           gato>c4o:0nas

gatarrão:c1o:5o         gato>c1o:1arrão
gatarrões:c2o:6o        gato>c2o:1arrões
gatarrona:c3o:6o        gato>c3o:1arrona
gatarronas:c4o:7o       gato>c4o:1arronas

altíssimo:b1r:6o        alto>b1r:1íssimo
altíssimos:b2r:7o       alto>b2r:1íssimos
altíssima:b3r:6o        alto>b3r:1íssima
altíssimas:b4r:7o       alto>b4r:1íssimas
```

The magnitude here is indicated by the third letter of the grammatical code: instead of '`n`' for normal, we used '`p`' for the diminutive, '`o`' for the augmentative, and '`r`' for the superlative.

In retrospect, however, we believe it was an error to view the magnitude variants as inflections of the neutral form. From the theoretical viewpoint, the relationship between *gatão* and *gato* (or even between *altíssimo* and *alto*) is much looser than that between *gatos* and *gato*, and has more to do with etymology, morphology, and semantics than with sentence-level grammar. From the practical viewpoint, our encoding above failed to take into account the existence of multiple augmentatives, diminutives, and superlatives. In particular, it does not record the fact that the plural of *gatão* is *gatões* and not *gatarrões*—since these two forms have the same grammatical code and the same standard form *gato*.

Our encoding also ignored many similar noun variants, including pejoratives like *casebre* (*shanty house*), and "oxymoronic" forms like *gatãozinho* (*little big cat*) and *palitão* (*big small stick*). It would have been better to view each magnitude variant as a different word; i.e. encode *gatinhos* and *gatinha* as inflections of *gatinho*, not of *gato*. The relationship between *gato* and *gatinho* could be encoded separately, as an inflection-independent attribute (see subsection 3.3).

4 Some statistics and implementation

The raw lexicon, collected and edited by M. V. G. Nunes and others [15], contained 1,490,002 distinct words, including 696,366 enclitic verb-pronoun contractions, 374,178 mesoclitic contractions, and 419,458 other word forms.[2] After appending the grammatical code to each

[2]It should be noticed that some contractions rarely used in Brazil were not included.

word (which had the side effect of disambiguating many homonyms), we got 2,242,112 direct (analytic) entries, and about that many reverse (synthetic) strings. Encoding the conjugation-invariant attributes of verb stems added another 13,467 entries, for a total of 4,500,688 strings.

The automaton built from these strings had 190,795 states and 346,534 valid transitions, i. e., less than two transitions per state, on average. By using a compact bit-coding of arcs and states, we were able to represent the whole automaton in only 1,115,160 bytes—i.e., about 0.25 byte per entry—, so that it could be kept resident in memory while queries are performed. It should be stressed that, in spite of this high compression ratio, the automaton can still be traversed very quickly (a few instructions per character).

All the experiments were carried out on top of a library of efficient routines for manipulating large acyclic automata, which we developed in the last few years. Most of the implementation was done in the language MODULA-3 [8, 16], developed at the Systems Research Center of the Digital Equipment Corporation. The main reasons for this choice were the simplicity of the language, its object-oriented programming paradigm, its availability on most UNIX workstations and its efficient implementation.

The most important module (abstraction) in our library is the one which implements reduced acyclic finite automata, and provides operations for inclusion, exclusion, enumeration and membership test of words, and for controlled enumeration of states, transitions and of prefixes and suffixes of a given state. The library also includes many auxiliary abstractions for handling texts, files, etc. Altogether, the basic library contains some 6,000 lines of source code.

In addition to this basic library, we implemented several auxiliary tools, specific for this application, mainly in order to generate and check all the encodings described above, starting with data which were prepared for us in a more complex format.

5 Conclusions

Our experiment shows that acyclic finite automata provide an efficient and compact implementation for a large lexical data base. One of the main advantages of this scheme is that irregular inflected forms do not need any exceptional treatment since their encodings follow the general rules.

It is quite clear that the same approach can be used for other languages in which inflected forms are generally obtained through prefix and/or suffix changes. This is the case of many Indo-European languages; simplified experiments with some of them show that similar results could be achieved. We believe that for many languages in which the changes are not confined to prefixes or suffixes it should be possible to devise encodings which "hide" this effect.

It should be also noticed that it is possible to use this system to implement conceptually higher formalisms such as the language DATR described in [7]. This implementation would allow various kinds of queries to be implemented very efficiently (see for instance [13]). As a matter of fact, an already implemented system like DATR can be used to generate the data needed to build the automaton.

Acknowledgments

The work reported in this paper was supported by Itautec-Philco S. A., by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). The data used for the case study were prepared by the team led by Prof. Maria das Graças Volpe Nunes, from the Institute of Mathematical Sciences of the University of São Paulo at São Carlos, SP.

References

[1] A. W. Appel and G. J. Jacobson. The world's fastest Scrabble program. *Communications of the ACM*, 31(5):572–578, 1988.

[2] L. Asker, B. Gambäck, and C. Samuelsson. EBL^2: An approach to automatic lexical acqusition. In *Proceedings of the International Conference on Computational Linguistics – COLING'92*, volume IV, pages 1172–1176, 1992.

[3] J. Bentley. A spelling checker. *Communications of the ACM*, 28(5):456–462, 1985.

[4] B. Bläser, U. Schwall, and A. Storrer. A reusable lexical database tool for machine translation. In *Proceedings of the International Conference on Computational Linguistics – COLING'92*, volume II, pages 510–516, 1992.

[5] C. Caligaris, A. Cappelli, M. N. Catarsi, and L. Moretti. An integrated environment for lexical analyses. In *Proceedings of the International Conference on Computational Linguistics – COLING'92*, volume III, pages 935–939, 1992.

[6] Jaime G. Carbonell and J. Siekmann, editors. *Machine Translation and the Lexicon*, volume 898 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1995.

[7] R. Evans and G. Gazdar. DATR: A language for lexical knowledge representation. *Computational Linguistics*, 22(2):167–216, June 1996.

[8] S. P. Harbison. *Modula-3*. Prentice Hall, 1992.

[9] L. Karttunen. Constructing lexical transducers. In *Proceedings of the International Conference on Computational Linguistics – COLING'94*, volume I, pages 406–411, 1994.

[10] L. Karttunen, R. M. Kaplan, and A. Zaenen. Two-level morphology with composition. In *Proceedings of the International Conference on Computational Linguistics – COLING'92*, volume I, pages 141–148, 1992.

[11] T. Kowaltowski, C. L. Lucchesi, and J. Stolfi. Application of finite automata in debugging natural language vocabularies. *Journal of the Brazilian Computer Society*, 1(3):5–11, 1995.

[12] T. Kowaltowski, C. L. Lucchesi, and J. Stolfi. Minimization of binary finite automata. *Journal of the Brazilian Computer Society*, 1(3):36–42, 1995.

[13] H. Langer. Reverse queries in DATR. In *Proceedings of the International Conference on Computational Linguistics – COLING'94*, volume II, pages 1089–1095, 1994.

[14] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software – Practice and Experience*, 23(1):15–30, 1993.

[15] R. T. Martins, R. Hasegawa, M.G.V. Nunes, G. Montilha, and O.N. Oliveira Jr. Linguistic issues in the development of ReGra: A grammar checker for Brazilian Portuguese. *Natural Language Engineering*. To appear.

[16] G. Nelson, editor. *System Programming with Modula-3*. Prentice Hall, 1991.

[17] D. Revuz and M. Zipstein. DZ: A text compression algorithm for natural languages. In A. Apostolico, M. Crochmore, Z. Galil, and U. Manber, editors, *Lecture Notes in Computer Science*, volume 644, pages 193–204. Springer-Verlag, 1992.

[18] E. Tzoukermann and M. Y. Liberman. A finite-state morphological processor for Spanish. In H. Karlgren, editor, *Proceedings of the International Conference on Computational Linguistics – COLING'90*, volume 3, pages 277–282, 1990.